UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE

Ph.D. Thesis

Vlad Paul Cosma

Declarative process models as explainable and verifiable Artificial Intelligence

Advisors: Prof. Thomas T. Hildebrandt (Principal) and Assoc. Prof. Tijs Slaats (Co-advisor)

This thesis has been submitted to the Ph.D. School of The Faculty of Science, University of Copenhagen in April 2024.

Abstract

As Artificial Intelligence-based decision support systems become an ever increasing part of our society, regulators have found it necessary to devise laws and guidelines on the ethical use of AI. In this light explainable AI has regained attention in the minds of researchers and businesses. Explainable AI has been with us for decades in the form of rule-based systems where high-level declarative rules are often implemented into more imperative or procedural behaviour-based code. Process mining has emerged as prominent AI field where the goal is to use process models to describe real-world business processes. The "white-box" qualities that make process models ideal at describing business processes also makes them prime candidates for use in decision support systems.

With the goal of enhancing model explainability and expressiveness, this thesis addresses selected aspects of the process mining pipeline, from data extraction to model verification, with a focus on enhanced trustworthiness, understandability, and verifiability of declarative process models. The objective is to advance the process mining field towards providing explainable Artificial Intelligence-based decision support systems.

To improve trustworthiness we propose a participatory design-based method for event log extraction that improves collaboration among stakeholders by extracting event logs that accurately and transparently capture business rules. To improve understandability we propose new algorithms to discover simpler, more understandable, declarative process models that take into account the users cognitive load. To enhance verifiability we propose two approaches tailored for the Dynamic Condition Response graphs declarative process notation. The first approach is based on a bisimilar mapping from Dynamic Condition Response graphs to Petri nets which preserves the declarative behaviour. This allows us to take advantage of mature model checking tools available for Petri nets. The second approach allows for the verification of select trace behaviour via test driven modelling. Specifically we check that we do not violate past modelling decisions when declarative models evolve over time.

Finally, to enhance expressiveness of mined declarative models we propose a timed extension for the declarative DisCoveR miner and show that verifiability is preserved by extending the bisimlar mapping from timed Dynamic Condition Response graphs to a timed variant of Petri nets.

Resumé

Kunstig intelligens (AI) og administrative beslutningsstøttesystemer baseret på AI bliver en stadig større del af vores samfund. På denne baggrund har det været nødvendigt at udarbejde love og retningslinjer for etisk og ansvarlig brug af AI. I dette lys har forklarbar AI (Explainable AI) fået ny opmærksomhed hos forskere og virksomheder. Forklarbar AI har været hos os i årtier i form af såkaldte symbolske AI systemer, baseret på deklarative regler og logik. I praksis er beslutningsstøttesystemer dog ofte en blanding af regler, kode og andre AI-modeller som til sammen udgør et uforklarligt system.

Process mining er opstået som et fremtrædende AI-område, hvor målet er at bruge AI til automatisk at generere ("process mine") procesmodeller fra hændelses-logs ("event logs") fra forretnings- og beslutningsprocesser. De kvaliteter, der gør procesmodeller ideelle til at beskrive forretningsprocesser, gør dem også til de gode kandidater til brug i forklarbare beslutningsstøttesystemer.

Med det formål at forbedre forståeligheden og udtryksfuldheden af procesmodeller, behandler denne afhandling udvalgte aspekter af process mining pipelinen, fra dataudtrækning til modelverifikation, med fokus på forbedret pålidelighed, forståelighed og verificering af deklarative procesmodeller. Målet er at fremme procesminedriftsområdet mod at tilbyde forståelig kunstig intelligensbaseret beslutningsstøttesystemer.

For at forbedre pålideligheden foreslår vi en metode baseret på brugerdeltagende design (participatory design) til udtrækning af hændelseslogfiler, der forbedrer samarbejdet mellem interessenter i forbindelse med at udtrække hændelseslogfiler, der nøjagtigt og transparent beskriver hændelser i organisationen. For at forbedre forståeligheden foreslår vi nye process mining algoritmer til at opdage simplere, mere forståelige, deklarative procesmodeller, der tager hensyn til brugernes kognitive belastning. For at forbedre verificeringen foreslår vi to tilgange tilpasset den deklarative procesnotation af Dynamic Condition Response (DCR) Grafer. Den første tilgang er baseret på en formel oversættelse fra Dynamic Condition Response Grafer til Petri net, som bevarer semantikken af processen. Dette tillader os at drage fordel af modne verifikationsværktøjer, der er tilgængelige for Petri-net. Den anden tilgang tillader verificering eksempler på beslutningsforløb af via testdrevet modellering. Specifikt kontrollerer vi, at vi ikke overtræder beslutningsforløb,

når deklarative modeller udvikler sig over tid.

Endelig, for at forbedre udtryksfuldheden af deklarative modeller der genereres ved hjælp af process mining foreslår vi en tidsmæssig udvidelse for den deklarative DisCoveR-miner og viser, at verificeringen bevares ved at udvide den semantik-bevarende oversættelse fra Dynamic Condition Response Grafer med tid til en variant af Petri-net med tid.

Acknowledgements

It has been an honour to be surrounded by so many bright minds and supportive people throughout my PhD, but also many years before it. This work would not have been possible without them.

My thanks go out to

My supervisors Thomas Hildebrandt and Tijs Slaats who guided me throughout the PhD, whose ideas, knowledge and insights made sure I was on the right path, whose friendship and support I came to value deeply, words cannot express my gratitude.

The PhD Committee members: Dmitriy Traytel, Barbara Weber and Claudio De Ciccio for their timely and valuable feedback.

Colleagues present & past at the University of Copenhagen: Hugo López, Axel Christfort, Asbjørn Flügge, Naja Møller, Boris Düdder, Christoffer Bach and all other researchers and collaborators I've met when I started my PhD within the section and within the EcoKnow project.

New faces & new ideas; colleagues I've met while being a visiting researcher at Utrecht University: Hajo Reijers, Xixi Lu, Suhwan Lee, Andrei Buliga, Vinicius Dani, Wouter van der Waal.

Inspiring people at KMD especially Peter Damm, who made my PhD possible, the KMD Momentum team, and my former employer Lars Christensen at Sensus, who back in 2016 gave my first insights into what it means to publish academic work and showed me how Computer Science research can truly have a beneficial impact on society.

My parents Gabriela and Marin, who listened to my frustrations when things got tough and were always there when I needed support, and my friends who provided me with much needed respite and laughter to keep going.

This work was partially funded by Innovation Fund Denmark EcoKnow.org Project, Independent Research Fund Denmark PAPRICAS.org Project, KMD and Copenhagen University.

Table of Contents

Part I

Overview

Chapter 1

Introduction

Quote The book "Principles of Model Checking" by Baier and Katoen states on page 8 the given fact that "Any verification using model-based techniques is only as good as the model of the system" [18]. To contextualize the work of this thesis, we wish to extend this fact with a follow-up intuition that

> "Any decision using AI-model-based techniques is only as good as the learned model (and data) of the system."

The vision of Artificial Intelligence The term AI, short for Artificial Intelligence, has been coined as a term in 1956 [111] and its definition is constantly evolving, but in essence always existed at the boundary between human decision making and computational theory. The machine learning field, which is an example of sub-symbolic AI field, is the most prominent field where data is used to learn models. Given an untrained model f, a training set X and a target y the goal is to learn the model $\hat{f}$ which given X predicts $\hat{y}$ and the choice of coefficients of $\hat{f}$ is made such that it optimizes a metric. In supervised learning an example metric is the error between the target y and the predicted $\hat{y}$. In essence the model learning task is $\hat{f}(X) = \hat{y}$. In machine learning popular model choices for f can be variants of decision trees, neural networks or linear models. Optimizing on a well defined metric is a comparatively easy task when put in relation to understanding the structure or properties of the learned model $\hat{f}$. That is because it does not necessarily capture human understandable concepts or symbols, it captures sub-symbolic properties of the training data which are dependent on the model type and

its training hyperparameters.

Predictive performance increased as training data became abundant and as fast hardware made it feasible to work with ever increasing and more complex datasets. These larger datasets and better computing power allowed for training larger and more complex models (random forests, deep learning, deep reinforcement learning). The rapid advancement in sub-symbolic AI techniques came at the price of explainability. Explainable Artificial Intelligence (xAI) [62] has emerged as a necessary step forward and is known for taking well studied techniques and theory from statistics (LIME [109]), game theory (SHAP [83]) and other fields, to provide explanations for the learned models' decisions and data. However xAI is seen as a post-hoc technique, as explainability in not considered, in general, when optimizing a model for predictive performance. With the enforcement of GDPRs "Right to explanation" [61] and the upcoming EU "AI Act" [37] explainability becomes a key factor in the successful adoption of learned models in decision support systems.

Knowledge-based systems If we instead shift our focus on knowledge-based or rule-based systems as an example of symbolic AI, which have been with us since 1969 [111], we see a different story. Rule-based decision support systems are explainable by design as they rely on high-level symbolic representations of data, a logic (defined by symbols, syntax, semantics) and employ search algorithms in the model state-space to make decisions or verify properties of the model. The declarative approach to building a knowledge-based system is to tell the system the rules of the data. In contrast the procedural approach encodes the knowledge from the data directly as system behaviour. When designing such an AI system the key to success is to embed "both declarative and procedural elements in its design, (based on the idea) that declarative knowledge can often be compiled into more efficient procedural" [111] systems. In rule-based systems, explanations can then be limited to a proof that a certain property always holds or an explanation of a fault (the sequence of steps leading to a property being violated). By symbols we refer here to high-level concepts that are understandable for a human domain expert.

Models As the model sits at the core of both learning and verification we now take a look at what a model is. Models are mathematical abstractions of

the real world. Models can be represented as simple mathematical equations ($a + b = c$), trained deep neural networks ($\hat{f}$) or as a set of rules.

In rule-based systems the role of the model is to abstract away specific implementation details related to the programming language syntax or data structures, while still describing its behaviour accurately (either as procedural flows of the state space or declarative constraints). In practice it should be straightforward for a domain expert to create the model-based on a given specification (documentation, requirements, laws, measurements, tolerances). At the same time model checking allows us to verify that the model satisfies the specification. Because model checking assumes the creation of models and specifications by domain experts, it also implies that efforts should be made towards having human understandable models and specifications. In model checking, models of the state-space are commonly represented as transition systems, automata or Petri nets. Rules, or specifications, are commonly represented as a formal logic (Linear Temporal Logic, Computational Tree Logic) or as desirable model properties (liveness, safety, fairness).

Process Mining and Business Process Management We are particularly interested in models describing the behaviour of business processes and testing specifications related to their performance. A business process is defined in the book "Business Process Management" by Weske [135] as a set of activities that are performed in coordination to realize a business goal. The field of "Business Process Management (BPM) includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes" [135]. A new definition of process science [129] has been coined where before only data science existed, and even before it there was data mining (which is now a less used term). At the intersection of process science and data science we find process mining. The discipline of process mining was started by van der Aalst and defined as: "Process mining brings together traditional model-based process analysis and data-centric analysis techniques" [129]. "Process mining aims to improve operational processes through the systematic use of event data" [4]. In relation to this thesis process mining sits at the intersection of learning explainable process models from data and making decisions based on those models.

The set of process mining techniques [4] is constantly expanding, here

we enumerate just a few. We have process discovery techniques, deterministic [17], fuzzy [63], probabilistic [24], hierarchical [82]. We have event abstraction techniques [138]. We have formal methods for model checking [72], conformance [30], alignments [31]. We have patterns to solve data quality issues [123]. We have scoring techniques based on conformance checking derived metrics [129], or based on empirically validated complexity metrics [5] or entropy metrics based on information theory [16]. The underlying assumption is that a process model should capture the decision space of the data while at the same time remain explainable and verifiable.

Process models The process model mined from data is a beast unlike any other, which in a single structure at the right abstraction level aims to capture all the data it is trained on (mined from), be concise and understandable enough for domain experts to make decisions based on it, while at the same time being verifiable for certain well-behavedness properties and accurately capture real-world behaviour.

A process model is an abstraction of a real-world process. It comes in two main flavours: imperative (describes the flow of activities) and declarative (describes how the process behaves). Popular imperative model notations are Directly Follows Graphs (DFGs) and the Business Process Modelling Notation [54] (BPMN). DECLARE [102, 103] is a popular declarative notation with rules based on Linear Temporal Logic [55] (LTL) on finite traces (LTLf) [47], a model being defined as a conjunction of rules. Petri nets [104] can be modelled in a declarative manner, but are more often seen as imperative models. Petri nets are the most popular models used in process mining because they are at a high enough level of abstraction to describe real-world behaviour, all modelling notations accept some form of mapping, synthesis or transformation to and from Petri nets, and mature tools exist to verify Petri nets or transform them to Automata and Transition Systems that can be used for verification [72, 113, 136].

Dynamic Condition Response (DCR) Graphs [67] are a relatively new process modelling notation. Recent works on process discovery [17], alignment [34] and transformation [44] are making them theoretically and practically interesting. Improved tool support via pm4py-dcr [42], dcr-js [81] and the dcrgraphs.net portal [98] are making it easier to get started with DCR graphs. DCR graphs are part of the same family of declarative notations as DECLARE. Similar to DECLARE, DCR graphs describe system behaviour

a set of rules/constraints. Similar to Petri nets, DCR graphs describe states as a marking on events.

Manufacturing or supply chain processes naturally lend themselves towards a more imperative view because they are highly structured. By contrast knowledge intensive processes such as healthcare or case work, where citizens/people are the case object, lend themselves better towards a more declarative view [50]. These are unstructured highly flexible processes, where ideal paths through the process are highly dependent on the case type or on certain case attributes, and are regulated to some extent (by laws, policies, guidelines).

Process discovery Process discovery is the task of mining a process model from data. Given a specific process modelling syntax p and a training dataset X the goal is to discover a model $\hat{p}$. In essence the process discovery task is $\hat{p}(X) = \hat{y}$, which in machine learning is defined as one class classification [95]. Notice that unlike our initial example we are not given a target y. That is because we consider our training dataset as only one class of desired behavior or positive examples. $\hat{y}$ will only tell us if the instances of our training dataset X fit the desired behavior or not. A key difference between the process discovery and machine learning approaches to one class classification is the goal of discovering a model. In process discovery the model $\hat{p}$ is the goal, whereas in machine learning the model output $\hat{y}$ is the goal. This results in process models that are explainable by design, they provide both predictive power and explainability. The predictive power comes from the analyses and simulations done on top of the process model. Furthermore their formal definitions makes certain properties, such as livelock or deadlock freedom, verifiable. Still, as a one-class classification task process discovery suffers from the same challenges. In machine learning one attempts to encode the dataset such that a well defined metric can optimize the model using minimization (or maximization). Such a general encoding into an numerical optimization problem is not necessarily achievable for process models. However process models have other properties which can be exploited when evaluating process discovery.

The general task of evaluating process discovery algorithms needs to take into account the expertise level of the user, the familiarity of the user with the specific process model notation, and the goal of the process model in relation to the decision support it needs to offer [4]. We limit our scope to

only consider the evaluation of process models in the context of a specific dataset. As process models have formal definitions and semantics one can define metrics in terms of the traces or language they accept (conformance or alignment-based metrics). Because process models are also meant for human inspection one can define metrics based on the number of elements and their connectiveness. Metrics such as precision and fitness need to be counterbalanced by generalization and simplicity [4, 129].

The core type of dataset X used in process mining is an event log. An event log is a specific type of data structure well suited for the storage of business process instances. It contains a notion of a case and a set of activities, together with other optional case and activity related attribute. A very common activity related attribute is the time when the activity happened. Most information systems have relational databases as their data structures and techniques exist to project these databases to event logs.

One concrete instance of a process discovery algorithm is DisCoveR [17], a miner for DCR graphs. It first creates a log abstraction and then uses a subset of DECLARE patterns to find rules. The rules are deterministic and based on set operations on the event log. The discovery algorithm returns perfectly fitting models for a given event log. However the mining notions from extended definitions of DCR graphs, such as activity groups (also known as nestings) [66] have not been considered. Therefore the mined models contain a large number of relations. With activity groups the model fitness stays the same, but visually the size of the model is reduced. Work done in [5] empirically evaluates understandability in terms of four complexity metrics. We use these metrics in our work as a proxy for understandability.

Structure The remaining of this thesis is structured as follows. In Part I Section 1.1 we continue with a presentation of the overall research problems being tackled and the objective of the thesis. We briefly mention related work in Section 1.2 and provide the definitions of the concepts used throughout this thesis in Section 1.3. A list of publications and code references are given in Section 1.4. In Chapters 2 to 5 we then give the motivation and summary of each contributing paper, we expand with a discussion section where relevant. We round off part I with a conclusion and possible avenues for future work in Chapter 6. In Part II we start with trustworthy event log creation from information systems in Section 7.1, then present understandable declarative process models through the discovery of activity groups in Section 8.1.

Next we look at verifiable DCR graphs by a transformation to Petri nets in Section 9.1, and through static and dynamic techniques for test driven modelling Section 9.2. We conclude Part II with a section introducing time mining for declarative process models in Section 10.1 and present a transformation from timed DCR graphs to timed arc Petri nets in Section 10.2.

1.1 Research problems

From data to insights and verification The work of this thesis handles the entire pipeline from trustworthy event log extraction to mining understandable declarative process models from event logs, enhancing those process models and finally using the process models to verify properties of the data.

We treat explainability in a narrow sense and in relation to process mining. Concretely explainability means increased trustworthiness, understandability and verifiablility. To increase trustworthiness in the process models used by domain experts, one approach is increased transparency and reduced uncertainty in the event log creation stage. When it comes to creating trustworthy event logs we ask: How can we ensure that our event logs capture the relevant information? Out of many tables from a relational database we rely only on a few. Given the potential data quality issues we require domain knowledge to fix them. We also require domain knowledge to define the goal of our process model. We require knowledge of information system usage patterns to correctly connect our process mining goals with the right data that can provide answers. In an attempt to mitigate these challenges in Section 7.1 we present the BERMUDA method and show how it can be applied to data from knowledge intensive processes such as healthcare and case work.

One view of understandability of declarative process models is in relation to their number of elements and the type of relations. Previous research [5] has empirically validated such insights and produced a set of process model quality metrics as a proxy to user understandability. We create three extensions of the DisCoveR miner with the aim of reducing the number of relations at the cost of introducing activity groups. In Section 8.1 we show that overall the extensions have a positive impact on the quality metrics.

Trust can be built through evidence of correct behaviour. To this end we aim to increase the verifiability of declarative process models. We provide

two avenues towards a DCR graph model checker. One approach presented in Section 9.1 relies on a transformation to Petri nets which is a bisimlarity preserving mapping. This enables us to use mature model checkers that have an established track record of good performance. Another detailed in Section 9.2 relies on the concept of open tests and alignments to directly check positive and negative traces against a DCR graph given a specific context.

Finally we are able to increase the expressiveness of our process models by adding time dimensions. To be able to verify time properties of our real-world process we mine timing constraints and show how the enhanced timed DCR graph is transformed to a timed arc Petri net. This work is detailed in Section 10.1 and Section 10.2.

Objective Overall the objective of the thesis is to push towards explainability the various declarative process mining concepts, from trustworthy event log extraction, through to the discovery of understandable declarative models and verification of DCR graphs.

1.2 Related work

1.2.1 Extracting event logs

To improve the trustworthiness of our event logs we have to get an overview of the data quality issues [13, 22, 23, 56, 59, 75, 79, 80, 112, 123] and event abstraction [29, 125, 138] challenges faced when extracting data for secondary use from primary sources. The most complete description can be found in the chapter "Foundations of Process Event Data" [3] in the Process Mining Handbook [4]. The same work [4] provides insights into the unique challenges of knowledge intensive processes, such as the healthcare domain [7, 91, 120] where domain experts are an invaluable resource [64]. This can be found in the chapter on "Using Process Mining in Healthcare" [90] as well as in [99, 110]. Finally in the chapter on "Responsible process mining" [87] of the same book [4] we want to highlight how accuracy, traceability and transparency are presented as major challenges for the interpretability of process mining results, although process models are in general regarded as "white-box" approaches.

1.2.2 Process discovery and hierarchical models

To improve understandability of the discovered models requires us to understand the patterns of rules or flow found by the discovery algorithms. Afterwards we can consider improving their understandability while balancing the trade-offs between simplicity and accuracy. We see discovering groupings in DCR graphs as one approach to reduce visual model complexity.

1.2.2.1 Declarative

Several notations for declarative process modelling have been developed. The most complete works introducing Dynamic Condition Response (DCR) graphs are the PhD Theses by Slaats [118] and Mukkamala [96]. Out of the existing DCR miners [17, 49, 100] we use the DisCoveR miner [17] as it outputs the most accurate models. Evidence for its accuracy is given by winning the Process Discovery Challenges in 2021 and 2023. The DisCoveR miner is the foundation for the group and choice algorithms presented in Section 8.1 as well as for mining delays and deadlines in Section 10.1. Examples in Sections 4.1, 5.1 and 5.2 also use DCR graphs mined with DisCoveR as the starting point for the transformation to Petri nets.

In addition to DCR graphs, the DECLARE [103] and Guard-Stage-Milestone [71] (GSM) notations have also seen broad use in the business process management research community. DECLARE provides a set of templates for modelling business constraints that are formalised as Linear Temporal Logic formulae (parameterized by activities). A DECLARE model is the conjunction of a set of instantiated formulae. The GSM notation [71] takes a declarative data-centric approach to modelling processes, where stages of activities in the process are connected through guards that need to be satisfied for their activation and milestones that represent their acceptance criteria.

DECLARE Miner [84] and MINERful [36] are the most known miners for DECLARE. We provide a DECLARE model mined with MINERful in Section 4.1 where we compare its synthesized Petri net with our transformation from DCR graphs. Other declarative notations such as, Process Intermediate Language (DPIL) and its derivation Case Management Model And Notation (CMMN) have little to no support when considering the process discovery task, we can only mention [115] for DPIL and [106] for GSM.

Haisjackl et al. [65] investigate hierarchy in declarative process models in an exploratory study, but they do not measure the effect it has on cognitive

load.

1.2.2.2 Imperative

Several metrics both for complexity (such as simplicity and control flow complexity - CFC) and performance (fitness, precision and generalization), have been used to evaluate imperative miners: Inductive Miner [24, 77], HM Miner [46, 132, 133], ILP Miner [51], ETL Miner [25], Prime miner [20], eST miner [86], DiSCover [131](not to be confused with the earlier mentioned DisCoveR miner for DCR graphs) and Split Miner [14]. Augusto et al. [15] find a correlation between complexity metrics and the quality of process models mined with imperative miners.

Discovering hierarchies or subprocesses is well studied for imperative process model notations [38, 39, 73, 76, 89, 124]. FlexHMiner [82] is a miner for hierarchical process models that discovers subprocesses based on three different methods, using domain knowledge, random clustering, and a flat tree, with the domain knowledge approach being on average the highest-scoring one. In the context of multi-level event logs, mining hierarchical process models [78] means discovering hierarchies in logs where each part can be mined with its own miner resulting in a combination of several process model notations, including DECLARE. Smirnov et al. [116] systematically catalog business process model abstraction techniques and show their value through use cases. Turetken et al. [127] do not investigate cognitive load, but the related concept of understandability. In particular they find that "Fully-flattened models are perceived easier to understand than models that are vertically modularized (using groups or sub-processes)" [127].

1.2.3 Transforming between formalisms

When transforming between different modelling formalisms we gain access to a whole new set of tools and techniques to analyze our processes with. Transforming from a declarative rule-based formalism to an imperative procedural one has the added benefit of allowing us to model a system by essentially "telling it what it needs to know" [111], often at the expense of an exponential state space increase in the imperative model. We rely on proofs that certain properties hold, such as the level of concurrency, and that the systems are to some extent behaviourally equivalent.

DECLARE DECLARE has been formalized in other languages such as coloured automata [85] and SCIFF [92, 93]. Mappings from DECLARE to Petri nets and R/I-nets were provided respectively in [107] and [48], however proofs of correctness are missing from each of these.

GSM A mapping has been proposed from Petri nets to GSM [105], in particular with a focus on representing the output of process discovery algorithms (which usually produce Petri nets) as GSM models. We are not aware of any direct mappings in the opposite direction. Similarly [57] provides a mapping from DCR graphs to GSM models, an opposite mapping is mentioned as future work but has not yet materialised.

DCR In [68] a subset of the DCR relations and their equivalent Petri net mapping is presented, without inhibitor arcs and without proof of correctness. [97] provides an encoding of DCR graphs as Büchi automata.

Petri nets Petri nets are widely used, therefore there are many translations to notations outside the declarative process modelling sphere, for example Ladder Logic Diagrams [126], Timed Automata [27] and mCRL2 [108].

Much work has gone into mapping other modelling notations into Petri nets, such as UML activity diagrams [122], UML sequence diagrams [137], UML state charts [70], and BPMN [52, 108]. Different classes of ω-language Petri nets have been introduced in [128] and their complexity has been studied in [58]. The definition of acceptance criteria for infinite words in [128] is based on markings being visited infinitely often, similar to the acceptance criteria of Büchi-automata.

1.2.4 Test-driven modelling

Within process mining the most common verified behaviour is that of comparing control-flow models against the observed behaviour through conformance checking [4]. The term trace alignment [74] was introduced as a preprocessing step to other process mining techniques. These days trace alignment primarily refers to the alignment of traces against model behaviour and is seen as a cornerstone of conformance checking.

Test-driven modelling (TDM) was introduced by Zugal et al. in [140, 141] as an application of test-driven development to declarative business processes.

Their studies [140] indicate in particular that simple sequential traces are helpful to domain experts in understanding the underlying declarative models. Connections between refinement, testing-equivalence and model-checking was observed in [26]. Test-driven modelling for declarative models has also been identified as a form of hybrid business process representations [8], falling into the category of representations that combine a declarative model with (more imperative) tool support to aid their understandability [9, 11]. In particular the effect of hybrid approaches towards improving the understandability of DCR graphs, which are closely related to the test-driven modelling approach presented here, was investigated by Andaloussi et al. in [10, 12].

1.2.5 Handling time

Mining timing information has been done on all major process modelling formalisms to increase their expressiveness by capturing temporal constraints: DECLARE [134], time BPMN [32, 33, 60], timed Petri nets [1, 139] and Timed Automata [40]. In [60] lead and lag time are equivalent to the minimum delay and maximum deadline. In [32] the authors mine extraneous delays. [1] mines probabilistic delays on Stochastic Petri nets. Work in the [2] shows how durations are overlayed on Petri nets on a per case basis. [40] exemplifies how one can use the UPPAAL statistical model checker [45] on Timed Automata.

1.3 Preliminaries

We now give the standard definitions that will be used throughout this thesis. The aim is to give a formal overview and help in better understanding the remaining chapters in Part I. Note that each paper in Part II will have in its preliminaries the necessary definitions for the contribution.

1.3.1 Event logs and traces

"An event log is a collection of events" [4]. Event logs are the core data structure of process mining. They can be seen as a projection of any data source, typically relational databases, into this standardized event log format. They are stored in the eXtensible Event Streams (.xes) standard format, but also in tabular form as comma separated values (.csv). When considering the

tabular form an event is in essence a row in the table. Event logs have a notion of a case (or process instance), "a case consists of events such that each event relates to precisely one case" [129]. Event logs also have a notion of activities. The sequence of activities is what gives us the case. Often activities are also associated with a timestamp. When a timestamp is present it is normally what defines the sequence of activities. We now give the formal definition of an event log.

Definition 1.1. *An event log over a given set of activities A and time domain TD being ISO8601 time stamps is defined as $L = (E, C, \alpha, \gamma, \beta, \succ)$ where:*

(i) E *is a finite set of events,*

(ii) C *is a finite set of cases (process instances),*

(iii) $\alpha : E \to A$ *maps each event to an activity,*

(iv) $\gamma : E \to TD$ *maps each event to a timestamp,*

(v) $\beta : E \to C$ *maps each event to a case and is surjective,*

(vi) $\succ \subseteq E \times E$ *is the succession relation, which for every case* $c \in C$ *is a total ordering on the set* $\beta^{-1}(c)$

Given an event log L we define the trace function $Tr_L : C \to \mathcal{P}(E)$ by $Tr_L(c) = \{e \mid \beta(e) = c\}$, i.e. returning all events belonging to the same case. We require that for events belonging to the same case the succession relation respect the time ordering, i.e. $\forall c \in C, \forall e_1, e_2 \in Tr_L(c)$ if $e_1 \succ e_2$ then $\gamma(e_1) \leq \gamma(e_2)$. We assume a function $\Delta : TD \times TD \to \omega$ returning an absolute number of time steps between two timestamps.

An event log we commonly refer to in the summaries is the Road Traffic Fine Data Management Process (RTFMP) [88] event log. The event log was extracted from an information system used for the handling of road traffic fines by the Italian police.

1.3.2 Process models

We are particularly concerned with the definition of two types of process models: DCR graphs and Petri nets.

1.3.2.1 DCR graphs

In this section we summarize the various DCR graphs definitions from [96, 118]. Formally, we define DCR graphs as attributed directed graphs. We deviate slightly from the original presentation in [69] to be consistent with later articles and define slightly more general graph structures. The majority of papers on DCR graphs use the term events for activities but here we use the term activities instead of events[1]. This is to be consistent with the definition of activities in event logs.

Core DCR

Remark 1.1. *Notation: For a set E we write $\mathcal{P}(E)$ for the set of all subsets of E, i.e. the powerset of E and $\mathcal{P}_{ne}(E)$ for the set of all non-empty subsets of E.*

Definition 1.2. *A* core DCR graph *G is given by a tuple $(E, M, R, @, L, l)$ where*

(i) *E is a finite set of* activities

(ii) *$M = (Ex, Re, In) \in \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$ is the* marking

(iii) *$R \subseteq E \times E$ is the set of* relations *between activities*

(iv) *$@ : R \to \mathcal{P}_{ne}(\{\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\})$ is the* relation type assignment

(v) *L is the set of action labels*

(vi) *$l : E \to L$ is the* labelling function *assigning an action label to each activity*

The marking $M = (Ex, Re, In)$ describes the state of an activity e in the following way. If e has been executed *at least once then $e \in Ex$. If e is* pending *(i.e. it must eventually be executed) then $e \in Re$. If e is* included *(i.e. it is currently relevant) then $e \in In$.*

The relation type assignment assigns one or more relation types to each edge. If $\rightarrow\bullet \in @(e, e')$ we say that there is a condition *from e to e'. If $\bullet\rightarrow \in @(e, e')$ we say that there is a* response *from e to e'.*

[1]Note that in part II there are definitions which still use the term events, therefore we make explicit the naming convention used in each paper.

If $\rightarrow+ \in @(e, e')$ *we say that there is an* inclusion *from* e *to* e'*. If* $\rightarrow\% \in @(e, e')$ *we say that there is an* exclusion *from* e *to* e'*.*

We will often write $e\mathcal{R}e'$ *or* $e\mathcal{R}e' \in R$ *for* $\mathcal{R} \in @(e, e')$*.*

Note that when we have $e\mathcal{R}e$ a relation between the same activity we call $\mathcal{R}$ a self relation. A common self relation is the self exclude relation $e \rightarrow\% \ e$.

Core DCR semantics The semantics below are adapted from Section 9.2 by removing the milestone relation and the roles.

The enabledness of an activity can be determined by looking at the marking of events immediately related to the activity by an incoming condition relation the graph. To define enabledness formally we will use the following notation. When G is a DCR graph, we write, e.g., $\mathsf{E}(G)$ for the set of activities of G, $\mathsf{Ex}(G)$ for the executed activities in the marking of G, etc. In particular, we write $\mathsf{M}(e)$ for the triple of boolean values $(e \in \mathsf{Ex}, e \in \mathsf{Re}, e \in \mathsf{In})$. We write $(\rightarrow\bullet\, e)$ for the set $\{e' \in \mathsf{E} \mid e' \rightarrow\bullet\ e\}$, write $(e \bullet\!\!\rightarrow)$ for the set $\{e' \in \mathsf{E} \mid e \bullet\!\!\rightarrow e'\}$ and similarly for $(e \rightarrow+)$ and $(e \rightarrow\%)$.

Definition 1.3 (Enabled activities)**.** *Let* $G = (E, \mathsf{M}, R, @, L, l)$ *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. An activity* $e \in \mathsf{E}$ *is* enabled*, written* $e \in \mathsf{enabled}(G)$*, iff (a)* $e \in \mathsf{In}$ *and (b)* $\mathsf{In} \cap (\rightarrow\bullet\, e) \subseteq \mathsf{Ex}$*.*

That is, enabled activities (a) are included and (b) their included conditions have already been executed.

The effect of executing an activity only changes the marking of the executed activity and activities related to it by outgoing response, inclusion or exclusion edge from the executed activity.

Definition 1.4 (Effect)**.** *Let* G *be a DCR graph with marking* $\mathsf{M} = (Ex, Re, In)$*. The effect of executing an enabled activity* e *is the marking* $\mathsf{effect}_G(\mathsf{M}, e) = (Ex', Re', In')$ *where*

$$\begin{aligned} Ex' &= Ex \cup \{e\} \\ Re' &= (Re \setminus \{e\}) \cup (e \bullet\!\!\rightarrow) \\ In' &= (In \setminus (e \rightarrow\%)) \cup (e \rightarrow+) \end{aligned}$$

We write $\mathsf{execute}(G, e)$ *for the DCR graph obtained by replacing the marking of* G *with* $\mathsf{effect}_G(\mathsf{M}, e)$*. We write* $G \rightarrow_e G'$ *for* $G' = \mathsf{execute}(G, e)$*.*

Definition 1.5 (Accepting)**.** *Let* G *be a DCR graph, with marking* $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. We say that* G *is* accepting*, written* $\mathsf{accepting}(G)$*, iff* $\mathsf{In} \cap \mathsf{Re} = \emptyset$*.*

Definition 1.6 (Runs)**.** *A sequence of events ϕ is a* run *of a DCR Graph iff $G \rightarrow^*_\phi G'$. It is an* accepting run *iff* $\mathsf{accepting}(G')$*. The* language *of G_0 is then the set of all such accepting runs.*

We omit the execution semantics for all but the core DCR graphs Definition 1.2 as they are introduced for the extended definitions in each paper from Part II where relevant.

Extended DCR The *extended* version of the *core* DCR graph definition contains two extra relations in the set of relation types, the milestone ($\rightarrow\!\!\diamond$) and the no-response ($\bullet\!\!\rightarrow\!\!\times$).

Definition 1.7. *An* extended DCR graph *G is given by a tuple $(E, M, R, @, L, l)$ where*

(i) *$(E, M, R, @, L, l)$ is a core DCR graph*

(ii) *$@ : R \rightarrow \mathcal{P}_{ne}(\{\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\} \cup \{\rightarrow\diamond, \bullet\!\!\rightarrow\!\!\times)\}$ is the* extended relation type assignment

For the relation type assignment we add two extra relations. If $\rightarrow\diamond \in @(e, e')$ we say that there is a milestone *from e to e'. If $\bullet\!\!\rightarrow\!\!\times \in @(e, e')$ we say that there is a* no-response *from e to e'.*

Role DCR

Definition 1.8. *A* role DCR graph *is a tuple $(E, M, R, @, L, l, \mathsf{by}, \mathsf{Rl})$*

- *$(E, M, R, @, L, l)$ is an extended DCR graph*
- *L is the set of* action labels
- *$\ell : E \rightarrow L$ is the* labelling function*, assigning action labels to activities*
- *Rl is a set of roles*
- *$\mathsf{by} : L \rightarrow \mathcal{P}(\mathsf{Rl})$ is the* role assignment function*, assigning zero or more roles to activities.*

Group DCR Here we define a DCR graph with only activities A. We omit the set of action labels and the labelling function, because l is the identity function, i.e. the finite set of activities E is equal to the finite set of action labels L. Defining a DCR graph with activities also has the advantage of creating a one to one correspondence with the activities defined for event logs. It is a more intuitive definition when discovering DCR graphs from event logs.

Definition 1.9. *A* DCR graph G with nested groups of activities *is given by a tuple* $(A, M, R, @, A_G, \rhd)$ *where*

(i) $AG = A \uplus A_G$ *is a finite set of* activities A *and* activity groups A_G,

(ii) $M = (Ex, Re, In) \in \mathcal{P}(A) \times \mathcal{P}(A) \times \mathcal{P}(A)$ *is the* marking,

(iii) $R \subseteq AG \times AG$ *is the set of* relations *between activities*

(iv) $@ : R \to \mathcal{P}_{ne}(\{\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%\})$ *is the* relation type assignment

(v) $\rhd$: $AG \rightharpoonup A_G$ *is a partial* grouping *function.*

We write $>$ *for* $\rhd^+$ *(the transitive closure of* $\rhd$*) and require that it is irreflexive. We write* $\geq$ *for reflexive closure of* $>$ *and* $\leq$ *for the inverse of* $\geq$.

A nested group defines a disjoint set of activities (an activity or activity group can be part of exactly one activity group). Another type of grouping in DCR graphs is a subprocess [101]. The key difference between nested groups and subprocesses is that groups do not have execution semantics as they are syntactic shorthand for the process model, i.e. any relation that applies to the group applies onto each activity or activity group that is part of the group and this definition is applied recursively.

Timed DCR

Remark 1.2. *Notation: Let* $\mathcal{N}$ *refer to the set of natural numbers (including zero) and* $\mathcal{N}_\omega = \mathcal{N} \cup \{\omega\}$ *refer to the set of natural numbers (including zero) and infinity (with infinity written as* ω*).*

Definition 1.10. *A* Timed DCR graph G *is given by a tuple* $(E, M, R, @, L, l)$ *where*

(i) E *is a finite set of* activities*,*

(ii) $M = (Ex, Re, In) \in \big((E \rightharpoonup \mathcal{N}) \times (E \rightharpoonup \mathcal{N}_\omega) \times E\big)$ *is the* timed marking*,*

(iii) $R \subseteq E \times E$ *is the set of* relations *between activities*

(iv) $@ : R \to \mathcal{P}_{ne}(\{\overset{k}{\rightarrow}\bullet, \rightarrow\!\!\!\!\otimes, \bullet\!\!\overset{d}{\rightarrow}, \bullet\!\!\rightarrow\!\!\!\!\times, \rightarrow\!\!\!\!+, \rightarrow\!\!\!\!\!\!\%\})$ *is the* relation type assignment *and* $k \in \mathcal{N}$ *is the* delay *and* $d \in \mathcal{N}_\omega$ *is the* deadline*.*

(v) L *is the set of action labels,*

(vi) $l : E \to L$ *is the labelling function between activities and labels.*

Time is represented as discrete ticks, that is natural numbers. Infinity (ω) is used to represent responses with infinite deadlines, i.e. a response that must eventually happen but not within a given time, these are visually shown as response arrows ($\bullet\!\!\rightarrow$) without a deadline (d).

The timed marking $M = (Ex, Re, In)$ describes the state of the DCR graph process by assigning execution times, deadlines and inclusion status to each activity in the following way. If an activity e has been executed at least once then $Ex(e) = k$, where $k \in \mathcal{N}$ is the number of time steps since the last execution of e. If e has not been executed then $Ex(e)$ is undefined. If $Re(e) = d$ then we say that e is pending with deadline $d \in \mathcal{N}_\omega$, which means that it must be executed within d time-steps or stay forever excluded. The deadline ω represents "eventually", which corresponds to the semantics of untimed DCR graphs. If $e \in In$ we say that it is included and otherwise it is excluded.

Constraining and effect relations A relation $r = (e, e') \in R$ from activity e to e' is a r is a *constraining* relation if $\{\overset{k}{\rightarrow}\bullet, \rightarrow\!\!\!\!\otimes\} \in @r$. If $\{\bullet\!\!\overset{d}{\rightarrow}, \bullet\!\!\rightarrow\!\!\!\!\times, \rightarrow\!\!\!\!+, \rightarrow\!\!\!\!\!\!\%\} \in @r$ we say that r is an *effect* relation. Note that there can be multiple relations between e and e' and r can be both a constraining and an effect relation at the same time.

We write $e \overset{k}{\rightarrow}\bullet\ e'$ if $\overset{k}{\rightarrow}\bullet \in @r$ and say there is a *condition* from e' to e with delay k. The meaning is that at least k time steps must happen after the last execution of e before e' can be executed. If $k = 0$ the condition relation simply states that the activity e' must have been executed at least

once before e can be executed, which corresponds to the condition relation of untimed DCR graphs.

We write $e \overset{d}{\bullet\!\!\rightarrow} e'$ if $\overset{d}{\bullet\!\!\rightarrow} \in @r$ and say there is a response from e to e' with deadline d. The meaning is that e' must happen within d time steps after the last execution of e or before that stay excluded.

1.3.2.2 Petri nets

There are numerous variants of Petri nets with different expressive power. We use safe Petri nets with inhibitor and read arcs and a notion of both finite and infinite acceptance criteria (PNirp). We also extend the definition to cover time by using a variant of safe Timed-arc Petri nets with read arcs and a notion of both finite and infinite acceptance criteria (TAPNrp). We say that a Petri net is safe if the execution of transitions preserves the safeness of markings, also known as the property of all the net places being 1-bounded.

PNirp We define safe Petri nets with inhibitor arcs, read arcs and pending places as follows.

Definition 1.11. *A Petri net with inhibitor and read arcs and pending places (PNirp) is a tuple* $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$, *where*

(i) P *is a finite set of places,*

(ii) T *is a finite set of transitions s.t.* $P \cap T = \emptyset$,

(iii) $A = IA \sqcup OA$ *is a finite set of input and output arcs, where:*

- *(1)* $IA \subseteq P \times T$ *is a finite set of input arcs,*
- *(2)* $OA \subseteq T \times P$ *is a finite set of output arcs,*

(iv) $Inhib\colon IA \longrightarrow \{true, false\}$ *is a function defining inhibitor arcs,*

(v) $Read\colon IA \longrightarrow \{true, false\}$ *is a function defining read arcs,*

(vi) Act is a set of labels (actions),

(vii) $\lambda : T \to Act$ *is a labelling function,*

(viii) $Pe \subseteq P$ *is the set of pending places,*

and the constraint that if $Inhib((p,t))$ *then* $\neg Read((p,t))$ *and if* $Read((p,t))$ *then* $\neg Inhib((p,t)) \wedge (t,p) \notin OA$*. That is, an input arc cannot be both a read arc and an inhibitor arc. And if there is a read arc from place* p *to transition* t*, then there cannot be an output arc from transition* t *to* p*.*

Inhibitor arcs (also called negative contextual arcs) are special arcs between places and transitions specifying the constraint that the transition is only enabled if all places related to it by inhibitor arcs are empty. In general, the addition of inhibitor arcs makes the model of Petri nets Turing complete [6]. However, with the additional requirement of safeness that inhibitor arcs can be transformed using a complement place and a read arc.

Read arcs (also called test, activator or positive contextual arcs) [19] specify the constraint that a transition is only enabled if all places related to it by read arcs have a token. A key difference between having a read arc and a pair of input and output arcs between a transition and a place, is that read arcs are not consuming the token. This means that two transitions with read arcs to the same place can occur concurrently [94]. However, if two transitions are connected to the same place by a read arc and a standard input arc respectively, the two transitions will still be in conflict. With the use of timed tokens, read arcs also have the property of preserving the token age, whereas a pair of input and output arcs will reset the age.

The PNirp marking is defined as the subset of places containing a token.

Definition 1.12. ***(safe Marking).*** *Let* $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$ *be a PNirp. A safe marking* M *on* N *is a subset* $M \subseteq P$ *of places. We say there is a token* x *at a place* $p \in P$*, written* $x \in M(p)$*, if* $p \in M$*. The set of all markings over* N *is denoted by* $M(N)$*.*

TAPNrp We define safe Timed-arc Petri nets with read arcs and pending places as follows. The definition is adapted from Def.1 [72]. We restrict time to discrete natural number time steps as in timed DCR graphs and the set of timed intervals. Let $\mathsf{Int}_{\geq\delta}$ represent the set of time intervals $[\delta, \infty)$ where $\delta \in \mathcal{N}$. Let $\mathsf{Int}_{\leq\Delta}$ represent the set of time intervals $[0, \Delta]$ where $\Delta \in \mathcal{N}$. We restrict the intervals $\mathsf{Int}_{\geq\delta}$ to be used on transport arcs, which is used to represent delays and we restrict the intervals $\mathsf{Int}_{\leq\Delta}$ to be used on places, which represent deadlines.

Definition 1.13 ([72, Def. 1])**.** *A TAPNrp is a tuple*

$$N = (P, T, A, Inhib, Read, Transport, Inv^T, Inv^P, Act, \lambda, Pe),$$

where

(i) P *is a finite set of places,*

(ii) T *is a finite set of transitions s.t.* $P \cap T = \emptyset$*,*

(iii) $A = IA \sqcup OA$ *is a finite set of input and output arcs, where:*

(1) $IA \subseteq P \times T$ *is a finite set of input arcs,*

(2) $OA \subseteq T \times P$ *is a finite set of output arcs,*

(iv) $Inhib$*:* $IA \to \{true, false\}$ *define inhibitor arcs,*

(v) $Read$*:* $IA \to \{true, false\}$ *define read arcs,*

(vi) $Transport : IA \times OA \to \{true, false\}$ *where* $\forall (p,t) \in IA \wedge (t', p') \in OA$ *such that* $Transport((p,t),(t',p'))$ *we require that* $t = t'$*,*

(vii) $Inv^T : IA \rightharpoonup \mathsf{Int}_{\geq \delta}$ *where we require* $Inv^T(a) = [\delta, \infty) \implies \exists a'.Transport(a, a')$

(viii) $Inv^P : P \to \mathsf{Int}_{\leq \Delta}$

(ix) Act *is a set of labels (actions),*

(x) $\lambda : T \to Act$ *is a labelling function,*

(xi) $Pe \subseteq P$ *is the set of pending places,*

and the following constraints:

1. *if* $Inhib((p,t))$ *then*

$$\{\neg Read((p,t))\} \wedge \{\neg Transport((p,t),(t,p'))|\forall p' \in P\}$$

2. *if* $Read((p,t))$ *then*

$$\{(t,p) \notin OA\} \wedge \{\neg Inhib((p,t))\} \wedge \{\neg Transport((p,t),(t,p'))|\forall p' \in P\}$$

3. *if* $Transport((p,t),(t,p'))$ *then* $\{\neg Read((p,t))\} \wedge \{\neg Inhib((p,t))\}$*;*

Constraints (1),(2) and (3) specify that the domain of input arcs defined by $Inhib$, $Read$ and $Transport$ must be disjoint. In addition for (2) if there is a read arc from place p to transition t, then there cannot be an output arc from transition t to p ($\{(t,p) \notin OA\}$).

As we have a safe Petri net we again define the marking as a subset of places containing a token.

Definition 1.14. ***(Marking)*** *Let N be a TAPNrp. A safe marking M on N is a subset $M \subseteq P$ of places. We say there is a token x at a place $p \in P$, written $x \in M(p)$, if $p \in M$. The set of all markings over N is denoted by $M(N)$. We define the token age as a function on places $M_t : P \to \mathcal{N}$. We say a token has age n written $M_t(p) = n$ where $n \in \mathcal{N}$.*

Remark 1.3. *The transformation of extended and timed DCR graphs is guaranteed by construction to only create safe PNirp and safe TAPNrp respectively.*

We omit the execution semantics as they are introduced for the specific definition in each relevant paper from Part II.

1.4 List of Publications and code

The following Table 1.1 lists the contributions summarized in the remaining chapters of Part I.

	Year	Title	First author	Paper type	Venue	Publication status
1	2021	BERMUDA: towards maintainable traceability of events for trustworthy analysis of non-PAISs	Yes	Extended abstract	EMISA Forum	Published
2	2022	BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces	Yes	Workshop Full	RPM ICPM	Published
3	2024	Improving Simplicity by Discovering Nested Groups in Declarative Models	Yes	Conference Full	CAISE	Accepted (Pending publication)
4	2023	Transforming DCR graphs to Safe Petri nets	Yes	Conference Full	PETRINETS	Published
5	2024	Static and Dynamic Techniques for Iterative Test-Driven Modelling for DCR graphs	No	Journal	DKE	Under review
6	2024	DD-DisCoveR: Mining timed DCR graphs using the pm4py DisCoveR DCR extension	Yes	Demo Short	ICPM Demo	Draft
7	2024	Mapping Timed Declarative DCR Graph Specifications to safe Timed-arc Petri Nets	Yes	To Be Determined	To Be Determined	Draft

Table 1.1: List of publications

The code related to Sections 8.1, 9.1, 10.1 and 10.2 has been implemented as an extension to the pm4py framework [21] called pm4py-dcr, which contains the algorithms presented in the papers and code implementations for

DCR graphs and Timed-arc Petri nets along with execution semantics, import and export capabilities for both modelling formalisms. The code also contains a python implementation for the original DisCoveR algorithm. The code can be found at `https://github.com/paul-cvp/pm4py-dcr` and is structured in a similar manner to the rest of the pm4py framework. The code for Section 9.2 can be found at `https://github.com/Axel0087/BitDCRAlign`. The code for the prototype tool presented in Section 7.1 is available at `https://github.com/paul-cvp/bermuda-method`.

Summaries

Chapter 2

Trustworthy event log creation BERMUDA

2.1 BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces

Remark 2.1. *This section summarizes the work published in the proceedings of ICPM Workshops 2022. It was also presented as a full paper at the ICPM 2022 Workshop on Responsible Process Mining, where it won a best follow-up paper award in the workshop.*

The work summarized in this chapter is based on a paper that has been published as [41]: Cosma VP, Hildebrandt TT, Gyldenkærne CH, Slaats T. BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces. International Conference on Process Mining 2022 Oct 23 (pp. 127-139). Springer Nature Switzerland.

2.1.1 Motivation and Summary

This section summarizes the BERMUDA method for extracting trustworthy event logs. The method is based on two case studies analyzing the process of extracting data from information systems. An initial evaluation of the method has shown positive feedback through increased transparency, accountability and traceability in relation to the data provenance.

The increased use of predictive AI models and process mining models have created huge expectations on how data and process science can improve society. However creating actionable insights from data does not guarantee improvements if the insights are not aligned with the end users expectations. One major source of misalignment is the fact that data analysis was never a primary use of the information systems from which the data was extracted. We explicitly differentiate between this primary (storage) and secondary (analysis) use of data. A lot of work has gone into extracting event logs from information systems that addresses the technical challenges of data quality issues and event abstraction, but comparatively little has been done to ensure transparent, traceable matching of event data to business activities [87].

A process model comes from one of two sources. Either it is created manually by a person with domain knowledge and modelling experience, or it is mined from event logs using automatic process discovery algorithms. When it is manually created we are presented with an idealized view of the process, one which is closely related to how domain experts consider the process to be. Their expertise stems from past experience, education, training, or by understanding the guidelines and laws governing the domain in which they are employed. We refer to activities in the handmade model as domain activities. When process models are mined using process discovery, the data needs to be in the form of event logs. Therefore the first task for a data scientist is to extract an event log from the information systems database. This task is also a moment when data quality issues and event abstraction have to be addressed. Once a process model is mined we also have to quantify the degree of agreement between the model and event log and it is at this stage where we have strong techniques to quantify this using alignment or conformance-based measures. In an attempt to improve the event log extraction stage, we propose a method to increase collaboration between data scientists, domain experts and systems engineers to create trustworthy event logs.

The projection of data into an event log should preserve the inherent proprieties of the data source. Domain knowledge is necessary to understand if the event log will capture the behaviour of the real-world process we intend to analyze. Finally the enactment of the real-world process in the data is filtered and limited by the user interface or software solution through which the data is entered. This leads us to question just how trustworthy event logs can be. Now we step into BERMUDA: Business Event Relation Map via

Role	Has domain knowledge?	Has data access?	Has information system expertise?
Domain expert	Yes	Limited to the assigned cases/patients	Yes, as a system user
Data analyst	Limited, as necessary for achieving the task	Yes, data can be anonymized or synthesized	Limited, for data extraction
System engineer	Limited, as necessary for developing the system	Limited to anonymized or synthesized	Yes, as a system engineer

Table 2.1: BERMUDA: Role-based access control and competences

User-interface to Data for Analysis. In other words, BERMUDA, the place where information gets lost. We employ Participatory Design (Definition 2.1) theory to propose a method and prototype tool for the responsible creation of trustworthy event logs through stakeholder collaboration.

Definition 2.1. *"Participatory Design can be defined as a process of investigating, understanding, reflecting upon, establishing, developing, and supporting mutual learning between multiple participants in collective 'reflection-in-action' [114]. The participants typically undertake the two principal roles of users and designers where the designer strive to learn the realities of the users situation while the users strive to articulate their desired aims and learn appropriate technological means to obtain them." [117]*

We define three roles within BERMUDA: the data scientist, domain expert and system engineer. Their key competences and data access requirements are highlighted in Table 2.1. The method takes advantage of each roles key competences while taking into account citizens data privacy concerns and the software vendors protection of intellectual property rights. BERMUDA is applied in three steps. First, a domain expert maps domain activities to user interface areas of the information system. Then system engineers link the user interface to the data source and provide the relevant data extraction scripts. Finally the data scientist can associate the domain activities with the extraction scripts to generate the event log.

The need for a method to facilitate knowledge sharing in the event log extraction stage arose from two concrete cases, one related to a municipal job center and another to a hospital. In the job center case we intended to automate the internal audit schema the job center used for checking their compliance with the law. In the hospital case we intended to build an AI model to predict future no-shows for patient appointments. Both cases intended to use their data for AI-based decision support and both studies revealed a discrepancy between the intended system use and its actual use. When

adding information to a case, users preferred selecting the "other" category from a drop-down menu, which severely limited the secondary usefulness of the data. Writing a textual description was a favored workaround instead of scrolling through the proposed categories. Knowledge of both the system and the domain was important for handling the missing data.

Based on the two cases a common pattern emerged. Data scientists elicited knowledge in iterations from domain experts and system engineers. They kept track of how domain activities were recorded and how data was extracted. After an initial evaluation with a domain expert and a data scientist we found that extracting data for secondary use through the BERMUDA method exhibited several positive properties. Both interviewees saw an advantage in having a link between domain activities and database events as a way remove ambiguities and improve transparency, accountability and traceability. Additionally they saw it beneficial to store the triple (domain activities, user interface areas and database extraction scripts) to prevent the event log from being out of date when any of the elements changed. Finally they suggested that the triple can be used when on-boarding new team members.

2.1.2 Discussion

Fitting within process mining life cycle models If we look at the L* lifecycle model [129] our method fits within stage 0 "Plan and justify" and stage 1 "Extract". If we look at the PM^2 Process Mining Methodology [130] our method fits within the task "Transferring process knowledge" in Stage 2 "Extraction".

The triples created in the method belong to stages 0 and 1 in the L* process mining life cycle model as depicted in Fig. 2.1, where Data and Business understanding are the main artifacts used in the extraction stage. Applying BERMUDA within the life cycle model allows for a more agile and iterative extraction step. After the initial extraction it is necessary to reiterate stages 0 and 1 to increase the quality of the output of stage 1, w.r.t. traceability, transparency and accuracy. It also improves maintainability when some of the resulting artifacts change as a result of refinements within the L* life cycle.

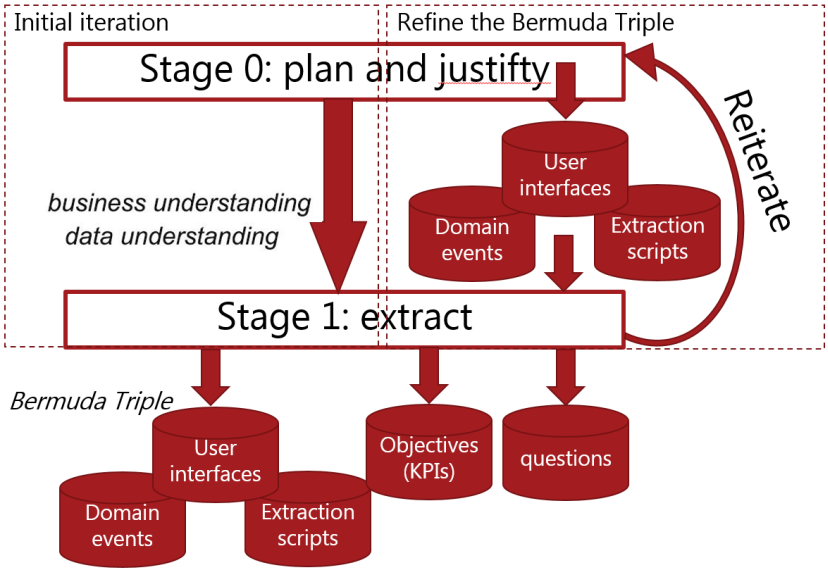


Figure 2.1: Fitting BERMUDA within the L* life cycle model

Chapter 3

Understandable declarative process models

3.1 Improving Simplicity by Discovering Nested Groups in Declarative Models

Remark 3.1. *This section summarizes an accepted conference paper for the International Conference on Advanced Information Systems Engineering (CAISE) 2024.*

The work summarized in this chapter is based on a preprint paper that will be published as: Improving Simplicity by Discovering Nested Groups in Declarative Models Cosma VP, Christfort AKF, Hildebrandt TT, Lu X, Reijers HA, Slaats T. International Conference on Advanced Information Systems Engineering 2024.

3.1.1 Motivation and Summary

This section summarizes the development and evaluation of 3 novel process discovery algorithms for mining declarative process models with activity groups. The contribution is also the first to evaluate 4 new empirically validated complexity metrics on an exhaustive set of 21 publicly available event logs. The novel algorithms discover more understandable process models. We use the complexity metrics as evidence for the increased understandability.

Process discovery allows us to automatically reconstruct the real-world process from stored event data. Discovered models come in two flavours

depending on the choice of miners (discovery algorithms): imperative or declarative. Imperative models are more akin to a factory or warehouse setting where the process is structured. In this setting imperative miners find structured and simple processes. Imperative mined models become quickly overly complicated as the process allows for more flexibility, in this case imperative miners discover spaghetti models [129]. As we look at more flexible processes, such as case work or healthcare processes, declarative models tend to capture flexibility more succinctly.

DCR graphs are an example of a declarative process model and DisCoveR is an award winning process miner that has consistently been shown to produce highly accurate models from event logs. DisCoveR mines models according to Definition 1.2 of *core* DCR graphs. However many of the discovered relations apply to groups of activities and specific behavioural patterns, such as mutual exclusive choices. This gives us models that have multiple relations between the same subsets of activities. One solution to reducing the visual number of relations while still maintaining the perfect fitness of the discovered models is to use the *group* Definition 1.9 of DCR graphs. Groups, also known as nestings in previous papers [66] are syntactic short-hand signifying that a relation to/from a group applies to the entire set of activities under that group. This definition also applies recursively, allowing us to have groups within groups.

We have developed three algorithms that map from the definition of core DCR graphs to the definition of group DCR graphs. The first algorithm, called *Choice*, finds mutual exclusion groups, that is groups of self-excluding activities with a pair of exclusion relations $\rightarrow\!\!\!\!\!\!\%$ between each activity in the group. Each group of activities is then mapped under an activity group with a self exclusion relation. The second algorithm, called *Group*, is an efficient greedy algorithm which finds all groups that reduce the number of relations w.r.t the graph size. The third algorithm, called *Choice+Group*, first applies the *Choice* pattern before greedily reducing the graph size using the *Group* algorithm.

Motivated by the results of Andaloussi et. al. [5] which found a correlation between a set of complexity metrics (size, density, separability and constraint variability) and users cognitive load, we evaluated our three algorithms against a set of 16 real life event logs and 5 synthetic ones. We benchmarked our algorithms against DisCoveR which produces core DCR

graphs[1] and found an improvement of 42% in size, 65% in density, 5% in separability and a deterioration of 22% in constraint variability for the greedy *Group* algorithm. *Choice+Group* came in a close second with *Choice* coming third.

On the evaluated event logs we saw that an improvement on size, density and separability implies a worsening on constraint variability. A possible explanation is that DisCoveR tends to find a high number of exclusion and condition relations. As we reduce these disproportionately preferred relations the overall ratio of each relation type tends to even out, therefore leading to a higher overall constraint variability.

As the complexity metrics are a proxy for the users understandability of declarative process models [5] we have therefore shown how to discover more understandable DCR graphs by reducing the number of relations while still maintaining perfect fitness w.r.t. the event log.

[1] Our algorithms also work on the extended definition of DCR graphs.

Chapter 4

Verifiable declarative process models

4.1 Transforming Dynamic Condition Response Graphs to Safe Petri Nets

Remark 4.1. *This section summarizes the conference paper published in the proceedings of Application and Theory of Petri Nets and Concurrency. Part of the book series: Lecture Notes in Computer Science (LNCS, volume 13929). It was presented at the PETRI NETS 2023 conference main track.*

The work summarized in this section is published as [44]: Cosma VP, Hildebrandt TT, Slaats T. Transforming Dynamic Condition Response Graphs to Safe Petri Nets. International Conference on Applications and Theory of Petri Nets and Concurrency 2023 May 28 (pp. 417-439). Cham: Springer Nature Switzerland.

4.1.1 Motivation and Summary

This section summarizes the transformation of the DCR graphs constraint-based process specification language to safe Petri nets with inhibitor arcs, read arcs and pending places (PNirp). We proved that the resulting PNirp is bisimalar with the DCR graph and that the bisimulation respects the acceptance criteria. We showed how to construct the PNirp from any DCR graph and that the construction follows closely the bisimilarity proof. Finally we

showed how pre and post optimization techniques allow us to create PNirps that are feasible for further analysis either through visual inspection, token replay, or through verification via Petri net model checkers.

"Declarative knowledge can often be compiled into more efficient procedural code" [111] is a quote from the chapter on Logical Agents from the text book "Artificial Intelligence, A modern approach" by Russell and Norvig. This statement applies to software engineering too, where requirements specifications are typically translated to imperative code when the system is implemented. Petri nets are a well established process notation when it comes to modelling system behaviour. Widely used both in the model checking community as well as in the process mining one, Petri nets benefit from a wide range of tool support and enjoy a central place when synthesizing or mapping to and from other process notations.

DCR graphs is a relatively new process notation that does not yet have a powerful model checker. DisCoveR [17], a DCR graphs miner, has already demonstrated with an early unsound translation to Petri nets to capture process behaviour more accurately than imperative miners using block-structured approaches. Motivated by the dual benefits of model checking DCR graphs and providing a declarative Petri net miner we defined a bisimilarity preserving mapping from core DCR graphs to PNirp. The mapping produces safe Petri nets by construction that are fully compatible with the TAPAAL model checker [28]. We presented a theorem stating the bisimulation relation between any DCR graph and its mapped PNirp which we proved by induction in the number of elements of the DCR graph. The bisimulation proof closely follows the construction stages of the PNirp. At each stage of the construction/proof we take a different element of the DCR graph and create its equivalent representation in the PNirp and at the end of each stage the two models are bisimilar up to the number of DCR graph elements already taken. We now give the acceptance criteria definitions and a short remark on arc pattern tables before briefly sketching the construction of the PNirp and its proof of bisimilarity.

Definition 4.1. *(Definition 5 in [44]) Let* $G = (E, M_0, R, @, L, l)$ *be a DCR graph. A finite or infinite sequence of transitions* $M_0 \xrightarrow{e_0}_G M_1 \xrightarrow{e_1}_G \ldots$ *in* $[[G]]$ *with* $M_i = (Ex_i, Re_i, In_i)$*, is* accepting *if* $e \in Re_i \cap In_i$ *implies* $\exists j \geq i.(e_j = e \vee e \notin Re_j \cap In_j)$.

Definition 4.2. *(Definition 11 in [44]) Let* $N = (P, M, T, A, Inhib, Read, Act, \lambda, Pe)$ *be a PNirp with safe marking* M*. A finite or infinite sequence of transi-*

tions $M_0 \xrightarrow{t_0}_N M_1 \xrightarrow{t_1}_N \ldots$ *in* $[[N]]$ *is* accepting *if* $p \in M_i \cap Pe$ *implies* $\exists j \geq i.(p, t_j) \in IA$.

Remark 4.2. *We briefly explain what an arc pattern table is for the base case on a single event e (induction step 0), and when mapping a relation (induction step > 0) between two events e and e′.* ***Base case (e):*** *Each row in the table represents one transition labelled by* $l(e)$*. Each column represents a place related to e. Each item in the table represents an arc type between the transition and the place.* ***Relations (eRe′):*** *Each row represents a copy of the entire set of transitions labelled by* $l(e)$ *(one or more). Each column represents a place related to* e'*. Each item is an arc type between each transition in the set of transitions labelled by* $l(e')$ *and the single place of* e'*.*

Events - induction step 0 Given a DCR event e we created four places, each place representing the four key states a DCR event can be in: an included place, an executed place, a pending included place and a pending excluded place. We made a distinction between the pending state of a DCR event to respect the acceptance criteria. We added a token to the included place if e was included in the DCR Marking. We added a token in the executed place if e was executed. We either added a token in the pending included or pending excluded place according to both the pending and included marking of e in the DCR Marking. Next we created four transitions with the same label as the DCR event label. We mapped the internal behaviour of the DCR event using arcs between the already created places and transitions according to the arc pattern table for the base case.

This single event PNirp now captured the change an event has on its own marking, and at the same time, by construction, at most one transition labelled by the event can be enabled at any given instance. Also by construction, at most one token can exist in any place and this defines the PNirp initial marking. We repeated this step for each event in the DCR graph. This consists of the *base case* in our bisimulation proof.

Relations - induction step k For each relation between two DCR events e and e' we first created n copies of all existing transitions at stage $k-1$ labelled by $l(e)$ (where n is the number of possible behaviours induced by e to the marking of e' according to the specific relation type). For each of the n transition copies labelled by $l(e)$ we mapped all the existing arcs to the same

places from stage $k-1$ and we added new arcs to the places of e' according to the arc pattern table for the specific relation type. Finally we removed all existing transitions at stage $k-1$ labelled by $l(e)$ and their associated arcs.

If stage $k-1=0$ (i.e. the base case) we would originally have 4 transitions labelled by $l(e)$ at stage 0. We created $n \times 4$ new transitions, each of the n copies having the same arcs w.r.t. the places of e. For each n, each of the 4 transitions would get the same new arc mapped to the places of e' according to the arc pattern table of that relation type. Finally we remove the original 4 transitions labelled by $l(e)$ at stage 0. In the end we are left with $n \times 4$ transitions labelled by $l(e)$ out of which at most one can be enabled at any given instance. This is because out of the 4 original transition at most one can be enabled (as it was shown for the *base case*) and for each of the n groups of transitions at most one group can be enabled due to the construction defined by the arc pattern table for the specific relation type. In other words when we do the Kronecker product of the arc pattern table at stage $k-1$ with the arc pattern table at stage k. By construction, at most one transition labelled by $l(e)$ is enabled at any given instance in the PNirp execution.

If stage $k-1 \neq 0$ we repeat the same procedure only instead of the 4 transitions labelled by $l(e)$ we would have potentially more if there have already been other relations mapped from e to any other e'. This consists of the *induction step* in our bisimulation proof.

In the paper we show the mapping in more details, specifically we differentiate based on the following:

1. When mapping effect relations (responses, inclusions and exclusions) the event creating the effect has its transitions copied;

2. When mapping constraining relations (conditions) the event being constrained has its transitions copied[1];

3. When mapping a self-relation (the case when $e = e'$) instead of creating copies we change the arc pattern table from the base case;

4. When mapping relations with more than one relation type between e and e' we either define an arc pattern table or the mapping is equivalent to an arc pattern table of a subset of relations types.

[1] in the full paper the condition relation $\rightarrow\bullet$ is intentionally inverted $\bullet\leftarrow$ to maintain consistency in the proof and arc pattern table, that event e is always the one for which we create transition copies

To produce usable models we provide two solutions to limit the exponential size blow up of the resulting PNirp. One approach was by statically examining the DCR graph structure to know which event markings are relevant. This allowed us to only create subsets of places and transitions at the events mapping stage. As the PNirp is safe, we also did a reachability analysis on the transition system of the mapped PNirp model and removed any dead regions.

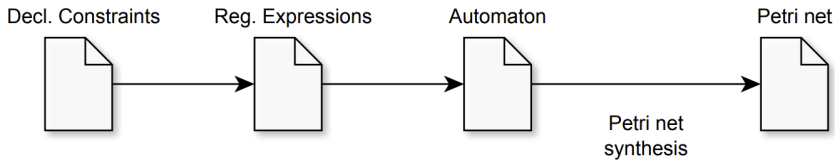
4.1.2 Discussion

Process discovery In an initial evaluation of the mapping we looked at how the DECLARE notation was mapped to a Petri net model. The authors in [107] used MINERful [36] on the BPIC13 [53] event log and used a synthesis approach via Finite State Automata to create the final safe Petri net. We give the figures shown in the original paper [107] (Figure 4.1) to compare with our results using DisCoveR (Figure 4.2). Both conversions produce safe Petri nets. The conversion from MINERful has 14 places and 31 transitions. The conversion from DisCoveR has 6 places and 8 transitions. The only caveat is that we use inhibitor arcs, but we replace read arcs with pairs of input and output arcs as a way to keep our PNirps compatible with TAPAAL [28].

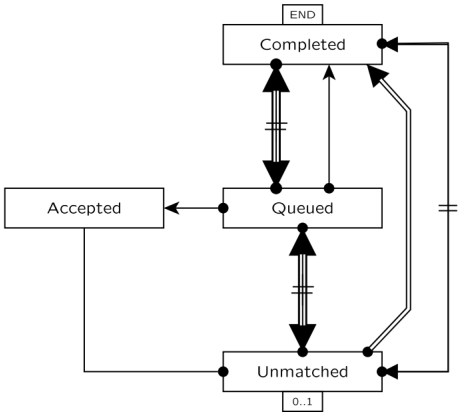
We investigated how the mapping to Petri nets compares with native Petri net miners (the Inductive [77] and ILP [51] miners), by first using DisCoveR on the event logs. We report our results in Table 4.1. Note that our runtime takes into account both the process discovery step and the transformation therefore it will always be slower and we also report from the total number of arcs (#Arcs) how many out of those are inhibitor arcs (#Inhibitor). We configured all miners to produce perfectly fitting models and we used the Petri net token-replay conformance checker when computing the precision and generalization metrics (we replaced the standard execution semantics with the inhibitor net semantics).

The most striking result in favor of our approach is on the Road Traffic Fine Management Process (RTFMP) event log. We show the resulting models from this log in Figure 4.3.

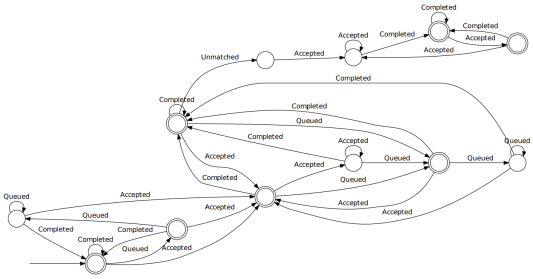
Model checking finite runs. In our transformed PNirp we defined the acceptance criteria for infinite runs in order to match with the DCR graphs acceptance criteria. However in order to allow for LTL model checking in



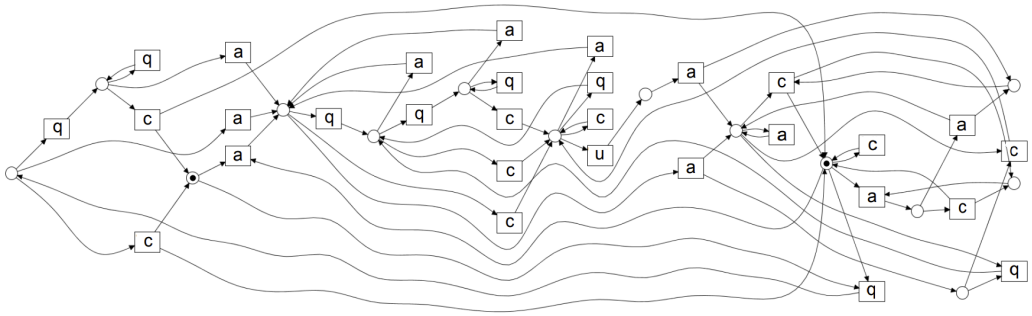
(a) Pipeline for DECLARE



(b) DECLARE model

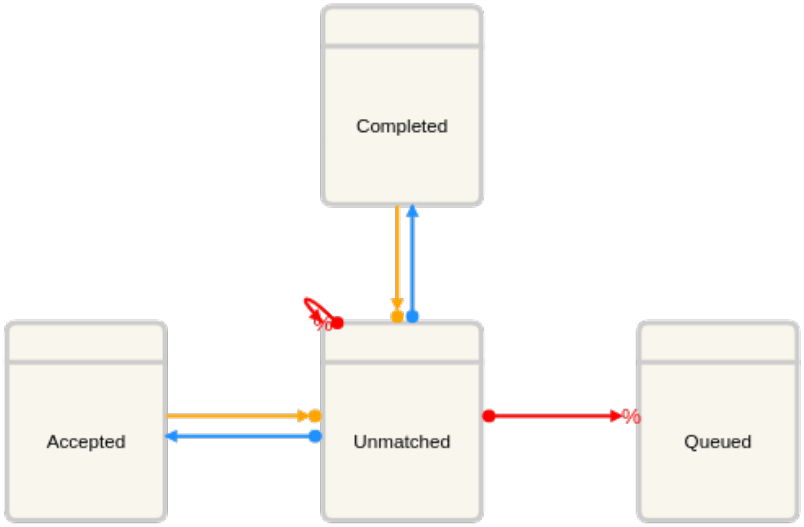


(c) FSA

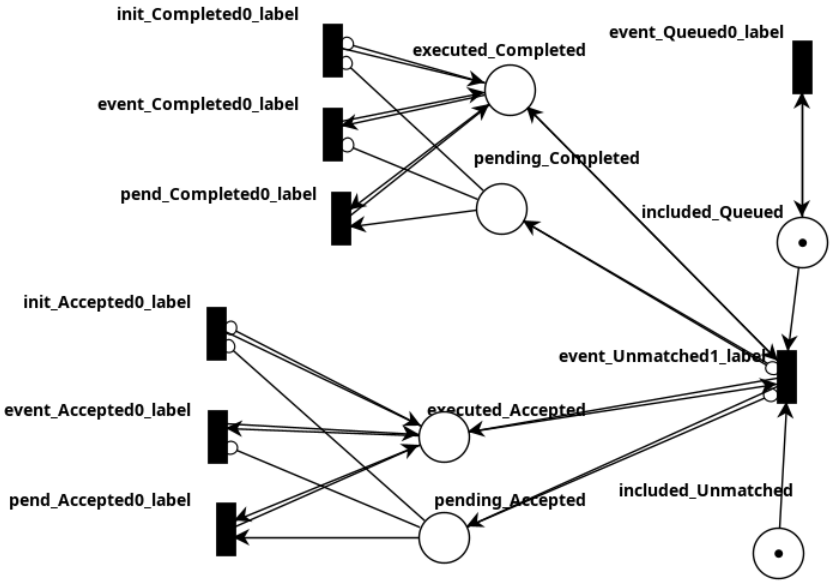


(d) Petri net

Figure 4.1: MINERful to DECLARE to Regex to FSA to Petri net for BPIC13 [107]

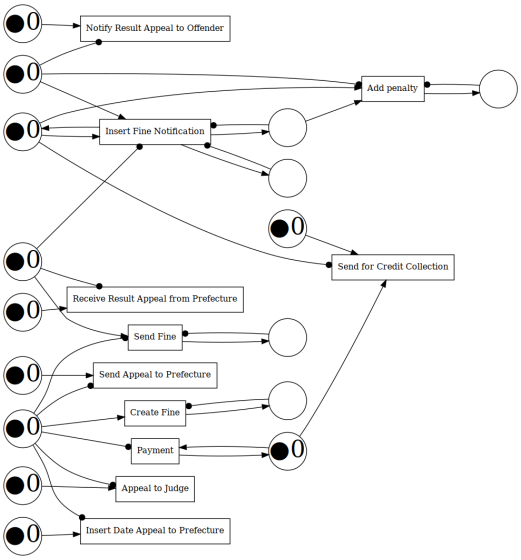


(a) DCR graph

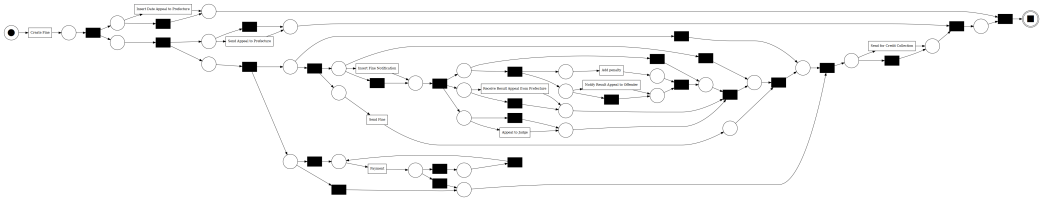


(b) Petri net

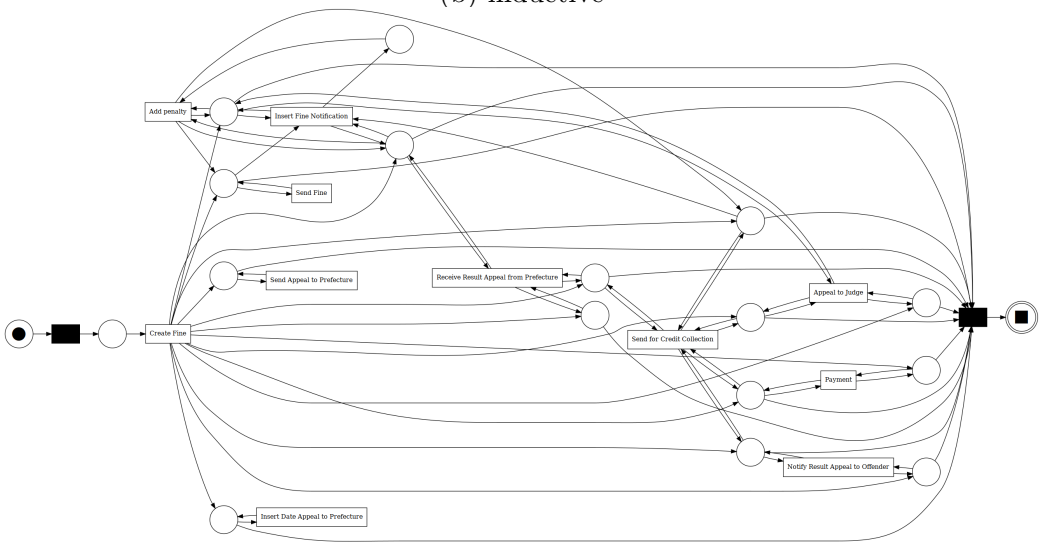
Figure 4.2: DCR to PNirp for BPIC13



(a) dcrtotapn



(b) inductive



(c) ilp

Figure 4.3: dcrtopn vs inductive vs ilp on RTFMP

Log name	Algorithm	Precision	Generalization	Size	#Transitions	#Places	#Arcs	#Inhibitor	Runtime(s)
BPIC12	dcrtopn	**0.321**	0.291	255	211	44	2988	791	493.623
	inductive	0.220	0.949	156	90	66	**198**	0	**12.577**
	ilp	0.132	**0.975**	**57**	**26**	**31**	242	0	37.978
BPIC13_i	dcrtopn	**0.888**	0.826	14	8	**6**	30	8	1.361
	inductive	0.684	0.892	50	28	22	60	0	**0.110**
	ilp	0.684	**0.918**	**13**	**6**	7	**26**	0	1.692
BPIC13_cp	dcrtopn	**0.884**	0.726	**13**	10	**3**	**24**	7	0.125
	inductive	**0.884**	0.904	53	28	25	62	0	**0.018**
	ilp	**0.883**	**0.926**	14	**6**	8	30	0	0.164
BPIC14_f	dcrtopn	**0.613**	0.981	33	17	**16**	109	43	6.325
	inductive	**0.613**	0.984	99	55	44	120	0	**1.094**
	ilp	0.523	**0.993**	**30**	**11**	19	**84**	0	17.248
SEPSIS	dcrtopn	**0.542**	0.806	**55**	28	**27**	268	51	17.814
	inductive	0.498	0.907	125	68	57	**152**	0	**0.352**
	ilp	0.386	**0.919**	65	**18**	47	238	0	2.390
RTFMP	dcrtopn	**0.849**	**0.986**	**27**	**11**	**16**	**36**	15	12.745
	inductive	0.624	0.978	73	39	34	92	0	**0.596**
	ilp	0.631	**0.987**	31	13	18	82	0	11.536
BPIC17_f	dcrtopn	0.622	0.634	92	50	42	434	123	23.156
	inductive	**0.730**	0.972	91	51	40	**104**	0	**1.263**
	ilp	0.201	**0.993**	**40**	**20**	**20**	134	0	37.345
BPIC17	dcrtopn	**0.274**	0.448	184	124	**60**	1414	484	2209.933
	inductive	0.210	0.937	178	105	73	**220**	0	**64.776**
	ilp	0.172	**0.983**	**92**	**28**	64	544	0	416.869
BPIC17-Offer	dcrtopn	**1.000**	**0.991**	22	**8**	14	39	14	5.591
	inductive	0.877	0.983	**17**	11	**6**	**22**	0	**0.224**
	ilp	0.683	**0.992**	29	10	19	84	0	4.708

Table 4.1: Petri net conversion and imperative miners

the TAPAAL tool we must limit our acceptance criteria only for finite traces both in the DCR graph Definition 4.3 and in the PNirp Definition 4.4.

Definition 4.3. *(Adapted from definition 5 of [44]) Let* $G = (E, M_0, R, @, L, l)$ *be a DCR graph. A finite or sequence of transitions* $M_0 \xrightarrow{e_0}_G M_1 \xrightarrow{e_1}_G \ldots \xrightarrow{e_n}_G M_n$ *in* $[[G]]$ *with* $M_n = (Ex_n, Re_n, In_n)$*, is* accepting *if* $Re_n \cap In_n = \emptyset$*.*

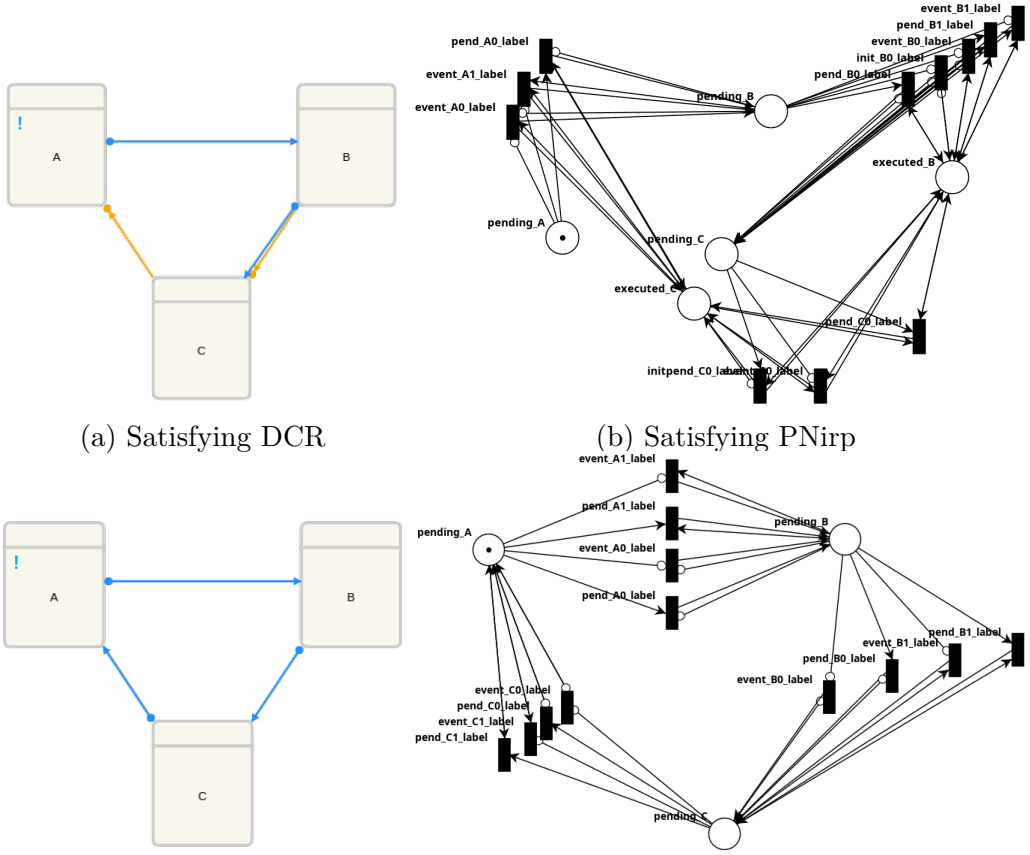
Definition 4.4. *(Adapted from definition 11 of [44])*

Let $N = (P, M, T, A, Inhib, Read, Act, \lambda, Pe)$ *be a PNirp with safe marking* M*. A finite sequence of transitions* $M_0 \xrightarrow{t_0}_N M_1 \xrightarrow{t_1}_N \ldots \xrightarrow{t_n}_N M_f$ *in* $[[N]]$ *is* accepting *if* $M_f \cap Pe = \emptyset$*.*

To verify that there exists an accepting run in the DCR graph is equivalent to checking that there exists an accepting path in the PNirp. In LTL we express this as eventually all pending included places have no tokens:

$$\Diamond \bigwedge_{e \in E} \text{pending\_}e = 0$$

Figure 4.4 shows two examples of DCR graph, PNirp pairs satisfying (a),(b) and failing (c),(d) the LTL query 4.1.



(a) Satisfying DCR (b) Satisfying PNirp

(c) Failing DCR (d) Failing PNirp

Figure 4.4: Examples of Passing (a,b) and Failing (c,d) models

$$\Diamond(\text{pending\_A} = 0 \wedge \text{pending\_B} = 0 \wedge \text{pending\_C} = 0) \tag{4.1}$$

4.2 Static and Dynamic Techniques for Iterative Test-Driven Modelling of Dynamic Condition Response Graphs

Remark 4.3. *This section is a summarises an invited journal paper, currently under review, for Data and Knowledge Engineering journal special issue on Augmented Business Process Management.*

4.2.1 Motivation and Summary

Declarative process notations map behaviour explicitly as a set of rules whereas imperative ones implicitly capture the behaviour through possible execution sequences. Imperative models tend to become spaghetti models when capturing flexible behaviour because every possible execution sequence needs to be modelled. Declarative models capture flexible behaviour more concisely, but lack the clearer more intuitive sequential view of behaviour. One approach to capture the benefits of both is to use the theory of test-driven modelling (TDM) and open tests.

In test-driven modelling one can express behaviour as sequences of activities. These sequences can represent desired behaviour as positive open tests and undesired behaviour as negative open tests. Finally the sequences must be executed under a context, i.e. a set of activities under which the sequence is tested. For positive open tests we require that there exists a model trace which projected onto the context is exactly the positive open test sequence. For negative open tests we require that for all model traces projected onto the context there is no trace that exactly matches the negative open test sequence.

Through iterative test-driven modelling one can define positive and negative open tests for the initial version of a process model and rerun the tests every time a new version of the model is created, similar to how regression testing in software engineering works. Running open tests requires us to explore the model state space and this can be time consuming (especially for negative open tests) if we want timely feedback in our modelling environment or we have large models and many tests. To improve the run time requirements of running open tests on DCR graphs we provided a set of static techniques which under certain types of model extensions are guaranteed to

preserve open tests and a translation of open tests to an alignment problem when static techniques are not satisfied.

For static checks we defined when a DCR graph extension preserves positive tests as a transparent process extension. We defined when a DCR graph extension preserves negative tests as an exclusion-safe extension. For the alignment problem on test cases we defined a specific cost function which assigns cost 0 when we do a synchronous move within the context or when we do a model move outside the context, and cost infinity otherwise. Under this cost function a DCR graph extension passes a positive open test if and only if any optimal alignment has cost 0, and passes a negative open test if and only if any optimal alignment has cost infinity.

When running alignments for open tests we noticed that we can further define a static check of non-reachability for when an activity is inaccessible under a context. This extra static check allows us to prune the model search-space during alignments by assigning cost infinity to any partial alignment for which the next event is inaccessible under the context. We have shown that this holds over any model-step outside the context. Finally we benchmarked our alignment technique for open tests with and without the static check pruning. We noticed that the static check of non-reachability slightly increased the runtime of positive open tests and significantly reduced the runtime of negative open tests. As expected the pruning is most effective when the entire search space must be exhausted, as it is for negative open tests. Overall on a test suite of 7 positive and 11 negative open tests we achieved a significant speedup.

Chapter 5

Handling time

5.1 DD-DisCoveR: Mining timed DCR graphs using the pm4py DisCoveR DCR extension

Remark 5.1. *This section sumarizes a preprint paper for the International Conference on Process Mining 2024 demo track.*

5.1.1 Motivation and Summary

This section summarizes the work done on the Delay Deadline DisCoveR (DD-DisCoveR) algorithm and is presented as an independent extension of the DisCoveR miner. The algorithm takes as input an event log with an associated Condition Response Graph and returns the minimum delay for conditions and maximum deadline for responses formatted in the ISO8601 date time standard. Condition Response Graphs are core DCR graphs without the dynamic inclusion and exclusion relations.

Timing constraints provide us with more information about the underlying real-world process and leads to more expressive process models.

We mined timing differences between activity pairs $A_1 \rightarrow\!\!\bullet\ A_2$ connected by a condition relation by finding the minimum time difference between A_1 and A_2 with no occurrence of either A_1 or A_2 in between. The mined minimum delay k therefore respects the execution semantics of a timed condition relation, namely that after an execution of A_1 an execution of A_2 may be

observed after at least k time steps. Mining timing information according to the condition execution semantics allows us to safely ignore consecutive instances of A_2 or instances of A_1 without a subsequent A_2 in the same trace.

We mined timing differences between activity pairs $A_1 \bullet\!\!\rightarrow A_2$ connected by a response relation by finding the maximum deadline d from both every consecutive pair of A_1 and A_2 (similar to the condition), and also from every consecutive pair of A_1 and A_1. The explanation for considering the additional pair (A_1, A_1) comes from the timed response execution semantics. The semantics state that after we have observed A_1 in our trace we must eventually observe A_2, after at most d time steps. Otherwise, if we have observed another A_1 after less than d time steps, then we have postponed the execution of A_2 by the exact time difference between the two consecutive repetitions of A_1.

With timing information we can extend any DCR graph definition to a timed DCR graph. We show how we applied this on the graph mined from the RTFMP event log in Figure 5.1. Note that for self-excluding activities with a delay condition we are required to wrap our activity into a subprocess to enforce our delay after the activity has been self-excluded.

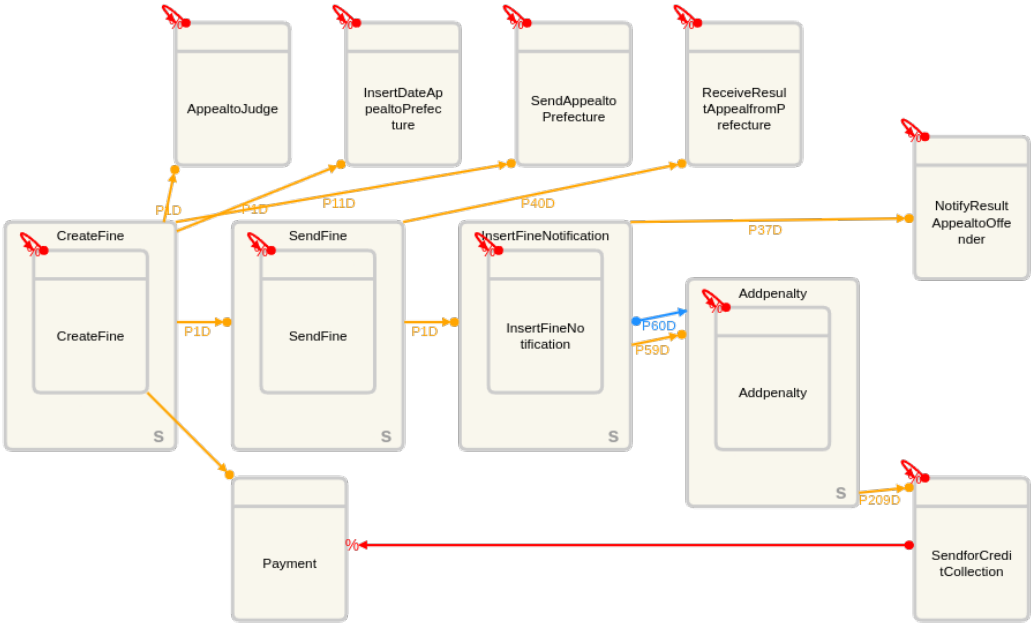


Figure 5.1: Timed DCR graph for the Road Traffic Fine dataset

5.2 Transforming Timed Dynamic Condition Response Graphs to safe Timed-arc Petri Nets

Remark 5.2. *This section summarizes a draft paper. It is a extends and contains material from the previously accepted PETRI NETS 2023 conference paper [44]: Cosma VP, Hildebrandt TT, Slaats T. Transforming Dynamic Condition Response Graphs to Safe Petri Nets. International Conference on Applications and Theory of Petri Nets and Concurrency 2023 May 28 (pp. 417-439). Cham: Springer Nature Switzerland.*

5.2.1 Motivation and Summary

This section summarizes the transformation of Timed DCR graphs to Timed Arc Petri Nets with read arcs and pending places (TAPNrp). It is an extension with time for the work summarized in Section 4.1. We therefore followed the same construction and the same proof structure as for the untimed version. We still had the base case stage where we mapped all the DCR activities and the relation stage where we mapped the DCR relations by creating copies of existing transitions and new arcs for each subset. What we highlighted were the changes induced by the addition of time. We also remarked that we now have a link from event logs via process discovery to timed automata, through previous work on mapping Timed-arc Petri Nets to Networks of Timed Automata [121].

Time in DCR graphs is represented by delays for the constraining condition relations and deadlines for the response effect relations. Between two activities $e \xrightarrow{k}\!\!\bullet\ e'$, a delay k is represented in the TAPNrp as the semi-closed interval $[k, \infty)$ and is used as a guard for all the transport arcs mapped between the executed place of e and the set of transitions labelled by $l(e')$. Between two activities $e \bullet\!\!\xrightarrow{d} e'$, a deadline d is represented in the TAPNrp as the closed interval $[0, d]$ and is used as an invariant for the pending included place of e' when it is made pending by e.

Each timed response to e' can have a different deadline as well as possibly an initially pending deadline from the initial DCR marking. To keep track of the source of the deadline in our TAPNrp we were required to have a pending included and pending excluded pair of places for each individual

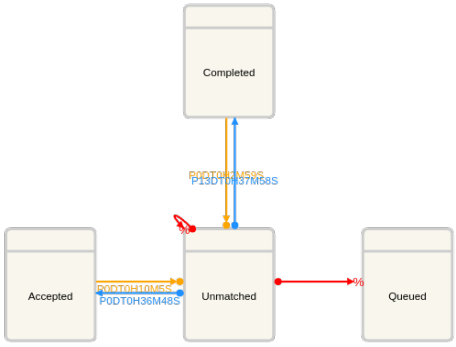
deadline. We also had to create extra transitions for the pending included places. Therefore the number of transitions required for each activity in the base case is now $2+2\times n$ where n is the number of individual deadlines. The extra requirement to maintain a bisimilar mapping is that at most one of the set of all pending places (included and excluded) can hold a token at any instance in the TAPNrp execution. We handle and prove this requirement holds by construction.

Any activity e that makes e' pending through a response ($e \overset{d}{\bullet\!\!\rightarrow} e'$) now also needs to keep track of any other activities making the same e' pending. We did this by creating extra transitions labelled by $l(e)$ and arcs for each of the pending places of e' in the relation stage of the mapping. Any activity e'' that either includes or excludes e' ($e'' \rightarrow\!\!+\ e'$ or $e'' \rightarrow\!\!\!\!\%\ e'$) also needs extra transitions labelled by $l(e'')$ and pairs of transport arcs to move tokens between pairs of pending included and pending excluded places of e'. We still use arc pattern tables to add arcs between copied subsets of transitions and places, but with an extra detail. The pairwise tracking of pending place means that we need to distinguish between adding the same arcs to all copies, and adding pairwise the same arcs. By construction the mapping is safe.

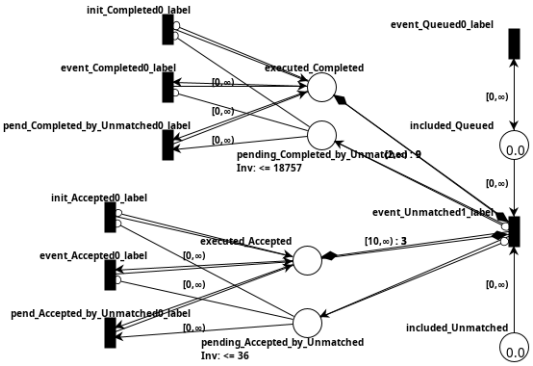
Finally we remark that pre optimizing the mapping through static analysis of the timed DCR graph is the same as for the untimed mapping and that any TAPNrp has at most the same reachability graph as the untimed PNirp (if we account for the expansion in pending places). Therefore we can apply the same pre and post optimization techniques as for the untimed case.

5.2.2 Discussion

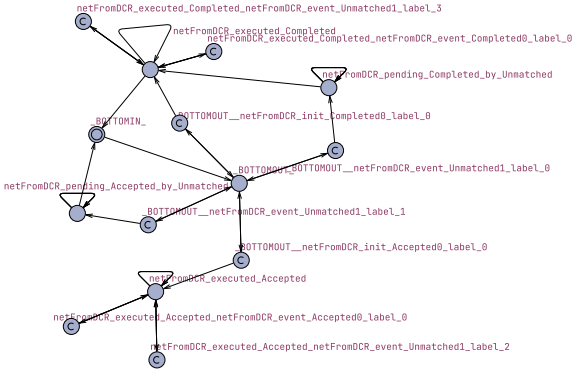
To highlight that we can now get from an event log to a network of timed automata we have run the process discovery and mapping pipeline and reported the resulting models for both BPIC13 (Fig. 5.2b) and RTFMP (Fig. 5.3b) event logs.



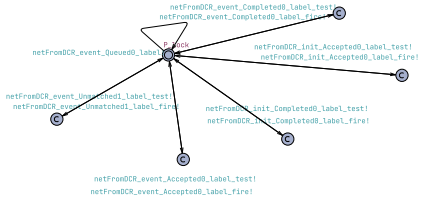
(a) Timed DCR model



(b) Timed arc Petri net

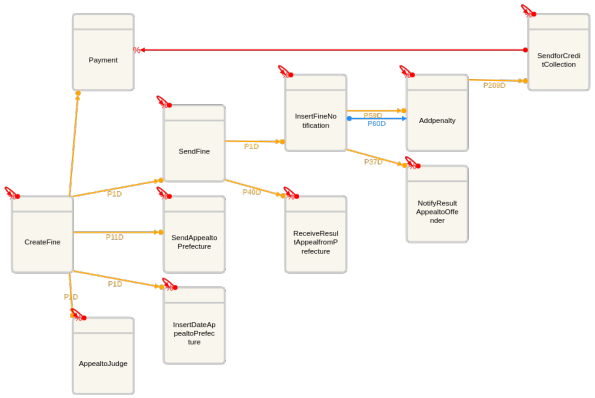


(c) Timed automata - Token

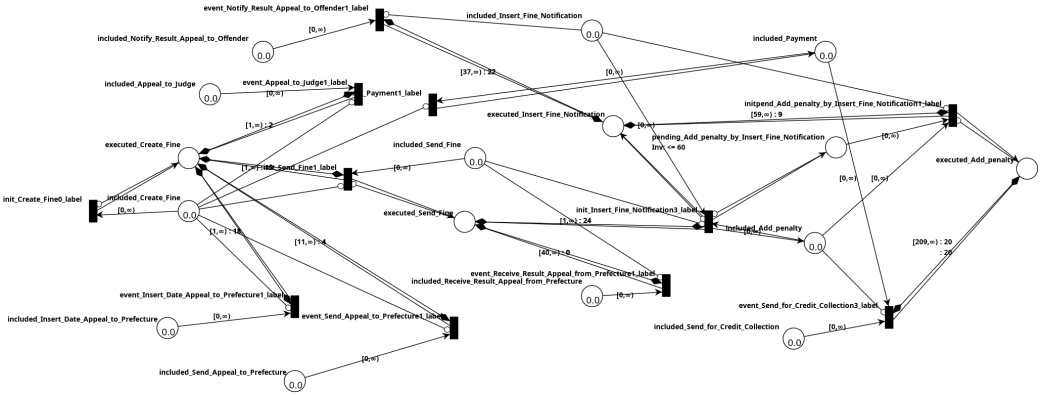


(d) Timed automata - Control

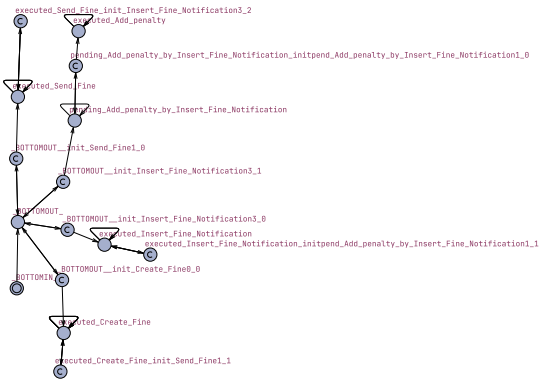
Figure 5.2: BPIC13 DCR to TAPN to NTA



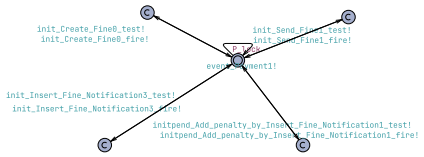
(a) Timed DCR model



(b) Timed arc Petri net



(c) Timed automata - Token



(d) Timed automata - Control

Figure 5.3: RTFMP DCR to TAPN to NTA

Chapter 6

Conclusion

We showed how explainability can be improved throughout the process mining stages by increasing the trustworthiness level of event logs and by increasing the understandability and verifiability of our declarative process models. Finally we showed how time enhances the expresiveness of DCR graphs while still maintaining the same level of verifiability.

First we showed the BERMUDA method extracts trustworthy event logs through increased transparency, accountability and traceability in the event log creation stages. It maintains a high accuracy to the real-world process by validating the control-flow with help from domain experts. Through BERMUDA we can react in time and adjust our event log to prevent concept drift in our discovered process models. We achieved this by keeping track of changes in regulation and in the information system using the BERMUDA triple.

Secondly we showed how understandability is enhanced in discovered declarative process models. We use activity groups both as a semantic choice pattern and a greedy grouping of activities to reduce the number of relations needed to visually represent the declarative process constraints. We showed how the grouped models overall score better on a set of declarative complexity metrics, that were empirically validated, as a proxy for end user understandability.

Thirdly we showed how verifiability is improved, on the one hand, by increased tool support through a mapping of declarative DCR graphs to the well established formalism of Petri nets; and on the other, through the development of verification techniques native to DCR graphs based on test driven modelling and alignments.

Finally we increased the expressiveness of our declarative models by adding time thus being able to capture temporal constraints related to our real-world process. We showed that this increased expressiveness preserves verifiability by mapping timed DCR graphs to a timed version of Petri nets.

6.1 Future work

For the immediate prospects we envision a follow-up empirical evaluation of our BERMUDA method to report qualitative results. We see potential avenues of automation for the BERMUDA tool which will subsequently allow us to investigate the (mis)match between the real-world process, the UI dictated process behaviour and the recorded database behaviour.

Next we acknowledge that our activity groups might introduce cognitive load that is unaccounted for in the initial study [5] and therefore we envision a repeat of the study using also activity groups and subprocesses to investigate if extensions of the complexity metrics are needed. The complexity metrics should then take into account hierarchical process models in their calculation. We envision a future theoretical contribution to the DCR graphs notation that relaxes the definition of activity groups thus allowing activities to be part of multiple non disjoint groups. Thus reaching the theoretical lower bound in the number of relations for our greedy group algorithm, while still maintaining perfect fitness.

The mapping of DCR graphs to Petri nets is currently limited by the space requirements of our algorithm. We wish to investigate divide and conquer algorithms for the construction of our net as well as running the reachability analysis in between the relation construction steps, not just as a post optimization task.

Finally we aim to pave the way towards a native model checking tool for both timed and untimed DCR graphs. By formalizing requirements as open tests and modelling our system as a DCR graph we can use static graph analysis and alignments to verify system properties. But before we can pursue this avenue further we must also establish formally which subsets of temporal logic open tests can support.

For the long term impact of our work, we believe that process mining has the potential to increase the explainability of sub-symbolic machine learning and deep learning-based AI techniques. Process mining can be at the forefront of creating AI-based decision support systems, explainable by design.

Bibliography

[1] In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops : BPM 2013 International Workshops, China, 2013, Revised Papers. pp. 15–27. Lecture Notes in Business Information Processing, Springer, Germany (2014). doi: 10.1007/978-3-319-06257-0-2

[2] van der Aalst, W.: Mining Additional Perspectives, pp. 275–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). doi: 10.1007/978-3-662-49851-49

[3] van der Aalst, W.M.: Foundations of process discovery. In: Process Mining Handbook, pp. 37–75. Springer (2022)

[4] van der Aalst, W.M., Carmona, J.: Process mining handbook. Springer Nature (2022)

[5] Abbad-Andaloussi, A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: Complexity in declarative process models: Metrics and multi-modal assessment of cognitive load. Expert Systems with Applications **233**, 120924 (2023)

[6] Agerwala, T.: A complete model for representing the coordination of asynchronous processes (1974), hopkins Computer Research Report 32

[7] Ancker, J.S., Shih, S., Singh, M.P., Snyder, A., Edwards, A., Kaushal, R., Investigators, H., et al.: Root causes underlying challenges to secondary use of data. In: AMIA Annual Symposium Proceedings. vol. 2011, p. 57. American Medical Informatics Association (2011)

[8] 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). doi: 10.1016/J.IS.2020.101505, https://doi.org/10.1016/j.is.2020.101505

[9] Andaloussi, A.A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: Complexity in declarative process models: Metrics and multi-modal assessment of cognitive load. Expert Syst. Appl. **233**, 120924 (2023). doi: 10.1016/J.ESWA.2023.120924, https://doi.org/10.1016/j.eswa.2023.120924

[10] Andaloussi, A.A., Burattin, A., Slaats, T., Petersen, A.C.M., Hildebrandt, T.T., Weber, B.: Exploring the understandability of a hybrid process design artifact based on DCR graphs. In: Reinhartz-Berger, I., Zdravkovic, J., Gulden, J., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling - 20th International Conference, BPMDS 2019, 24th International Conference, EMMSAD 2019, Held at CAiSE 2019, Rome, Italy, June 3-4, 2019, Proceedings. Lecture Notes in Business Information Processing, vol. 352, pp. 69–84. Springer (2019). doi: 10.1007/978-3-030-20618-5_5, https://doi.org/10.1007/978-3-030-20618-5_5

[11] Andaloussi, A.A., Davis, C.J., Burattin, A., López, H.A., Slaats, T., Weber, B.: Understanding quality in declarative process modeling through the mental models of experts. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12168, pp. 417–434. Springer (2020). doi: 10.1007/978-3-030-58666-9_24, https://doi.org/10.1007/978-3-030-58666-9_24

[12] Andaloussi, A.A., Zerbato, F., Burattin, A., Slaats, T., Hildebrandt, T.T., Weber, B.: Exploring how users engage with hybrid process artifacts based on declarative process models: a behavioral analysis based on eye-tracking and think-aloud. Softw. Syst. Model. **20**(5), 1437–1464 (2021). doi: 10.1007/S10270-020-00811-8, https://doi.org/10.1007/s10270-020-00811-8

[13] Andrews, R., Emamjome, F., ter Hofstede, A.H.M., Reijers, H.A.: An expert lens on data quality in process mining. In: ICPM. pp. 49–56. IEEE (2020)

[14] Augusto, A., Conforti, R., Dumas, M., La Rosa, M.: Split miner: Discovering accurate and simple business process models from event logs. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 1–10. IEEE (2017)

[15] Augusto, A., Mendling, J., Vidgof, M., Wurm, B.: The connection between process complexity of event sequences and models discovered by process mining. Information Sciences **598**, 196–215 (2022)

[16] Back, C.: Hybrid Process Mining: Inference Evaluation Across Imperative Declarative Approaches. Ph.D. thesis (2021)

[17] Back, C.O., Slaats, T., Hildebrandt, T.T., Marquard, M.: Discover: accurate and efficient discovery of declarative process models. In: International Journal on Software Tools for Technology Transfer (2021). doi: 10.1007/s10009-021-00616-0

[18] Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)

[19] Baldan, P., Busi, N., Corradini, A., Michele Pinna, G.: Functional concurrent semantics for petri nets with read and inhibitor arcs. In: CONCUR 2000—Concurrency Theory: 11th International Conference University Park, PA, USA, August 22–25, 2000 Proceedings 11. pp. 442–457. Springer (2000)

[20] Bergenthum, R.: Prime miner-process discovery using prime event structures. In: 2019 International Conference on Process Mining (ICPM). pp. 41–48. IEEE

[21] Berti, A., Van Zelst, S.J., van der Aalst, W.: Process mining for python (pm4py): bridging the gap between process-and data science. arXiv preprint arXiv:1905.06169 (2019)

[22] Bose, J.C.J.C., Mans, R.S., van der Aalst, W.M.P.: Wanna improve process mining results? In: CIDM. pp. 127–134. IEEE (2013)

[23] Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Handling concept drift in process mining. In: CAiSE. Lecture Notes in Computer Science, vol. 6741, pp. 391–405. Springer (2011)

[24] Brons, D., Scheepens, R., Fahland, D.: Striking a new balance in accuracy and simplicity with the probabilistic inductive miner. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 32–39. IEEE (2021)

[25] Buijs, J.C., Van Dongen, B.F., van Der Aalst, W.M.: On the role of fitness, precision, generalization and simplicity in process discovery. In: On the Move to Meaningful Internet Systems: OTM 2012: Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I. pp. 305–322. Springer (2012)

[26] Bushnell, D.M.: Research Conducted at the Institute for Computer Applications in Science and Engineering for the Period October 1, 1999 through March 31, 2000. Technical Report NASA/CR-2000-210105, NAS 1.26:210105, NASA (2000)

[27] Byg, J., Jørgensen, K.Y., Srba, J.: An efficient translation of timed-arc petri nets to networks of timed automata. In: Breitman, K., Cavalcanti, A. (eds.) Formal Methods and Software Engineering. pp. 698–716. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[28] Byg, J., Jørgensen, K.Y., Srba, J.: Tapaal: Editor, simulator and verifier of timed-arc petri nets. In: Automated Technology for Verification and Analysis: 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings 7. pp. 84–89. Springer (2009)

[29] Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Ontology-Based Data Access and Integration, pp. 2590–2596. Springer New York, New York, NY (2018)

[30] Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance checking. Switzerland: Springer.[Google Scholar] **56**, 12 (2018)

[31] Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: Process Mining Handbook, pp. 155–190. Springer (2022)

[32] Chapela-Campa, D., Dumas, M.: Modeling extraneous activity delays in business process simulation. In: 2022 4th International Conference on Process Mining (ICPM). pp. 72–79. IEEE (2022)

[33] Cheikhrouhou, S., Kallel, S., Guermouche, N., Jmaiel, M.: Toward a time-centric modeling of business processes in bpmn 2.0. In: Proceedings of International Conference on Information Integration and Web-Based Applications &; Services. p. 154–163. IIWAS '13, Association for Computing Machinery, New York, NY, USA (2013). doi: 10.1145/2539150.2539182

[34] Christfort, A.K.F., Slaats, T.: Efficient optimal alignment between dynamic condition response graphs and traces. In: International Conference on Business Process Management. pp. 3–19. Springer (2023)

[35] Christfort, A.K.F., Slaats, T.: Efficient optimal alignment between dynamic condition response graphs and traces. In: Di Francescomarino, C., Burattin, A., Janiesch, C., Sadiq, S. (eds.) Business Process Management. pp. 3–19. Springer Nature Switzerland, Cham (2023)

[36] Ciccio, C.D., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Transactions on Management Information Systems (TMIS) **5**(4), 1–37 (2015)

[37] Commission, E.: The ai act, `https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai`

[38] Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers. In: Business Process Management: 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings 12. pp. 101–117. Springer (2014)

[39] Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Bpmn miner: Automated discovery of bpmn process models with hierarchical structure. Information Systems **56**, 284–303 (2016)

[40] Cornanguer, L., Largouët, C., Rozé, L., Termier, A.: TAG: Learning Timed Automata from Logs. In: AAAI 2022 - 36th AAAI Conference on Artificial Intelligence. pp. 1–9. Virtual, Canada (Feb 2022), `https://hal.inria.fr/hal-03564455`

[41] Cosma, V.P., Hildebrandt, T.T., Gyldenkærne, C.H., Slaats, T.: Bermuda: Participatory mapping of domain activities to event data

via system interfaces. In: International Conference on Process Mining. pp. 127–139. Springer (2022)

[42] Cosma, V.P.: Dcr extension to pm4py, `https://github.com/paul-cvp/pm4py-dcr`

[43] Cosma, V.P., Hildebrandt, T.T., Slaats, T.: BERMUDA: towards maintainable traceability of events for trustworthy analysis of non-process-aware information systems. EMISA Forum **41**(1), 33–34 (2021)

[44] Cosma, V.P., Hildebrandt, T.T., Slaats, T.: Transforming dynamic condition response graphs to safe petri nets. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 417–439. Springer (2023)

[45] David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International journal on software tools for technology transfer **17**, 397–415 (2015)

[46] De Cnudde, S., Claes, J., Poels, G.: Improving the quality of the heuristics miner in prom 6.2. Expert Systems with Applications **41**(17), 7678–7690 (2014)

[47] De Giacomo, G., Vardi, M.Y., et al.: Linear temporal logic and linear dynamic logic on finite traces. In: Ijcai. vol. 13, pp. 854–860 (2013)

[48] De Smedt, J., Vanden Broucke, S., De Weerdt, J., Vanthienen, J.: A full r/i-net construct lexicon for declare constraints. Available at SSRN 2572869 (2015)

[49] Debois, S., Hildebrandt, T.T., Laursen, P.H., Ulrik, K.R.: Declarative process mining for dcr graphs. In: Proceedings of SAC. pp. 759–764 (2017)

[50] van Der Aalst, W.M., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development **23**, 99–113 (2009)

[51] van derWerf, J.M.E., van Dongen, B.F., Hurkens, C.A., Serebrenik, A.: Process discovery using integer linear programming. Fundamenta Informaticae **94**(3-4), 387–412 (2009)

[52] Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of bpmn process models using petri nets. Queensland University of Technology, Tech. Rep pp. 1–30 (2007)

[53] van Dongen, B.F., Weber, B., Ferreira, D.R., De Weerdt, J.: Report: business process intelligence challenge 2013. In: Business Process Management Workshops: BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers 11. pp. 79–87. Springer (2014)

[54] Dumas, M., Rosa, L.M., Mendling, J., Reijers, A.H.: Fundamentals of business process management. Springer (2018)

[55] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st international conference on Software engineering. pp. 411–420 (1999)

[56] Emamjome, F., Andrews, R., ter Hofstede, A.H.M., Reijers, H.A.: Alohomora: Unlocking data quality causes through event log context. In: ECIS (2020)

[57] Eshuis, R., Debois, S., Slaats, T., Hildebrandt, T.: Deriving consistent gsm schemas from dcr graphs. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) Service-Oriented Computing. pp. 467–482. Springer International Publishing, Cham (2016)

[58] Finkel, O.: On the high complexity of petri nets ω-languages. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 69–88. Springer (2020)

[59] Fischer, D.A., Goel, K., Andrews, R., van Dun, C.G.J., Wynn, M.T., Röglinger, M.: Enhancing event log quality: Detecting and quantifying timestamp imperfections. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management. pp. 309–326. Springer International Publishing, Cham (2020)

[60] Gagne, D., Trudel, A.: Time-bpmn. In: 2009 IEEE Conference on Commerce and Enterprise Computing. pp. 361–367 (2009). doi: 10.1109/CEC.2009.71

[61] Goodman, B., Flaxman, S.: European union regulations on algorithmic decision-making and a "right to explanation". AI magazine **38**(3), 50–57 (2017)

[62] Gunning, D., Aha, D.: Darpa's explainable artificial intelligence (xai) program. AI magazine **40**(2), 44–58 (2019)

[63] Günther, C.W., Van Der Aalst, W.M.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: International conference on business process management. pp. 328–343. Springer (2007)

[64] H. Gyldenkaerne, C., From, G., Mønsted, T., Simonsen, J.: Pd and the challenge of ai in health-care. In: Proceedings of the 16th Participatory Design Conference 2020-Participation (s) Otherwise-Volume 2. pp. 26–29 (2020)

[65] Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., Weber, B.: Understanding declare models: strategies, pitfalls, empirical results. Software & Systems Modeling **15**(2), 325–352 (2016)

[66] Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: International conference on fundamentals of software engineering. pp. 343–350. Springer (2011)

[67] Hildebrandt, T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. The Journal of Logic and Algebraic Programming **82**(5-7), 164–185 (2013)

[68] Hildebrandt, T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. The Journal of Logic and Algebraic Programming **82**(5), 164–185 (2013). doi: https://doi.org/10.1016/j.jlap.2013.05.005, `https://www.sciencedirect.com/science/article/pii/S1567832613000283`, formal Languages and Analysis of Contract-Oriented Software (FLACOS'11)

[69] Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Places (2011), https://api.semanticscholar.org/CorpusID:14353309

[70] Hu, Z., Shatz, S.M.: Mapping uml diagrams to a petri net notation for system simulation. In: SEKE. pp. 213–219. Citeseer (2004)

[71] Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of WS-FM'10. pp. 1–24. Springer-Verlag, Berlin, Heidelberg (2011)

[72] Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of timed-arc petri nets. In: SOFSEM 2011: Theory and Practice of Computer Science: 37th Conference on Current Trends in Theory and Practice of Computer Science, Novỳ Smokovec, Slovakia, January 22-28, 2011. Proceedings 37. pp. 46–72. Springer (2011)

[73] Jagadeesh Chandra Bose, R., Van der Aalst, W.M.: Abstractions in process mining: A taxonomy of patterns. In: Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, 2009. Proceedings 7. pp. 159–175

[74] Jagadeesh Chandra Bose, R., van der Aalst, W.M.: Process diagnostics using trace alignment: Opportunities, issues, and challenges. Information Systems **37**(2), 117–141 (2012), management and Engineering of Process-Aware Information Systems

[75] Jans, M., Soffer, P.: From relational database to event log: Decisions with quality impact. In: Business Process Management Workshops. Lecture Notes in Business Information Processing, vol. 308, pp. 588–599. Springer (2017)

[76] Leemans, M., Van Der Aalst, W.M., Van Den Brand, M.G.: Recursion aware modeling and discovery for hierarchical software event log analysis. In: 2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER). pp. 185–196. IEEE (2018)

[77] Leemans, S.J., Fahland, D., Van Der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops: BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers 11. pp. 66–78. Springer (2014)

[78] Leemans, S.J., Goel, K., van Zelst, S.J.: Using multi-level information in hierarchical process mining: Balancing behavioural quality and model complexity. In: 2020 2nd International Conference on Process Mining (ICPM). pp. 137–144

[79] Leopold, H., van der Aa, H., Pittke, F., Raffel, M., Mendling, J., Reijers, H.A.: Searching textual and model-based process descriptions based on a unified data format. Softw. Syst. Model. **18**(2), 1179–1194 (2019)

[80] Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: CAiSE Forum. Lecture Notes in Business Information Processing, vol. 317, pp. 182–199. Springer (2018)

[81] Lopez, H.A.: Javascript dcr editor, `https://dcr-js.github.io/DCRjs/`

[82] Lu, X., Gal, A., Reijers, H.A.: Discovering hierarchical processes using flexible activity trees for event abstraction. In: 2020 2nd International Conference on Process Mining (ICPM). pp. 145–152. IEEE (2020)

[83] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 4768–4777. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)

[84] Maggi, F.M., Bose, R.J.C., van der Aalst, W.M.: Efficient discovery of understandable declarative process models from event logs. In: Advanced Information Systems Engineering: 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings 24. pp. 270–285. Springer (2012)

[85] 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.) Business Process Management. pp. 132–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[86] Mannel, L.L., van der Aalst, W.M.: Finding uniwired petri nets using est-miner. In: Business Process Management Workshops: BPM 2019 International Workshops, Vienna, Austria, September 1–6, 2019, Revised Selected Papers 17. pp. 224–237. Springer (2019)

[87] Mannhardt, F.: Responsible process mining. In: Process Mining Handbook, pp. 373–401. Springer International Publishing Cham (2022)

[88] Mannhardt, F., De Leoni, M., Reijers, H.A., Van Der Aalst, W.M.: Balanced multi-perspective checking of process conformance. Computing **98**, 407–437 (2016)

[89] Mannhardt, F., De Leoni, M., Reijers, H.A., Van Der Aalst, W.M., Toussaint, P.J.: From low-level events to activities-a pattern-based approach. In: Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, 2016. Proceedings 14. pp. 125–141. Springer

[90] Martin, N., Wittig, N., Munoz-Gama, J.: Using process mining in healthcare. In: Process mining handbook, pp. 416–444. Springer International Publishing Cham (2022)

[91] Meystre, S.M., Lovis, C., Bürkle, T., Tognola, G., Budrionis, A., Lehmann, C.U.: Clinical data reuse or secondary use: current status and potential future progress. Yearbook of medical informatics **26**(01), 38–52 (2017)

[92] Montali, M.: Specification and verification of declarative open interaction models: a logic-based approach, vol. 56. Springer Science & Business Media (2010)

[93] Montali, M., Pesic, M., Aalst, W.M.P.v.d., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. ACM Trans. Web **4**(1) (jan 2010). doi: 10.1145/1658373.1658376, https://doi.org/10.1145/1658373.1658376

[94] Montanari, U., Rossi, F.: Contextual nets. Acta Informatica **32**, 545–596 (1995)

[95] Moya, M.M., Hush, D.R.: Network constraints and multi-objective optimization for one-class classification. Neural networks **9**(3), 463–474 (1996)

[96] Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (2012)

[97] Mukkamala, R.R., Hildebrandt, T.T.: From dynamic condition response structures to büchi automata. In: 2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering. pp. 187–190 (2010). doi: 10.1109/TASE.2010.22

[98] multiple: Dcr graphs portal, `https://dcrgraphs.net/`

[99] Munoz-Gama, J., Martin, N., Fernandez-Llatas, C., Johnson, O.A., Sepúlveda, M., Helm, E., Galvez-Yanjari, V., Rojas, E., Martinez-Millana, A., Aloini, D., et al.: Process mining for healthcare: Characteristics and challenges. Journal of Biomedical Informatics **127**, 103994 (2022)

[100] Nekrasaite, V., Parli, A.T., Back, C.O., Slaats, T.: Discovering responsibilities with dynamic condition response graphs. In: Advanced Information Systems Engineering: 31st International Conference, CAiSE 2019. pp. 595–610. Springer

[101] Normann, H., Debois, S., Slaats, T., Hildebrandt, T.T.: Zoom and enhance: Action refinement via subprocesses in timed declarative processes. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) Business Process Management. pp. 161–178. Springer International Publishing, Cham (2021)

[102] Pesic, M.: Constraint-based workflow management systems: shifting control to users (2008)

[103] Pesic, M., Schonenberg, H., Van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: 11th IEEE international enterprise distributed object computing conference (EDOC 2007). pp. 287–287. IEEE

[104] Petri, C.A.: Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn (1962)

[105] Popova, V., Dumas, M.: From petri nets to guard-stage-milestone models. In: La Rosa, M., Soffer, P. (eds.) Business Process Management Workshops. pp. 340–351. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[106] Popova, V., Fahland, D., Dumas, M.: Artifact lifecycle discovery. International Journal of Cooperative Information Systems **24**(01), 1550001 (2015)

[107] Prescher, J., Di Ciccio, C., Mendling, J., et al.: From declarative processes to imperative models. SIMPDA **1293**, 162–173 (2014)

[108] Raedts, I., Petkovic, M., Usenko, Y.S., van der Werf, J.M.E., Groote, J.F., Somers, L.J.: Transformation of bpmn models for behaviour analysis. MSVVEIS **2007**, 126–137 (2007)

[109] Ribeiro, M.T., Singh, S., Guestrin, C.: " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1135–1144 (2016)

[110] Rojas, E., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Process mining in healthcare: A literature review. Journal of biomedical informatics **61**, 224–236 (2016)

[111] Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Pearson (2016)

[112] Sànchez-Ferreres, J., van der Aa, H., Carmona, J., Padró, L.: Aligning textual and model-based process descriptions. Data Knowl. Eng. **118**, 25–40 (2018)

[113] Schmidt, K.: Lola a low level analyser. In: Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000 Aarhus, Denmark, June 26–30, 2000 Proceedings 21. pp. 465–474. Springer (2000)

[114] Schön, D.A.: The reflective practitioner: How professionals think in action. Routledge (2017)

[115] Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: A framework for efficiently mining the organisational perspective of business processes. Decision Support Systems **89**, 87–97 (2016)

[116] Sergey Smirnov, Hajo A. Reijers, M.W.T.N.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases **30**, 63 – 99 (2012)

[117] Simonsen, J., Robertson, T.: Routledge international handbook of participatory design, vol. 711. Routledge New York (2013)

[118] Slaats, T.: Flexible Process Notations for Cross-organizational Case Management Systems. Ph.D. thesis, IT University of Copenhagen (2015)

[119] Slaats, T., Debois, S., Hildebrandt, T.: Open to change: A theory for iterative test-driven modelling. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) Business Process Management. pp. 31–47. Springer International Publishing, Cham (2018)

[120] Smylie, J., Firestone, M.: Back to the basics: Identifying and addressing underlying challenges in achieving high quality and relevant health statistics for indigenous populations in canada. Statistical Journal of the IAOS **31**(1), 67–87 (2015)

[121] Srba, J.: Timed-arc Petri nets vs. networks of timed automata. In: Proceedings of the 26th International Conference on Application and Theory of Petri Nets (ICATPN'05). LNCS, vol. 3536, pp. 385–402. Springer-Verlag (2005)

[122] Staines, T.S.: Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. In: 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008). pp. 191–200 (2008). doi: 10.1109/ECBS.2008.12

[123] Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)

[124] Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M., Norre, S.: Interest-driven discovery of local process models. Information Systems **77** (2018)

[125] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Event abstraction for process mining using supervised learning techniques. In: Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 1. pp. 251–269. Springer (2018)

[126] Thapa, D., Dangol, S., Wang, G.N.: Transformation from petri nets model to programmable logic controller using one-to-one mapping technique. In: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). vol. 2, pp. 228–233 (2005). doi: 10.1109/CIMCA.2005.1631473

[127] Turetken, O., Dikici, A., Vanderfeesten, I., Rompen, T., Demirors, O.: The influence of using collapsed sub-processes and groups on the understandability of business process models. Business & Information Systems Engineering **62**, 121–141 (2020)

[128] Valk, R.: Infinite behaviour of petri nets. Theoretical computer science **25**(3), 311–341 (1983)

[129] Van Der Aalst, W., van der Aalst, W.: Data science in action. Springer (2016)

[130] Van Eck, M.L., Lu, X., Leemans, S.J., Van Der Aalst, W.M.: Pm: a process mining project methodology. In: International conference on advanced information systems engineering. pp. 297–313. Springer (2015)

[131] Verbeek, E.: Discovering an s-coverable wf-net using discover. In: 2022 4th International Conference on Process Mining (ICPM). pp. 64–71. IEEE (2022)

[132] Weijters, A., Ribeiro, J.T.S.: Flexible heuristics miner (fhm). In: 2011 IEEE symposium on computational intelligence and data mining (CIDM). pp. 310–317. IEEE (2011)

[133] Weijters, A.J., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristicsminer algorithm (2006)

[134] van der Werf, J.M.E.M., Mans, R., van der Aalst, W.M.P.: Mining declarative models using time intervals. In: Moldt, D. (ed.) Joint Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'13) and the International Workshop on Modeling and Business Environments (ModBE'13), Milano, Italy, June 24 - 25, 2013. CEUR Workshop Proceedings, vol. 989, pp. 313–331. CEUR-WS.org (2013), http://ceur-ws.org/Vol-989/paper04b.pdf

[135] Weske, M., et al.: Concepts, languages, architectures. Business Process Management (2007)

[136] Wolf, K.: How petri net theory serves petri net model checking: a survey. Transactions on Petri Nets and Other Models of Concurrency XIV pp. 36–63 (2019)

[137] Yang, N., Yu, H., Sun, H., Qian, Z.: Modeling uml sequence diagrams using extended petri nets. In: 2010 International Conference on Information Science and Applications. pp. 1–8 (2010). doi: 10.1109/ICISA.2010.5480384

[138] van Zelst, S., Mannhardt, F., de Leoni, M.: Event abstraction in process mining: literature review and taxonomy, vol. 6, pp. 719–736. Springer New York, New York, NY (2021). doi: 10.1007/s41066-020-00226-2

[139] Zhang, Z., Guo, C., Ren, S.: Mining timing constraints from event logs for process model. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). pp. 1011–1016. IEEE (2020)

[140] Zugal, S., Pinggera, J., Weber, B.: The impact of testcases on the maintainability of declarative process models. Enterprise, Business-Process and Information Systems Modeling pp. 163–177 (2011)

[141] Zugal, S., Pinggera, J., Weber, B.: Creating declarative process models using test driven modeling suite. In: CAiSE Forum 2011. pp. 16–32 (2012). doi: 10.1007/978-3-642-29749-6_2

Part II

Papers

Chapter 7

Trustworthy event log extraction

7.1 BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces

Remark 7.1. *The work in this chapter has been published as [41]: Cosma VP, Hildebrandt TT, Gyldenkærne CH, Slaats T. BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces. International Conference on Process Mining 2022 Oct 23 (pp. 127-139). Springer Nature Switzerland.*

Part of the material also appears in a two page extended abstract (not part of this thesis) which has been published as [43]: Cosma VP, Hildebrandt TT, Slaats T. BERMUDA: towards maintainable traceability of events for trustworthy analysis of non-process-aware information systems. Emisa Forum Volume 41, Number 1, Pages 33–34, Year 2021.

BERMUDA: Participatory Mapping of Domain Activities to Event Data via System Interfaces*

Vlad P. Cosma[1,2][0000−0001−8022−6402], Thomas T. Hildebrandt[1][0000−0002−7435−5563], Christopher H. Gyldenkærne[3][0000−0003−2858−7328], and Tijs Slaats[1][0000−0001−6244−6970]

[1] Copenhagen University, Copenhagen 2200, Denmark
{vco,hilde,slaats}@di.ku.dk
[2] KMD ApS, Ballerup 2750, Denmark
vco@kmd.dk
[3] Roskilde University, Denmark
chrgyl@ruc.dk

Abstract. We present a method and prototype tool supporting participatory mapping of domain activities to event data recorded in information systems via the system interfaces. The aim is to facilitate responsible secondary use of event data recorded in information systems, such as process mining and the construction of predictive AI models. Another identified possible benefit is the support for increasing the quality of data by using the mapping to support educating new users in how to register data, thereby increasing the consistency in how domain activities are recorded. We illustrate the method on two cases, one from a job center in a danish municipality and another from a danish hospital using the healthcare platform from Epic.

Keywords: Data quality · Secondary use · Event extraction · Event matching · Participatory Design

1 Introduction

The abundance of data recorded in information systems and easily accessible technologies for data processing, such as predictive AI models and process mining [1,2], have created huge expectations of how data science can improve the society.

However, there has also been an increasing voicing of concerns [3, 11, 18, 39], pointing out that merely having access to data and technologies is not sufficient to guarantee improvements. In the present paper we focus on data quality and responsible event extraction in the context of secondary use of event data [34] recorded in information systems. That is, data representing events in the domain of use, such as the start and completion of work tasks which has as primary use to support case workers and document the progress of a case, but is intended to be used for secondary purposes, such as building predictive AI models or the discovery of processes using process mining tools.

* A 2-page extended abstract presenting early ideas of the paper was published in [13].

The challenges of event data quality are manifold [9], including handling event granularity, incorrect or missing data and incorrect timestamps of events [17]. A more fundamental problem in the context of secondary use of event-data is that of ensuring a consistent and correct matching of event data to business activities [7].

The lack of research in the area of event log creation has been pointed out in several papers [2,7,9,16,21,26,29,30,36,38]. This task is in general associated with words and expressions like: costly, time consuming, tedious, unstructured, complex, garbage-in garbage-out. Historically, research for data-driven innovation and improving productivity has shown to pay little to no attention to how data is created and by who. Data is often created within a system and its user interface where a given context for capturing and using data has been established through continuous sense-making between people that have local and often individual understanding of why data is generated and for what. Studies claim [22, 41]that data science initiatives are often initiated at high-level and allocated from domain of data creation while the data science product is re-introduced as a model that needs to be adapted by the practice where data is created. While data driven systems can be evaluated with good results on artificial data from the data domain, it is often a struggle to create value for the domain users. This is due to trust of data origin, what it represents and how new intents for its purpose comes through what could be considered a back-door top-down method. A Participatory Design(PD)-study [18] investigated a mismatch between data extraction findings at an administration level of cross-hospital management and how doctors and clinical secretaries represented their ways of submitting data, highlighting a need for re-negotiating data creation and its purpose in a way so data scientists can contribute to better data capture infrastructures as well as giving health-care workers a saying in how such data capture infrastructures are prioritized in their given domains of non-digital work. In PD [8, 23, 32] as a field such presented tensions are not new. Here PD as a design method and practice has sought to create alignment between workers existing understanding of own work and emerging systems through design as a practice for visualising such tensions across actors of an innovation or IT project. PD is from here seeking, in a democratic manner, to find solutions and interests that can match partners across hierarchies.

As a means to facilitate responsible secondary use of event data, we propose in this paper the BERMUDA (*Business Event Relation Map via User-interface to Data for Analysis*) method to capture and maintain the link between domain knowledge and the data in the information system. The method supports involvement of domain experts in the mapping of activities or events in the business domain to user-interface elements, and of system engineers in the mapping of user-interface elements to database records used by data scientists. In other words, the method helps documenting the inter-relationship in the "BERMUDA triangle" between the domain concepts, the user interface and the database, which often disappears. We see that by breaking down the barrier between data-creators and data scientists and building tools for involvement and iterative

feedback of data infrastructures and their user front-end, new discussions for data cooperation can occur. The mapping is independent of any specific data analysis, but should of course include the activities and events of relevance for the analysis at hand. In particular, the method contributes to the responsible application of process mining [27] by supporting a collaborative creation of event logs.

The motivation for the method came from research into the responsible engineering of AI-based decision support tools in Danish municipalities within the EcoKnow [19] research project and later the use of the method was also found relevant in a study of a Danish hospital wanting to create an AI-based predictive model for clinical no-shows. The method and prototype were initially evaluated by a consultant employed in a process mining company and a municipal case worker collaborating with the authors in the EcoKnow research project.

The paper is structured as follows. Prior and related work is discussed in Sec. 2. Sec. 3 explains our proposed BERMUDA method, where we also show a prototype tool. Sec. 4 introduces two specific case studies in a job center and a danish hospital. A brief evaluation of the use of the method in the first case along with a discussion on the results is made in Sec. 5. Lastly, in Sec. 6 we conclude and discuss future work.

2 Prior and Related work

Within health-care informatics, problems arising from having a primary use of data (original intend of health-care delivery and services) and different, secondary use of data (emergence of new possibilities through statistics and data science) has been highlighted in several studies [5, 28, 37]. The authors of [5] found that underlying issues for data quality and reuse was attributed to differential incentives for the accuracy of the data; flexibility in system software that allowed multiple routes to documenting the same tasks; variability in documentation practices among different personnel documenting the same task; variability in use of standardized vocabulary, specifically, the internally developed standardized vocabulary of practice names; and changes in project procedures and electronic system configuration over time, as when a paper questionnaire was replaced with an electronic version.

Such underlying socio-technical issues to data capturing can attribute to an overall lower degree of data integrity resulting in little to no secondary usefulness of data representing health-care events. A similar [18] study conducted by this papers co-authors highlighted the need for iteratively aligning data creation and use with domain experts and data creators (i.e. doctors, nurses, secretaries, etc) when conducting data science on operational data from hospitals.

We see event abstraction [40] as a related topic to our paper, however we approach the problem in a top-down manner i.e. from domain knowledge down to the data source. A similar top-down approach exists in database systems [12] where an ontology of domain concepts is used to query the databases. We do not aim to propose techniques for process discovery as there are a plethora of tools

already in use for this task, some of which [35] also allow for domain expert interventions. We propose BERMUDA both for pre-processing of data before moving to process discovery or building predictive models, and for training of new users in how to consistently record data suitable for the secondary uses.

The paper [21] provides a procedure for extracting event logs from databases that makes explicit the decisions taken during event log building and demonstrates it through a running example instead of providing tool support. The paper [7] present a semi-automatic approach that maps events to activities by transforming the mapping problem into the a constraint satisfaction problem, but it does not directly handle the event log extraction.

In [29] the authors describe a meta model that separates the extraction and analysis phases and makes it easy to connect event logs with SQL queries. In [30] they associate events from different databases into a single trace and propose an automated event log building algorithm. They point towards the lack of domain knowledge as a driving force for an automated and efficient approach. They discuss that their definition of event log "interestingness" as an objective score ignores aspects of domain level relevance. Both papers bind database scripts and event log concepts in order to build ontologies/meta-models, but do not link to domain knowledge in order to provide traceability to domain experts, such that the limitations of the "interestingness" score may be overcome.

To summarize, most work [6, 9, 10, 16, 17, 24, 25, 33, 38] on event data quality so far has focused on technical means to repair and maintain the quality of event logs [15]. Our approach complements these approaches by focusing on the socio-technical problem of aligning what is done in practice by the users of the information systems, i.e. how is a domain activity registered within the system, and at the other hand, where is this event stored in the database.

3 BERMUDA: Mapping domain events to data

Our method relies on so-called BERMUDA triples **(e,i,d)** as illustrated in Fig.1, recording the relation between respectively a domain event **e**, a user interface element **i** of the information system in which the domain event is registered and the location of the resulting data element **d** in the database. A concrete example from one of our case studies can be seen in Fig.2. Here a domain event "Register ... during the first interview" is described in a textual audit schema. This is linked by a screen shot to the drop down menu in the user interface, where the case worker performs this concrete registration. And finally, the location of the resulting data element is recorded by an SQL statement that extracts the event.

There are typically three roles involved in the recording such BERMUDA triples: Data scientist (or analyst), domain expert and system engineer. As guidance towards applying our method we recommend following these steps:

1. **Domain to user interface**. For each domain event **e**, the domain experts record an association **(e,i)** between the domain event **e** and an (user or system) interface element **i**.

Fig. 1. BERMUDA method

2. **User interface to data**. Through code inspection or simulation, system engineers develop the correct database query **d** to extract the data recording the event **e** created via the interface element **i** resulting in a triple **(e,i,d)**.
3. **Triples to event log**. The data scientist merges and refines the database queries and creates the initial version of the event log. The event log entries are enriched with extra attributes that hold a reference to the domain event, the interface element and the data source from where the entry originated.

Prototype tool. To facilitate the adoption of the BERMUDA method we present a prototype tool to illustrate how the triples can be created and an event log extracted. A screenshot from the prototype is shown in Fig. 2. Briefly, the UI consists of 3 input areas in the top for documenting the individual parts of triples (description of domain event, system interface, script for extracting the event from the system), an input area at the bottom for adding and selecting a

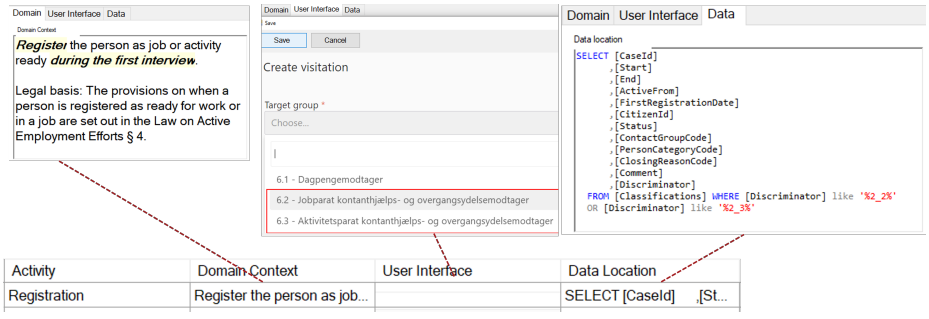


Fig. 2. BERMUDA method Prototype

triple to document, and a display area (not shown in the figure) for the resulting event log. [4]

The prototype has a simple role base access control supporting the use of the method in practice. All roles have access to the description of domain events, in order to build trust through a common domain understanding. Domain experts have access to domain events and the user interface input areas. System engineers need access to all areas, but not the production data in the information system. Data scientists are allowed access to all areas except they can not see the data extraction scripts, if they are covered by intellectual propriety rights. They can however run the scripts on the production system, to extract the event data.

4 Cases: Secondary use of Municipal and Health data

We discuss the method in relation to two concrete cases from Denmark where data in respectively a municipality and a hospital were intended to be used for AI-based decision support. Case 1 is elicited at a municipal job center in Denmark and case 2 covers our work with a regional research hospital where a project aiming for producing and using an AI model for no-shows. Both cases unveiled a gap between how data is produced in a local context for its primary purpose of case management and what it represents when extracted and used for decision support. We made an evaluation of our BERMUDA prototype for case one and speculate how it could be used in case two.

Case 1: As part of the EcoKnow research project [19], we had by the software vendor KMD (kmd.dk), been given access to interact with the system engineers that developed the case management system used in danish job centers. Collaborating with colleagues in the EcoKnow research project performing field studies at the job center [4,20,31], we also had the opportunity to gather domain knowledge through workshops, semi-structured interviews and informal methods from job center employees. Finally, we had access to historical data from about 16000 citizens with the purpose of researching the possibilities for improving compliance and the experienced quality of case management in municipalities.

In addition to our case we interviewed a consultant at a process mining company Infoventure (infoventure.dk), doing conformance checking, using the same case management system but a different data source. Their current practice relies on first co-creating a document with employees at the job center, which contained the necessary domain knowledge and screenshots of user interface elements with relevant explanations. Next it was the task of the consultant to build extraction scripts for the identified domain events. During this phase there was ongoing communication with the software vendor and job center employees through meetings, calls or emails, in order to build up the necessary domain and system knowledge. Often he would observe specific data (an exact timestamp or citizen registration number) in the user interface and proceed to search for that exact information in the database. This process was done either offline, with the

[4] The prototype is available at: `https://github.com/paul-cvp/bermuda-method`.

aid of screenshots, or on site by sitting next to a case worker. The links between domain events and the data extracted from the database was recorded in an ad-hoc way and only available to the consultant.

Domain Activities/events: We used a management audit schema comprised of 21 questions. From these questions we define the domain activities/events relevant for the case compliance analysis. For example: From the audit question "Is the first job interview held within one week of the first request? Legal basis: LAB § 31(3)" we can identify several domain event data of interest: first request, first job interview, first week passed.

Graphical User Interface (GUI) Areas for recording domain events A caseworker employed at the job center associated the domain events identified in the audit questions with areas of the user interface where caseworkers record the event. From the 21 questions, 11 domain events could be identified that could be given a user interface association. For 3 of the domain events, the caseworker was unsure where to record it. A data scientist was able to associate 12 of the 21 domain events to a field in the user interface. This relatively low number of associations can be explained by the fact that the audit schema was created by the municipality and not the vendor of the it-system, and thus, some of the domain events relevant for the audit did not have a direct representation in the user interface. Therefore certain events were completely missing or documented in free text fields, while others require access to other systems used by the municipality. In particular, as also observed in [4], the free text field was sometimes used to describe the categorisation of the unemployed citizen (as activity or job ready) or the reason for the choice of categorisation, by selecting the reason "other", instead of using one of the specific predefined values available in the system interface.

Data and database organization. The database contains 133 tables with 1472 columns in total. By having access to source code and the system engineers, we mapped the identified GUI elements to the database. Furthermore this limited our inspection to 8 main tables from which the data was extracted and 4 tables used for mapping table relations, thus ensuring data minimisation as specified in the General Data Protection Regulation (GDPR) [14].

Case 2: In the wake of a grand scale implementation of an EPIC[5] Regional Electronic Health Record-system (EHR-system) purchase and implementation, we have since 2017 been engaged in a longitudinal case-study of facilitating and developing an AI-model for predicting patient no-shows based on clinical event and demographic data. The project was pioneering as the first test of the models developed from local data and appointed a small endoscopy unit at Bispebjerg hospital (a research hospital in the capital region of Denmark). The project have a foundation in participatory design and end-user involvement in pursuit

[5] epic.com

of creating visions for use of data and AI, as well as creating synergy effects for data creation among clinicians, nurses and clinical secretaries as domain experts creating clinical event data used to predict future no-shows.

We extracted 8 different data sets together with the regional data team to learn about implications for applying such data for machine-learning purposes. We here learned, that missing data values and incomplete submissions were largely representing the first data sets and that due to missing guidelines and coordinated workflows each individual health care person had different understanding of the categories used to report clinical appointment statuses.

Domain Events: Interpretations of the events. We conducted 2 follow-up interviews with clinical secretaries to understand the local flow of data submission into the EHR-system. The clinical secretaries demonstrated their data submission practices and their understanding of how to document clinical appointment statuses into the EHR-system. We further conducted four 2-hour workshops involving the clinical secretaries in putting context to their workflow and use of categories to assign meaning to no-show categories. In the same period, we invited Regional data management and extraction teams to learn from practices and iteratively extract data sets with no-show data.

Data and database organization. 8 data sets were extracted in total over a period of 3 months before a machine learning algorithm could be fed with a data set with sufficient domain contexts to remove categories that didn't have meaning for secondary use. The best example of this was again the free text category "other" as a category for assigning reason for no-shows or cancellations of appointments. This category was heavily used by all clinical staff due to its ability to avoid reading through 16 other categories of reason for mentioned outcome. The first data set had 81.000 rows and observations with 2/3 of those past appointments being assigned "other" with text-field inputs sometimes representing the same categories as suggested in the drop-down menu and sometimes left empty or with "other" written in the text-field. A further 11.000 appointments were deemed incomplete or "in process" several months after appointment date. When sorting out unassigned events for appointment status the department only had 2880 observations left for the machine learning algorithm.

5 Initial Evaluation

As an initial qualitative evaluation of the usefulness of the method, we conducted two semi-structured interviews, one with a municipal case worker acting as a domain expert and another with a data scientist working as consultant in the process analysis company Infoventure. Both interview respondents collaborated with the authors in the Ecoknow research project. The municipal case worker was given the task of mapping business activities to user-interface elements of a case and document management system. The consultant was asked about the current practice of documenting event log extraction for process mining, illustrated by

a concrete case, and how the Bermuda prototype could support or improve this practice.

Overall, the evaluation indicated, that the BERMUDA method exhibits the following positive proprieties:

- **Transparency, Accountability, and Traceablity.** The BERMUDA triples make it possible to trace the relation between events extracted from a data base, e.g. for the creation of an event log, and domain events. Both interviewees saw the advantage in unambiguously referencing domain events across different roles of a data science project (domain expert, software engineer, data scientist), thereby providing accountability for the data provenance/lineage, while also building trust across different roles.
- **Accuracy.** Through the participatory co-creation of the event log it is possible to observe that the event log correctly captures the relevant domain knowledge. As each of the roles interact with each other, they can observe that the correct steps were taken in the extraction of event data for secondary use. This was already to some extend part of the current practices, but BERMUDA supported the consistent documentation.
- **Maintainability and Training.** The interview participants indicated that the Bermuda method is useful for maintaining event logs over time when changes happen in the domain or system, because the information is documented consistently in one place. They also pointed out, that the method and tool for the same reason could be valuable both in training new data scientists and new case workers.
- **Protection of Intellectual Property.** Since each link in the BERMUDA triangle can be defined independently, the system engineers can provide mappings that can be used to extract events without revealing the code of the system. We observed this in the interaction between the data science consultant and the system engineers developing the job center solution.

Limitations. Firstly, the tool is not mature enough to replace a general SQL scripting environment. Secondly, it does not yet account for data that are not stored in an SQL database, nor for data that is not recorded via user interface, as for instance data recorded automatically by system events.

6 Conclusion and future work

In this paper we presented BERMUDA, a method for facilitating the responsible secondary use of event data in data science projects by supporting the collaboration between domain experts, system engineers and data scientists on associating domain events, via user interfaces to data in the database. This facilitates transparent extraction of event logs for analysis and thereby accountable data lineage. We discussed its use through cases of data science projects at a job center in a Danish municipality and a Danish hospital. In particular, both cases highlight the frequent use of the category “other” in the registration of reasons for domain events, instead of using pre-defined values in drop down menus. We

showed through a prototype tool how BERMUDA can facilitate the interactions between domain experts, system engineers and data scientists. Furthermore we conducted interviews in order to lightly evaluate its usefulness and limitations.

In the future we expect to conduct more field trials of the method and interview more practitioners in order to do a thematic analysis for better qualitative feedback. We aim to investigate how the results of applying BERMUDA can be used when training domain experts to use the appropriate categories instead of "other". We also aim to extend the tool with an automatic signaling system to monitor for changes in the user interface and in the database structure to notify the data scientist of possible misalignment in existing processes. We hope to increase the robustness of the tool and its compatibility with existing process mining tools. We also aim to provide the prototype as an online tool in order to facilitate remote cooperative work. Finally we aim to support a broader range of input and output formats by applying the method on diverse data sources from information systems in relevant domains.

Acknowledgements

Thanks to Infoventure, KMD Momentum, Bispebjerg Hospital, The Capital Region of Denmark, Gladsaxe and Syddjurs municipalities, and the reviewers.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. van der Aalst, W.M.P., et. al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I. Lecture Notes in Business Information Processing, vol. 99, pp. 169–194. Springer (2011)
3. on AI, H.L.E.G.: Ethics guidelines for trustworthy ai, `https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai`
4. Ammitzbøll Flügge, A., Hildebrandt, T., Møller, N.H.: Street-level algorithms and ai in bureaucratic decision-making: A caseworker perspective. Proc. ACM Hum.-Comput. Interact. **5**(CSCW1) (apr 2021). https://doi.org/10.1145/3449114, `https://doi.org/10.1145/3449114`
5. Ancker, J.S., Shih, S., Singh, M.P., Snyder, A., Edwards, A., Kaushal, R., Investigators, H., et al.: Root causes underlying challenges to secondary use of data. In: AMIA Annual Symposium Proceedings. vol. 2011, p. 57. American Medical Informatics Association (2011)
6. Andrews, R., Emamjome, F., ter Hofstede, A.H.M., Reijers, H.A.: An expert lens on data quality in process mining. In: ICPM. pp. 49–56. IEEE (2020)
7. Baier, T., Rogge-Solti, A., Weske, M., Mendling, J.: Matching of events and activities - an approach based on constraint satisfaction. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) The Practice of Enterprise Modeling. pp. 58–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

8. Björgvinsson, E., Ehn, P., Hillgren, P.A.: Participatory design and" democratizing innovation". In: Proceedings of the 11th Biennial participatory design conference. pp. 41–50 (2010)
9. Bose, J.C.J.C., Mans, R.S., van der Aalst, W.M.P.: Wanna improve process mining results? In: CIDM. pp. 127–134. IEEE (2013)
10. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Handling concept drift in process mining. In: CAiSE. Lecture Notes in Computer Science, vol. 6741, pp. 391–405. Springer (2011)
11. Cabitza, F., Campagner, A., Balsano, C.: Bridging the "last mile" gap between ai implementation and operation:"data awareness" that matters. Annals of translational medicine **8**(7) (2020)
12. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Ontology-Based Data Access and Integration, pp. 2590–2596. Springer New York, New York, NY (2018)
13. Cosma, V.P., Hildebrandt, T.T., Slaats, T.: Bermuda: Towards maintainable traceability of events for trustworthy analysis of non-process-aware information systems. In: EMISA Forum: Vol. 41, No. 1. De Gruyter (2021)
14. Council of European Union: Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data. `https://publications.europa.eu/s/llVw` (May 2016)
15. De Weerdt, J., Wynn, M.T.: Foundations of process event data. Process Mining Handbook. LNBIP **448**, 193–211 (2022)
16. Emamjome, F., Andrews, R., ter Hofstede, A.H.M., Reijers, H.A.: Alohomora: Unlocking data quality causes through event log context. In: ECIS (2020)
17. Fischer, D.A., Goel, K., Andrews, R., van Dun, C.G.J., Wynn, M.T., Röglinger, M.: Enhancing event log quality: Detecting and quantifying timestamp imperfections. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management. pp. 309–326. Springer International Publishing, Cham (2020)
18. H. Gyldenkaerne, C., From, G., Mønsted, T., Simonsen, J.: Pd and the challenge of ai in health-care. In: Proceedings of the 16th Participatory Design Conference 2020-Participation (s) Otherwise-Volume 2. pp. 26–29 (2020)
19. Hildebrandt, T.T., et. al.: EcoKnow: Engineering Effective, Co-Created and Compliant Adaptive Case Management Systems for Knowledge Workers, p. 155–164. Association for Computing Machinery, New York, NY, USA (2020), `https://doi.org/10.1145/3379177.3388908`
20. Holten Møller, N., Shklovski, I., Hildebrandt, T.T.: Shifting concepts of value: Designing algorithmic decision-support systems for public services. In: Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society. NordiCHI '20, Association for Computing Machinery, New York, NY, USA (2020), `https://doi.org/10.1145/3419249.3420149`
21. Jans, M., Soffer, P.: From relational database to event log: Decisions with quality impact. In: Business Process Management Workshops. Lecture Notes in Business Information Processing, vol. 308, pp. 588–599. Springer (2017)
22. Jung, J.Y., Steinberger, T., So, C.: Domain experts as owners of data: towards sustainable data science (2022)
23. Kensing, F., Simonsen, J., Bodker, K.: Must: A method for participatory design. Human-computer interaction **13**(2), 167–198 (1998)
24. Leopold, H., van der Aa, H., Pittke, F., Raffel, M., Mendling, J., Reijers, H.A.: Searching textual and model-based process descriptions based on a unified data format. Softw. Syst. Model. **18**(2), 1179–1194 (2019)

25. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: CAiSE Forum. Lecture Notes in Business Information Processing, vol. 317, pp. 182–199. Springer (2018)
26. Lux, M., Rinderle-Ma, S.: Problems and challenges when implementing a best practice approach for process mining in a tourist information system. In: BPM (Industry Track). CEUR Workshop Proceedings, vol. 1985, pp. 1–12. CEUR-WS.org (2017)
27. Mannhardt, F.: Responsible process mining. Process Mining Handbook. LNBIP **448**, 373–401 (2022)
28. Meystre, S.M., Lovis, C., Bürkle, T., Tognola, G., Budrionis, A., Lehmann, C.U.: Clinical data reuse or secondary use: current status and potential future progress. Yearbook of medical informatics **26**(01), 38–52 (2017)
29. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Connecting databases with process mining: a meta model and toolset. Softw. Syst. Model. **18**(2), 1209–1247 (2019)
30. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Case notion discovery and recommendation: automated event log building on databases. Knowl. Inf. Syst. **62**(7), 2539–2575 (2020)
31. Petersen, A.C.M., Christensen, L.R., Harper, R., Hildebrandt, T.: "we would never write that down": Classifications of unemployed and data challenges for ai. Proc. ACM Hum.-Comput. Interact. **5**(CSCW1) (apr 2021), https://doi.org/10.1145/3449176
32. Robertson, T., Simonsen, J.: Participatory design: an introduction. In: Routledge international handbook of participatory design, pp. 1–17. Routledge (2012)
33. Sànchez-Ferreres, J., van der Aa, H., Carmona, J., Padró, L.: Aligning textual and model-based process descriptions. Data Knowl. Eng. **118**, 25–40 (2018)
34. Schrodt, P.A.: The statistical characteristics of event data. International Interactions **20**(1-2), 35–53 (1994)
35. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Cortado—an interactive tool for data-driven process discovery and modeling. In: Buchs, D., Carmona, J. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 465–475. Springer International Publishing, Cham (2021)
36. Slaats, T.: Declarative and hybrid process discovery: Recent advances and open challenges. J. Data Semant. **9**(1), 3–20 (2020). https://doi.org/10.1007/s13740-020-00112-9
37. Smylie, J., Firestone, M.: Back to the basics: Identifying and addressing underlying challenges in achieving high quality and relevant health statistics for indigenous populations in canada. Statistical Journal of the IAOS **31**(1), 67–87 (2015)
38. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)
39. Team, R.: Responsible data science, https://redasci.org/
40. van Zelst, S., Mannhardt, F., de Leoni, M.: Event abstraction in process mining: literature review and taxonomy, vol. 6, pp. 719–736. Springer New York, New York, NY (2021). https://doi.org/10.1007/s41066-020-00226-2
41. Zhang, A.X., Muller, M., Wang, D.: How do data science workers collaborate? roles, workflows, and tools. Proceedings of the ACM on Human-Computer Interaction **4**(CSCW1), 1–23 (2020)

Chapter 8

Understandable declarative process models

8.1 Improving Simplicity by Discovering Nested Groups in Declarative Models

Remark 8.1. *The work will be published as: Improving Simplicity by Discovering Nested Groups in Declarative Models Cosma VP, Christfort AKF, Hildebrandt TT, Lu X, Reijers HA, Slaats T. International Conference on Advanced Information Systems Engineering (CAISE) 2024*[1].

[1] `https://cyprusconferences.org/caise2024/`

Improving Simplicity by Discovering Nested Groups in Declarative Models

Vlad Paul Cosma[1], Axel Kjeld Fjelrad Christfort[1], Thomas T. Hildebrandt[2], Xixi Lu[2], Hajo A. Reijers[2], and Tijs Slaats[1]

[1] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark {vco, axel, hilde, slaats}@di.ku.dk
[2] Information and Computing Sciences, Utrecht University, Utrecht, Netherlands {x.lu, h.a.reijers}@uu.nl

Abstract. Discovering simple, understandable and yet accurate process models is a well-known issue for models mined from real-life event logs. In this paper, we consider algorithms for automatically computing nested groups of activities in declarative process languages, concretely Dynamic Condition Response (DCR) Graphs, to reduce complexity while preserving accuracy. The DCR Graphs notation is, on the one hand, supported by the very accurate DisCoveR process mining algorithm, and on the other hand, by mature design and execution tools used in industrial processes and enterprise information management systems. We evaluate our approach by applying the DisCoveR miner to a large benchmark of real-life and synthetic event logs, measuring the size, density, separability, and constraint variability of mined models with and without grouping of activities. In earlier work, these measures have been shown to have a significant effect on the intrinsic cognitive load for users of declarative models, in particular DCR Graphs. We also evaluate the effect of prioritizing in particular the grouping of activities that model mutual exclusive choices. Our evaluation confirms that grouping of activities in general lowers the complexity on 3 of the 4 measures, while prioritizing choices in some cases makes the improvement slightly smaller.

Keywords: Process Discovery · Declarative · Simplicity · Choices · Nested Groups · DCR Graphs

1 Introduction

Process Discovery has been one of the most prominent tasks in Process Mining, allowing to automatically reconstruct the process from event data. Process discovery has been applied in many domains such as healthcare and financing and is the foundation for enhanced process analysis. Most existing discovery techniques focus on the imperative approach, tailoring towards structured, simple processes. When the process is more flexible and complex, these techniques tend to discover spaghetti models. By explicitly modelling the constraints between

activities, declarative process languages such as Declare [1] and Dynamic Condition Response (DCR) Graphs [2–4] have been shown to represent processes with a high degree of flexibility more succinctly than imperative process languages. The DisCoveR [5] process mining algorithm produces DCR Graphs, and through experiments on public logs and in particular, winning the 2021 and 2023 Process Discovery Contest (PDC), has been shown to produce highly accurate models. However, some more restrictive constructs, such as mutual exclusive choices, require many constraints to be modelled, which may lead to models that are difficult for users to comprehend.

In the present paper, we address this shortcoming by utilizing the extension of DCR Graphs with nested groups of activities [6] to automatically reduce the number of visual elements in the model. A constraint for a group of activities is in essence just a short-hand for having the constraint for all activities in the group. In particular, the short-hand maintains the semantics of the model and, as a result, its accuracy. By being just a syntactic short-hand, nested groups of activities are different from hierarchies of sub-processes, which typically introduce a new notion of state for a sub-process. As an example, consider the commonly encountered choice pattern, i.e. a group of activities of which only one can be executed. This is modelled in a DCR graph by a so-called exclusion constraint between any pair of activities in the group in each direction, also including a constraint from any activity to itself. That is, a choice between a group of N activities requires N^2 constraints. In Fig. 1 we see the quadratic relation reduction on the DCR Graph mined from the Business Process Intelligence Challenge 2017 Offers (BPIC17o) event log [7]. The use of a group allows to replace N^2 exclusions constraints between the mutually exclusive activities `O_Refused`, `O_Cancelled` and `O_Accepted` by a single self-exclusion on the choice group `Choice1`.

Andaloussi et al. [8] have shown that such a large reduction in the number of relations has a significant impact on the comprehensibility of DCR Graphs in terms of perceived difficulty, answer correctness, and answer time. In particular they proposed 4 simplicity measures for DCR Graphs capturing size, density, separability, and constraint variability and showed that these were accurate predictors for the intrinsic cognitive load and therefore understandability of models. We evaluate our approach on these measures by mining models for a large set of public event logs and show that we achieve on average a 42% reduction in size, a 65% reduction in density, a 5% increase in separability, in exchange for a 22% increase in constraint variability. While the first three of these results are linked to an increase in simplicity, the latter may be interpreted as a decrease in simplicity. However, as we will discuss in more detail in Section 5, the increase in constraint variability is a direct consequence of the reduction in size and we posit that together these results indicate a significant increase in the expected understandability of the mined models.

Our algorithm and experiments are available as an open source python implementation[3] extending the pm4py library [9], where we provide the original

[3] `https://github.com/paul-cvp/pm4py-dcr`

DisCoveR miner together with DCR execution semantics and model import and export capabilities which are compatible with the DCRSolutions design tool[4].

As such, our contributions include (1) two novel algorithms for discovering nested groups of activities in DCR graphs in Sec. 3, (2) a thorough evaluation of these algorithms on an exhaustive set of public event logs in Sec. 4, and (3) the first application of the simplicity measures proposed in [8] to real models leading to important insights regarding the interplay between these measures discussed in Sec. 5, and in particular (4) an improved DisCoveR miner that provides simpler models without sacrificing accuracy. Finally, it is worth noting, that the use of nested groups to reduce the number of edges in a graph can be applied to any graph model, such as e.g. Declare and Petri Net models.

2 Preliminaries

2.1 Dynamic Condition Response Graphs with Nested Groups

Below we give the definition of Dynamic Condition Response (DCR) graphs with nested groups of activities as introduced in [3, 4, 6] but simplified to the graphs discovered by the DisCoveR process miner, that is, graphs where each activity is represented by a unique node in the graph and we only have the original four relations between nodes as introduced in [2].

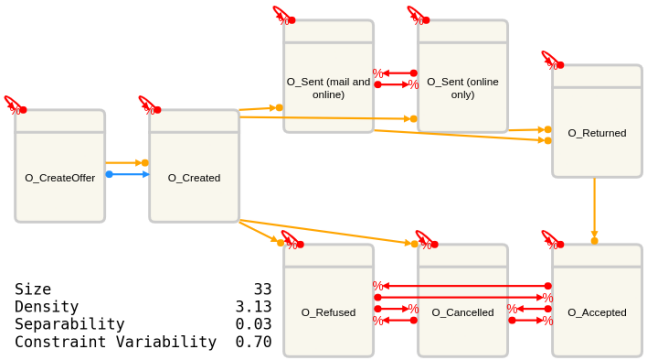
Definition 1. *A* DCR Graph G with nested groups of activities *is given by a tuple* $(A, M, R, A_G, \rhd)$ *where*

(i) $AG = A \uplus A_G$ *is a finite set of* activities A *and* activity groups A_G,
(ii) $M = (Ex, Re, In) \in \mathcal{P}(A) \times \mathcal{P}(A) \times \mathcal{P}(A)$ *is the* marking,
(iii) $R = \{\rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\!+, \rightarrow\!\!\%\}$ *are the four basic relations, i.e.*
(iv) $\rightarrow\!\bullet \subseteq AG \times AG$, *is the* condition *relation,*
(v) $\bullet\!\rightarrow \subseteq AG \times AG$, *is the* response *relation,*
(vi) $\rightarrow\!\!+, \rightarrow\!\!\% \subseteq AG \times AG$ *are* include *and* exclude *relations respectively,*
(vii) $\rhd\colon AG \rightharpoonup A_G$ *is a* grouping *function,*

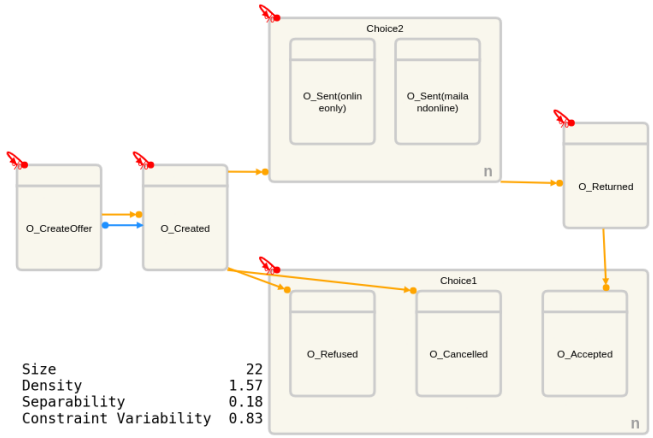
We write $>$ *for* $\rhd^+$ *(the transitive closure of* $\rhd$*) and require that it is irreflexive. We write* $\geq$ *for reflexive closure of* $>$ *and* $\leq$ *for the inverse of* $\geq$.

Compared to the original DCR Graphs [2], DCR Graphs with nested groups allow also groups as nodes of the graph, and the grouping function $\rhd$ defines a partial order $\geq$ on groups and activities, determining which group an activity or group belongs to (if any). As already explained in the introduction, the idea of DCR Graphs with nested groups is, that a relation from/to a group is a concise way of expressing a relation from/to all members of the group. As shown in [6] and formalized in Def. 2 below, a DCR Graph G with nested groups of activities can be mapped to a semantically equivalent standard, flat DCR Graph $G^\flat$, by replacing a relation from/to a group by a relation from/to all the members of the group and removing the group.

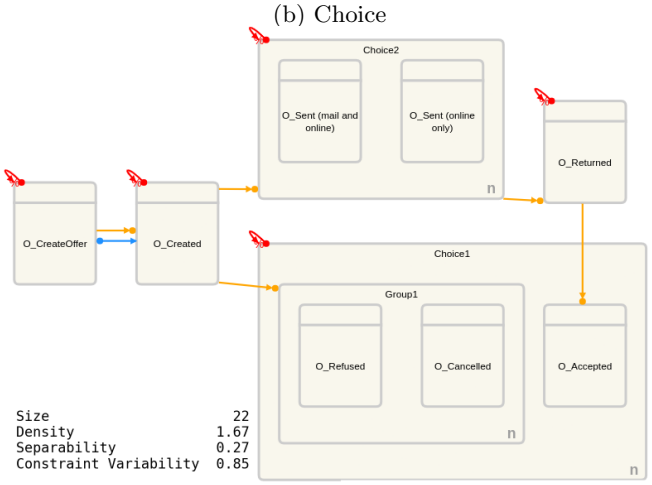
[4] Freely available for academic use at `https://dcrsolutions.net`



(a) Flat



(b) Choice



(c) Choice+Group (same as Group)

Fig. 1: DCR Graphs of the BPIC17 Offer log.

Definition 2. *Let $G = (A, M, R, A_G, \triangleright)$ be a DCR graph with nested groups of activities. We define the equivalent standard DCR Graph as $G^\flat = (A, M, R_{\leq\geq})$, where $R_{\leq\geq} = \{\phi^\flat \mid \phi \in R \text{ and } \phi^\flat = \leq \phi \geq \cap (A \times A)\}$.*

The execution semantics for a DCR Graph G with nested activity groups is then defined in terms of the standard flat DCR Graph $G^\flat$ as it is done in [6]. That is, an activity is enabled in G if it is enabled in $G^\flat$, and the marking resulting from executing the activity in G is the marking resulting from executing the activity in $G^\flat$. Since the semantics of DCR Graphs is not the key point of the present paper, we refer the interested reader to e.g. [3, 4]. Intuitively, an activity a is enabled if every condition $a' \rightarrow\!\!\bullet^\flat\ a$ that is included in the present marking is also previously executed, i.e. $a' \in In \cap Ex$. The effect of executing an enabled activity is to add the activity to the set of executed activities in the marking, remove it from the set Re of pending activities, and include/exclude/make pending the activities that the activity has include/exclude/response relations to.

2.2 Declarative complexity metrics

The work in [8] introduces 4 complexity metrics for declarative process models: size, density, separability and constraint variability. The authors also provide a qualitative study of the metrics which correlates the change in metrics with a change in the users' cognitive load.

Definition 3. *Let $G = (A, M, R, A_G, \triangleright)$ be a nested DCR Graph with $AG = A \uplus A_G$. We define the following metrics:*

***Size (S):** Defined as the sum of activities, groups and relations:*

$$S(G) = |AG| + |R| \tag{1}$$

***Density (D):** Defined as the maximum number of relations over the number of activities in weakly connected components of the graph. Let $Comp(G) = \{c_1, \ldots, c_n\}$ be the set of weakly connected components of graph G, then for a given $c \in Comp(G)$ we denote the number of activities/groups in the component c as AG_c and the number of relations as R_c.*

$$D(G) = \max_{c \in Comp(G)} \left| \frac{R_c}{AG_c} \right| \tag{2}$$

***Separability (Sep):** Measures the number of weakly connected components over the number of activities, groups and relations in the model:*

$$Sep(G) = \frac{|Comp(G)|}{|AG| + |R|} \tag{3}$$

***Constraint Variability (CV):** Related to Shannon entropy, it is defined as the maximum entropy over different relation types in the components of the model. Let R_c be the set of different types of relations within a component c and R_c^r be the relation of type $r \in R$ in component c. Then the relative frequency is:*

$$p(c,r) = \begin{cases} \frac{|R_r^c|}{|R_c|} & \textit{if } |R_c| > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

Now we can define constraint (or relation) variability as:

$$CV(G) = \max_{c \in \{c' | c' \in Comp(G) \wedge |R_{c'}| > 0\}} \left\{ - \sum_{r \in R_c} p(c,r) \cdot \log_{|R|} p(c,r) \right\} \tag{5}$$

.

An increase in size, density and constraint variability relates to an increase in the users' cognitive load. Separability is inversely correlated, a decrease in separability relates to an increase in cognitive load.

Example: Based on the BPIC17 Offer log [7] we consider the mined flat DCR Graph and the semantically equivalent DCR Graphs with choices and nested groups in Fig. 1. The nested graphs have a significant reduction in the number of relations while remaining behaviourally equivalent. Observe that the `Choice` graph in Fig. 1b can still be improved by reducing the two condition relations to `O_Refused` and `O_Cancelled`. In Fig. 1c the greatest relation reduction comes at the expense of adding 3 activity groups `Choice1`, `Choice2` and `Group1` to the graph. The declarative complexity metrics on the flat and fully grouped process model show this reduction. The flat model 1a has a size of 33, density of 3.13, separability of 0.03 and a constraint variability of 0.70. The grouped model 1c has improvements on size 22 (33%), density 1.67 (46%), and separability 0.27 (25%), and a worsening for constraint variability 0.85 (-21%). Observe that between Fig. 1b and 1c size stays the same as the number of removed relations is the same as the added nested groups, but density will change because it is a ratio of the two.

3 Discovering Nested Groups in DCR Graphs

Starting from a flat DCR Graph we provide a `Choice` algorithm to group maximal subsets of activities that are all connected by exclusion relations and replacing the exclusion relations by a single self-exclusion for the group, and a greedy algorithm `Group` for groups of activities that all share a relation and replace the individual relations by a single relation for the group. Both algorithms are easily seen from the definition to preserve the trace semantics.

3.1 Choice

Choices in a DCR Graph are represented as a set of activities that, when executed, mutually exclude each other and themselves. We now define the mutual exclusion sub-graph $G_{\#}$ as we are only interested in self excluding activities and the exclusion relations between them.

Algorithm 1 choice($G_{\#}$)

```
1: cliques = enumerate_all_cliques(G_#)
2: cliques = sort(cliques, key = length, reverse = True)        ▷ Largest clique first
3: used = set.empty()
4: ▷ = map.empty()
5: A_G = set.empty()
6: for clique ∈ cliques do
7:     if clique ∪ used = ∅ and length(clique) > 1 then        ▷ Check that the activities inside the clique have not been already used
8:         a_g = id()
9:         A_G = A_G ∪ {a_g}
10:        ▷ (a_g) = clique                                    ▷ Update the nesting map
11:        used = used ∪ clique                                ▷ Update the used activities set
12:    end if
13: end for
14: return (A_G, ▷)          ▷ Return the set of choice group activities, and the nesting map
```

Definition 4. *Let $G = (A, M, R)$ be a DCR Graph. Let $G_{\#} = (A_{\#}, \#)$ be the mutual exclusion sub-graph of G where: $A_{\#} = \{a | (a, a) \in \not\rightarrow\!\!\%\}$, and*

$$\# = \{(a, a') | a \in A_{\#}, a' \in A_{\#}, (a, a') \in \not\rightarrow\!\!\%, (a', a) \in \not\rightarrow\!\!\%\}.$$

Note that the mutual exclusion relation # is an undirected edge. Now we define Algorithm 1 that takes the mutual exclusion sub-graph $G_{\#}$ and finds the optimal grouping of connected activities. The equivalent problem from graph theory is that of finding all cliques with a size greater than 1. In the worst case, time complexity is known to be exponential in the number of nodes.

To update the flat DCR Graph we first add the resulting cliques as choice activity groups. Then we remove all exclusions $\not\rightarrow\!\!\%$ between the grouped activities together with their self excludes, add a self-exclude relation to the activity group and move any shared relations by all individual activities to point in/out of the choice group. For each choice group with activities `clique` added we thus reduce the number of visual exclusions by $|\texttt{clique}|^2 - 1$. For each further relation the grouped activities share one get further reduction of $|\texttt{clique}| - 1$ relations. An example can be seen in Fig. 1b, where the two activities grouped inside Choice2 leads to the removal of $2^2 - 1 = 3$ exclusion relations and two condition relations.

3.2 Group

We here explore another approach, finding all groups that reduce the relations of the graph with regard to a metric. The metric scores candidate groups higher when the relation reduction is also higher.

In order to do this we propose the efficient greedy algorithm as seen in Algorithm 4. The algorithm works on an encoded DCR graph, which can be obtained from a flat DCR graph with Algorithm 2. This encoding simply represents a graph as the sets of all incoming and outgoing relations for each activity, allowing for easy computation of shared relations by intersection. It does, however, split each relation into each of its start and end points, and as such it requires

Algorithm 2 encode(G)

1: $(A, M, R) = G$
2: `enc = map.empty()`
3: **for** $a \in A$ **do**
4: `enc`$(a) = \{b, out, r.type) | \rightarrow \in R, (a, b) = r \in \rightarrow\}$
5: `enc`$(a) =$ `enc`$(a) \cup \{(b, in, r.type) | \rightarrow \in R, (b, a) = r \in \rightarrow\}$
6: **end for**
7: **return** `enc`

Algorithm 3 decode(`enc`)

1: $AG =$ `enc.keys`
2: $\rightarrow\bullet = \{ (b, ag) | ag \in AG.((b, in, condition) \in$ `enc`$(ag) \}$
3: $\bullet\rightarrow = \{ (ag, b) | ag \in AG.((b, out, response) \in$ `enc`$(ag) \}$
4: $\rightarrow\% = \{ (ag, b) | ag \in AG.((b, out, exclude) \in$ `enc`$(ag) \}$
5: $\rightarrow+ = \{ (ag, b) | ag \in AG.((b, out, include) \in$ `enc`$(ag) \}$
6: **return** $(AG, (AG, \emptyset, \emptyset), (\rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%))$

slight bookkeeping to keep consistent under alteration as can be seen in lines 26-29 of the algorithm.

Algorithm 4 works by computing the intersection of the encoded relations for each pair of activities as the candidate groups. We then choose the best group with regards to the size metric, *e.g.* number of relations removed, and add it to the encoding by moving all the shared relations onto the group from the individual activities and updating $\triangleright$. For each found group, we apply this method recursively to find further nested groups. The method is then repeated on the remaining un-grouped activities until no further groups can be found that improve the metric.

Finally, we decode the resulting encoding with Algorithm 3, which when joined with the found groups A_G and the group function $\triangleright$, yield exactly a DCR Graph with nested groups as per Definition 1.

4 Evaluation

Data Sets: To evaluate the `Choice` and `Group` algorithms, we used the same real-world event logs from [10] on which imperative miners have been evaluated. In addition, we used BPIC 2017 [11] and the Offer subset (BPIC17o) [7], BPIC 2019 [12] and Dreyers [13] logs. An overview of the results for these logs is provided in Table 2. To show that the reduction works also on noisy logs, we extended our evaluation to synthetically generated logs from the 2019-2023 PDC [5]. From the PDC data sets, we only use the training logs. In total we use 21 publicly available data sets, 16 real world logs and 5 synthetic ones. Note that the 5 PDC data sets are collections of logs, we therefore average the metrics across each collection such that they are equally weighted in the overall aggregated *results.*

[5] `https://www.tf-pm.org/competitions-awards/discovery-contest`

Algorithm 4 group(`enc`, $A =$ `enc.keys`)

```
1: ▷= map.empty()
2: A_G = ∅
3: while True do
4:     cands = map.empty()
5:     for a, t ∈ A do                                  ▷ Find all candidate groupings by intersection
6:         →_s= enc(a) ∩ enc(t)
7:         cands(→_s) = cands(→_s) ∪ {a, t}
8:     end for
9:                                                      ▷ Choose the best one by the given score
10:    →_b, best_score = max(cands.keys, metric(→_s))
11:    if best_score = 0 then                           ▷ Return if no improvement can be made
12:        break
13:    end if
14:
15:    a_g = id()
16:    A_G = A_G ∪ {a_g}
17:
18:    if nested then                                   ▷ if this grouping is nested
19:        Update(▷)                                    ▷ then ▷ needs to point to the parent grouping
20:    end if                                           ▷ this is left as an implementation detail
21:
22:    enc(a_g) =→_b                                    ▷ add shared relations to the grouping
23:    for a ∈ cands(→_b) do
24:        ▷ (a) = a_g
25:        enc(a) = enc(a)\ →_b                         ▷ remove shared relations from each grouped activity
26:        for (b, direction, type) ∈→_b do             ▷ redirect other ends of shared relations
27:            enc(b) = enc(b) \ {(a, flip(direction), type)}
28:            enc(b) = enc(b) ∪ {(a_g, flip(direction), type)}
29:        end for
30:    end for
31:
32:    (enc', A'_G, ▷') = nest(enc, cands(→_b))         ▷ Recursively find nested groupings
33:    A_G = A_G ∪ A'_G
34:    enc = enc'
35:    ▷=▷ ∘ ▷'
36:    A = A \ cands(→_b)                               ▷ Remove already grouped events from further consideration
37: end while
38: return (enc, A_G, ▷)
```

Setup: Starting from the event log we use DisCoveR to mine a perfectly fitting flat DCR Graph. We then derive three models from the flat one: (1) a `Choice` based grouping as defined in Section 3.1, (2) `Group`, the greedy algorithm from Section 3.2, and (3) a `Choice+Group` approach where we first find `Choices` and then further `Group` the model by the greedy relation reduction. By Definition 2 the derived models are also perfectly fitting DCR Graphs.

Metrics: We calculate the Size, Density, Separability and Constraint Variability to evaluate the process models derived from the mined flat DCR Graph. All 3 algorithms achieve the relation reduction at the cost of adding activity groups.

Results: The results for the real-life logs are also shown in Table 2 and our full evaluation results are available online[6]. In Fig. 2 we show the percentage changes compared to the flat DCR Graph as box plots for each metric and algorithm combination. The flat DCR Graph is shown as a dashed red line at 0%.

[6] `https://github.com/paul-cvp/all-complexity-results`

Fig. 2: Boxplot of metrics as percentage improvements (negative percentages indicate deterioration)

Overall the `Group` algorithm performs best, achieving a 42% median improvement on Size, 65% improvement on Density, 5% improvement on Separability, and a −22% worsening of Constraint Variability. Note that the scale on the constraint variability box plot is between 0% and −100%. The improvement trend in size, density and separability correlates with a decrease in the users' cognitive load [8]. Constraint variability is the only metric where our approaches perform worse. This is an expected outcome as the ratio between the total number of relations and the individual relations tends to even out.

5 Discussion

5.1 Interpretation

Comparison of algorithms. The intuition behind the relation reduction can be seen in Table 1. We observe visually that the models have less overlapping relations, or relations that have to cross over other activities. By understanding the concept of groups, users looking at the discovered models will spend less time following relations back and forth between the connected activity pairs.

As can be seen in our results, the general `Group` algorithm has a considerably higher impact on all simplicity metrics than the `Choice` algorithm. `Group` also runs on process models that have been manually created, which extends its applicability beyond refining mined models. Combining the two algorithms as `Choice+Group` has a similar effect on the metrics, albeit slightly lower on average. A detailed inspection of the results in Table 1 shows that the difference is not

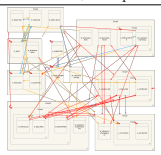
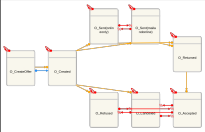
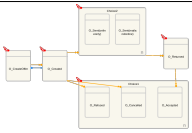
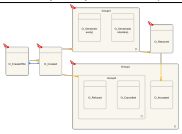
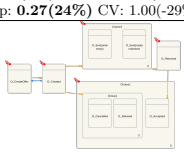
Datasets	Flat	Choice	Group	Choice+Group
BPIC12	 S: 156(0%) D: 5.50(0%) Sep: 0.01(0%) CV: 0.59(0%)	 S: 134(14%) D: 4.15(24%) Sep: 0.01(0%) CV: 0.66(-10%)	 S: **91(41%)** D: 1.96(64%) Sep: **0.05(4%)** CV: 0.78(-31%)	 S: 101(35%) D: **1.91(65%)** Sep: 0.04(3%) CV: 0.77(-29%)
RTFMP	 S: 33(0%) D: 2.00(0%) Sep: 0.03(0%) CV: 0.77(0%)	 S: 33(0%) D: 2.00(0%) Sep: 0.03(0%) CV: 0.77(0%)	 S: **30(9%)** D: **1.40(30%)** Sep: **0.27(24%)** CV: 1.00(-29%)	 S: **30(9%)** D: **1.40(30%)** Sep: **0.27(24%)** CV: 1.00(-29%)
BPIC17o	 S: 33(0%) D: 3.13(0%) Sep: 0.03(0%) CV: 0.70(0%)	 S: **22(33%)** D: **1.57(49%)** Sep: 0.18(15%) CV: 0.83(-17%)	 S: **22(33%)** D: 1.67(46%) Sep: **0.27(25%)** CV: 0.85(-21%)	 S: **22(33%)** D: 1.67(46%) Sep: **0.27(25%)** CV: 0.85(-21%)
SEPSIS	 S: 107(0%) D: 6.00(0%) Sep: 0.02(0%) CV: 0.90(0%)	 S: 76(28%) D: 3.63(39%) Sep: 0.03(1%) CV: 1.00(-10%)	 S: 54(49%) D: 2.07(46%) Sep: 0.15(13%) CV: 0.96(-6%)	 S: **51(65%)** D: **1.47(75%)** Sep: **0.18(16%)** CV: 0.97(-7%)

Table 1: Results for a subset of the logs (best results highlighted in **bold**)

uniform. `Group` performs best on logs such as BPIC12 because it is not restricted by a prior choice grouping. It also performs best on logs, such as RTFMP, where there are no discovered choices. For others, such as BPIC17-Offer, all derived models from `Choice`, `Group` and `Choice+Group` perform equally well. Finally for SEPSIS `Choice+Group` is best. An important difference between these two algorithms, which is not expressed by the metrics, is the fact that choice groups have a clear semantics and are easy to recognize for modellers with a basic understanding of DCR Graphs. Therefore, we conjecture that prioritizing the finding of choice groups has a positive impact on the understandability.

Constraint Variability and multi-perspective measures: On all evaluated event logs we see that an improvement on the first three measures implies a worsening in constraint variability. A possible explanation for this is that DisCoveR tends to find an unbalanced set of relations where conditions and exclusions are more common. This reduces the constraint variability of the flat model. Since the `Choice` algorithm reduces exclusions, and the `Group` algorithm prioritizes reducing the most relations, exclusions and conditions are more likely to be grouped than the other relations. As a result, the constraint variability of the grouped model increases. We can see this in Fig. 1: consider the response relation $\bullet\!\!\rightarrow$

Log name	Algorithm	|AG|	|R|	S	D	Sep	CV
BPIC12	Flat	24	132	156	5.50	0.01	0.59
	Choice	26	108	134	4.15	0.01	0.66
	Group	32	59	91	1.96	0.05	0.78
	Choice+Group	36	65	101	1.91	0.04	0.77
BPIC13_cp	Flat	4	4	8	1.00	0.13	0.81
	Choice	4	4	8	1.00	0.13	0.81
	Group	5	3	8	0.50	0.38	0.00
	Choice+Group	5	3	8	0.50	0.38	0.00
BPIC13_i	Flat	4	6	10	1.50	0.10	1.00
	Choice	4	6	10	1.50	0.10	1.00
	Group	6	3	9	1.00	0.44	1.00
	Choice+Group	6	3	9	1.00	0.44	1.00
BPIC14_f	Flat	9	27	36	3.00	0.03	1.00
	Choice	9	27	36	3.00	0.03	1.00
	Group	11	8	19	1.60	0.37	1.00
	Choice+Group	11	8	19	1.60	0.37	1.00
BPIC15_1f	Flat	70	475	545	6.79	0.00	0.82
	Choice	81	437	518	5.88	0.02	0.86
	Group	112	237	349	2.24	0.03	1.00
	Choice+Group	107	309	416	3.20	0.04	1.00
BPIC15_2f	Flat	82	902	984	11.00	0.00	0.64
	Choice	90	875	965	10.24	0.01	0.65
	Group	133	379	512	2.95	0.02	0.72
	Choice+Group	148	429	577	3.19	0.03	1.00
BPIC15_3f	Flat	62	699	761	11.27	0.00	0.50
	Choice	71	659	730	9.65	0.01	0.51
	Group	97	202	299	2.27	0.05	0.92
	Choice+Group	108	390	498	4.18	0.03	0.52
BPIC15_4f	Flat	65	522	587	8.03	0.00	0.82
	Choice	70	500	570	7.32	0.01	0.83
	Group	103	202	305	2.12	0.05	1.00
	Choice+Group	110	245	355	2.52	0.05	1.00
BPIC15_5f	Flat	74	573	647	7.74	0.00	0.79
	Choice	82	546	628	7.11	0.01	0.82
	Group	114	240	354	2.19	0.03	0.98
	Choice+Group	117	347	464	3.19	0.03	0.95
BPIC17	Flat	26	119	145	4.58	0.01	0.74
	Choice	27	108	135	4.00	0.01	0.76
	Group	40	67	107	1.78	0.03	0.83
	Choice+Group	40	70	110	1.79	0.02	0.83
BPIC17-Offer	Flat	8	25	33	3.13	0.03	0.71
	Choice	10	12	22	1.57	0.18	0.83
	Group	11	11	22	1.67	0.27	0.86
	Choice+Group	11	11	22	1.67	0.27	0.86
BPIC17_f	Flat	18	64	82	3.56	0.01	0.86
	Choice	19	61	80	3.33	0.03	0.86
	Group	25	49	74	2.00	0.03	0.88
	Choice+Group	25	48	73	1.96	0.03	0.86
BPIC19	Flat	42	599	641	14.26	0.00	0.32
	Choice	45	504	549	11.20	0.00	0.36
	Group	69	218	287	3.25	0.01	0.48
	Choice+Group	76	215	291	2.95	0.01	0.49
Dreyers	Flat	33	268	301	8.12	0.00	0.53
	Choice	33	268	301	8.12	0.00	0.53
	Group	52	109	161	2.38	0.04	0.67
	Choice+Group	65	112	177	1.95	0.05	0.66
RTFMP	Flat	11	22	33	2.00	0.03	0.77
	Choice	11	22	33	2.00	0.03	0.77
	Group	14	16	30	1.40	0.27	1.00
	Choice+Group	14	16	30	1.40	0.27	1.00
SEPSIS	Flat	16	91	107	6.00	0.02	0.91
	Choice	17	59	76	3.63	0.03	1.00
	Group	22	32	54	2.07	0.15	0.96
	Choice+Group	24	27	51	1.47	0.18	0.98

Table 2: Algorithm and metrics from real world event logs

between `O_CreateOffer` and `O_Created`, relative frequency $p(G, \bullet\!\!\rightarrow)$ is 1/25 in the flat graph and 1/11 in the grouped one.

This insight into the correlated increase in constraint variability strengthens the claim from the original study that a combination of measures is necessary "to provide a multi-perspective view of users' cognitive load when engaging with declarative process models." [8]. It remains an open question whether or not it is possible to improve on all 4 metrics simultaneously via algorithmic means.

5.2 Metric validity

Our experimental results show an improvement in 3 out of the 4 metrics that were selected. This is a reassuring result, which provides positive support for the effectiveness of our proposal.

The question remains on whether the metrics themselves provide a valid and decisive assessment for our particular proposal. The original metrics are specifically tuned to DCR Graphs, but do not take groups as a visual element into account. It cannot be entirely ruled out, therefore, that the groups themselves introduce additional cognitive load, which is not covered by the metrics.

What is reassuring is that previous studies have found that hierarchy does not introduce additional cognitive load in declarative processes [14, 15]; neither do groupings for BPMN models [16]. Therefore, one possible use of our findings is to use groups to create hierarchies in DCR Graphs and hide relations within

the groups, only making them visible on-demand. It seems very plausible that this will reduce any cognitive load, which is worthwhile to investigate further.

6 Related work

The main work done within declarative process discovery has been on Declare models, DCR graphs, and Log Skeletons [17]. Declare Miner [18] and MINERful [19] discover Declare models. All existing DCR miners [5, 20, 21] discover flat DCR Graphs. Our work is built on the DisCoveR miner as it outputs the most accurate DCR Graphs [5]. Zugal et al. [14] and Haisjackl et al. [15] found that introducing hierarchical sub-processes in declarative process models does not lead to any significant change in cognitive load. Turetken et al. [16] showed that groupings have no measurable effect on cognitive load for BPMN models. However, unlike our approach, the use of BPMN groupings in [16] does not decrease the number of visual elements in a model. Smirnov et al. [22] systematically catalog business process model abstraction techniques and show their value through use cases.

Discovering hierarchies or sub-processes is well studied for imperative process model notations [23–27]. Here the use of the word "hierarchy" denotes different levels of event abstraction, which in contrast to our use of groups changes the semantics (and accuracy) of the model. In the context of fuzzy mining, a hierarchy is based on metrics from the event log and relates to how a directly follows graph can be simplified by removing less frequent nodes and edges [28]. In the context of multi-level event logs, mining hierarchical process models [29] means discovering hierarchies in logs where each part can be mined with its own miner resulting in a combination of several process model notations, including Declare. FlexHMiner [30] is a miner for hierarchical process models that discovers subprocesses based on three different methods, using domain knowledge, random clustering, and a flat tree, with the domain knowledge approach being on average the highest-scoring one. Prime Miner [31] was the only other work that explicitly mines choices. Several complexity metrics have been used for imperative models as a proxy for simplicity [10, 32, 33]. Augusto et al. [33] review complexity metrics on a similarly large set of event logs, and indeed find a correlation between the metrics and the quality of process models mined with imperative miners. They investigate how pre-processing event logs before mining can improve the results. A similar approach is taken for Declare in [34].

7 Conclusion

We provided the first evaluation of automatic grouping of activities in order to reduce the complexity of mined declarative models and used the complexity metrics proposed in [8] on a set of 16 real-life and 5 synthetic logs. The result is a likely significant reduction in cognitive load for model users. For future work, a user-centered evaluation study seems desirable to corroborate and expand our current, metric-based evaluation. As mentioned in our discussion, the

inclusion of groupings may introduce cognitive load not taken into account by the used metrics. By involving users it can be tested whether this effect materializes or not. Additionally, it would be worth investigating how the reduction of cognitive load through groupings ties to the expertise level of users, i.e., beginners/intermediate/experts, which may have important ramifications for tailoring tool support and instruction materials. Finally, it seems worthwhile to evaluate the use of activity groups for Declare models.

References

1. M. Pesic, H. Schonenberg, and W. M. Van der Aalst, "Declare: Full support for loosely-structured processes," in *11th IEEE international enterprise distributed object computing conference (EDOC 2007)*. IEEE, pp. 287–287.
2. T. T. Hildebrandt and R. R. Mukkamala, "Declarative event-based workflow as distributed dynamic condition response graphs," in *PLACES*, 2010, pp. 59–73.
3. R. R. Mukkamala, "A formal model for declarative workflows: Dynamic condition response graphs," Ph.D. dissertation, IT University of Copenhagen, 2012.
4. T. Slaats, "Flexible process notations for cross-organizational case management systems," Ph.D. dissertation, IT University of Copenhagen, 2015.
5. C. O. Back, T. Slaats, T. T. Hildebrandt, and M. Marquard, "Discover: accurate and efficient discovery of declarative process models," in *International Journal on Software Tools for Technology Transfer*, 2021.
6. T. Hildebrandt, R. R. Mukkamala, and T. Slaats, "Nested dynamic condition response graphs," in *International conference on fundamentals of software engineering*. Springer, 2011, pp. 343–350.
7. B. F. van Dongen. Bpi challenge 2017 - offer log. [Online]. Available: https://doi.org/10.4121/12705737.v2
8. A. Abbad-Andaloussi, A. Burattin, T. Slaats, E. Kindler, and B. Weber, "Complexity in declarative process models: Metrics and multi-modal assessment of cognitive load," *Expert Systems with Applications*, vol. 233, p. 120924, 2023.
9. A. Berti, S. J. Van Zelst, and W. van der Aalst, "Process mining for python (pm4py): bridging the gap between process-and data science," *arXiv preprint arXiv:1905.06169*, 2019.
10. A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.
11. B. F. van Dongen, "Bpi challenge 2017," 2017. [Online]. Available: https://data.4tu.nl/articles/_/12696884/1
12. ——. Bpi challenge 2019. [Online]. Available: https://doi.org/10.4121/UUID:D06AFF4B-79F0-45E6-8EC8-E19730C248F1
13. S. Debois and T. Slaats, "The analysis of a real life declarative process," in *IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015*. IEEE, 2015, pp. 1374–1382.
14. S. Zugal, P. Soffer, C. Haisjackl, J. Pinggera, M. Reichert, and B. Weber, "Investigating expressiveness and understandability of hierarchy in declarative business process models," *Software & Systems Modeling*, vol. 14, pp. 1081–1103, 2015.

15. C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber, "Understanding declare models: strategies, pitfalls, empirical results," *Software & Systems Modeling*, vol. 15, no. 2, pp. 325–352, 2016.
16. O. Turetken, A. Dikici, I. Vanderfeesten, T. Rompen, and O. Demirors, "The influence of using collapsed sub-processes and groups on the understandability of business process models," *Business & Information Systems Engineering*, vol. 62, pp. 121–141, 2020.
17. H. Verbeek, "The log skeleton visualizer in prom 6.9: The winning contribution to the process discovery contest 2019," *International Journal on Software Tools for Technology Transfer*, vol. 24, no. 4, pp. 549–561, 2022.
18. F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Information Systems*, vol. 74, pp. 136–152, 2018.
19. C. D. Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Transactions on Management Information Systems (TMIS)*, vol. 5, no. 4, pp. 1–37, 2015.
20. V. Nekrasaite, A. T. Parli, C. O. Back, and T. Slaats, "Discovering responsibilities with dynamic condition response graphs," in *Advanced Information Systems Engineering: 31st International Conference, CAiSE 2019*. Springer, pp. 595–610.
21. S. Debois, T. T. Hildebrandt, P. H. Laursen, and K. R. Ulrik, "Declarative process mining for dcr graphs," in *Proceedings of SAC*, 2017, pp. 759–764.
22. M. W. T. N. Sergey Smirnov, Hajo A. Reijers, "Business process model abstraction: a definition, catalog, and survey," *Distributed and Parallel Databases*, vol. 30, pp. 63 – 99, 2012.
23. R. Jagadeesh Chandra Bose and W. M. Van der Aalst, "Abstractions in process mining: A taxonomy of patterns," in *Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, 2009. Proceedings 7*, pp. 159–175.
24. N. Tax, B. Dalmas, N. Sidorova, W. M. van der Aalst, and S. Norre, "Interest-driven discovery of local process models," *Information Systems*, vol. 77, 2018.
25. F. Mannhardt, M. De Leoni, H. A. Reijers, W. M. Van Der Aalst, and P. J. Toussaint, "From low-level events to activities-a pattern-based approach," in *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, 2016. Proceedings 14*. Springer, pp. 125–141.
26. M. Leemans, W. M. Van Der Aalst, and M. G. Van Den Brand, "Recursion aware modeling and discovery for hierarchical software event log analysis," in *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2018, pp. 185–196.
27. R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Bpmn miner: Automated discovery of bpmn process models with hierarchical structure," *Information Systems*, vol. 56, pp. 284–303, 2016.
28. R. J. C. Bose, E. H. Verbeek, and W. M. van der Aalst, "Discovering hierarchical process models using prom," in *IS Olympics: Information Systems in a Diverse World: CAiSE Forum 2011, London, UK, June 20-24, 2011, Selected Extended Papers 23*. Springer, 2012, pp. 33–48.
29. S. J. Leemans, K. Goel, and S. J. van Zelst, "Using multi-level information in hierarchical process mining: Balancing behavioural quality and model complexity," in *2020 2nd International Conference on Process Mining (ICPM)*, pp. 137–144.
30. X. Lu, A. Gal, and H. A. Reijers, "Discovering hierarchical processes using flexible activity trees for event abstraction," in *2020 2nd International Conference on Process Mining (ICPM)*. IEEE, 2020, pp. 145–152.

31. R. Bergenthum, "Prime miner-process discovery using prime event structures," in *2019 International Conference on Process Mining (ICPM)*. IEEE, pp. 41–48.
32. J. Mendling, *Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness*. Springer, 2008, vol. 6.
33. A. Augusto, J. Mendling, M. Vidgof, and B. Wurm, "The connection between process complexity of event sequences and models discovered by process mining," *Information Sciences*, vol. 598, pp. 196–215, 2022.
34. P. H. P. Richetti, F. A. Baião, and F. M. Santoro, "Declarative process mining: Reducing discovered models complexity by pre-processing event logs," in *Business Process Management*. Springer, 2014, pp. 400–407.

Chapter 9

Verifiable declarative process models

9.1 Transforming Dynamic Condition Response Graphs to safe Petri Nets

Remark 9.1. *The work has been published as [44]: Cosma VP, Hildebrandt TT, Slaats T. Transforming Dynamic Condition Response Graphs to Safe Petri Nets. International Conference on Applications and Theory of Petri Nets and Concurrency 2023 May 28 (pp. 417-439). Cham: Springer Nature Switzerland.*

Transforming Dynamic Condition Response Graphs to safe Petri Nets

Vlad Paul Cosma[1,2], Thomas T. Hildebrandt[2], and Tijs Slaats[2]

[1] KMD, Ballerup, Denmark
vco@kmd.dk
[2] Computer Science Department, Copenhagen University, Denmark
{vco,hilde,slaats}@di.ku.dk

Abstract. We present a transformation of the Dynamic Condition Response (DCR) graph constraint based process specification language to safe Petri Nets with inhibitor and read arcs, generalized with an acceptance criteria enabling the specification of the union of regular and ω-regular languages. We prove that the DCR graph and the resulting Petri Net are bisimilar and that the bisimulation respects the acceptance criterium. The transformation enables the capturing of regular and omega-regular process requirements from texts and event logs using existing tools for DCR requirements mapping and process mining. A representation of DCR Graphs as Petri Nets advances the understanding of the relationship between the two models and enables improved analysis and model checking capabilities for DCR graph specifications through mature Petri net tools. We provide a python script implementing the transformation from the DCR XML export format to the PNML exchange format extended with arc types. In the implementation, all read arcs are replaced by a pair of standard input and output arcs. This directly enables the simulation and analysis of the resulting Petri Nets in tools such as TAPAAL, but means that the acceptance criterium for infinite runs is not preserved.

Keywords: Petri Nets · DCR graphs · Bisimilarity

1 Introduction

Whereas process control-flow is traditionally captured using imperative notations such as Business Process Modelling Notation (BPMN), process requirements for information systems are typically presented as declarative rules, describing the constraints (i.e. provisions and obligations) for the execution of individual tasks in a process. For instance, a requirement for an e-shop application may specify that payment information must be provided before a payment can be made, and that a payment can be made and is required to eventually happen, if an order has been made. The requirements are typically translated to imperative code when the system is implemented.

In this paper we consider the transformation from process requirements presented in the declarative Dynamic Condition Response (DCR) graphs notation

to processes expressed in a variant of the well-known process notation of Petri Nets [40]. The DCR graphs notation was introduced in [26, 13] as a formal specification language for distributed workflows and further developed in a range of papers, e.g. adding time, sub processes and data (see e.g. [14, 28, 15, 33]).

In its core form, DCR graphs is a graph-based notation with a single kind of node and a few basic relations. Nodes of the graph denote actions (or events) of the process and four kinds of directed edges between nodes denoting constraints and effects between actions, as will be explained below in our e-shop running example. The core DCR graph notation can express all regular and ω-regular languages [8] and in particular liveness properties, e.g. that some action must eventually happen (not to be confused with the standard notion of live Petri Nets). The fact that DCR graphs can express all ω-regular languages makes the notation more expressive than the classical declarative process language of Linear-time Temporal Logic (LTL) [29] that can only express the star-free omega-regular languages. The DCR graph notation is also different from LTL in that it has an operational execution semantics, similarly to Petri Nets expressed as a marking on the nodes of the graph.

The declarative nature and operational semantics makes DCR graphs similar to the model of Petri Nets, yet there are still notable differences. Firstly, DCR graphs abstracts from the notion of places, which is prominent for Petri Nets. Secondly, DCR graphs can directly express infinitary languages and liveness properties. This makes DCR graphs closer to traditional declarative notations such as LTL and textual representations of rules. Indeed, the highlighter tool [21] supports the mapping back and forth between textual requirement specifications and DCR graphs. On the other hand, Petri Nets with their notion of tokens, branching and loops are closer to imperative notations such as BPMN processes. Moreover, while DCR graphs are supported by design and specification tools and process engines used by industry[3], there are still no powerful model checking tools as it is the case for Petri Nets, such as the TAPAAL tool [6].

Thus, the motivation for providing the transformation of DCR graphs to safe Petri Nets with inhibitor and read arcs is threefold, as illustrated in Fig. 1: Firstly, we provide a path for transforming declarative requirements supported by industrial design tools to Petri Nets, which are closer to imperative process models such as BPMN. Secondly, the transformation enables the use of Petri Nets verification and analysis tools, notably the TAPAAL tool [6], for DCR graph specifications. Finally, the transformation allows us to use the DisCoveR miner [3] to mine Petri nets via an intermediate DCR graph representation. Hereby we get the high accuracy of DisCoveR in an imperative model and maintain a higher degree of concurrency in the model than is usually the case for block-structured approaches. This was already demonstrated with an early (unsound) translation of DCR graphs to Petri Nets, which managed to win the prize for best imperative miner in the 2021 Process Discovery Contest[4].

[3] Available freely for academic use at DCRSolutions.net

[4] `https://icpmconference.org/2021/process-discovery-contest/`

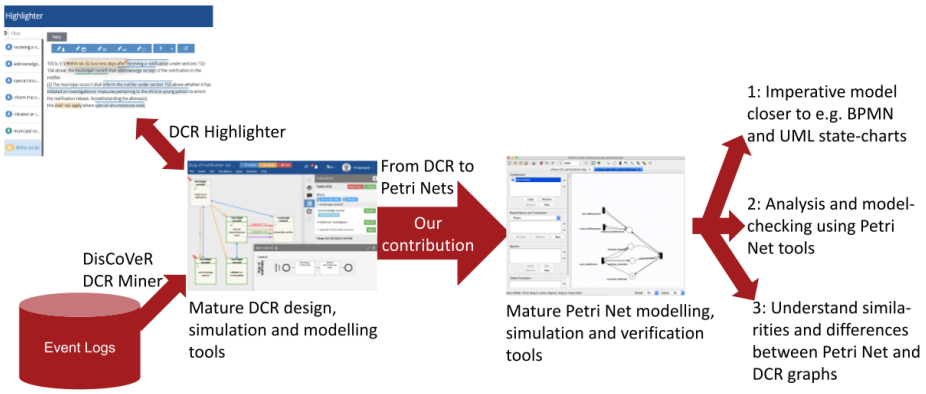


Fig. 1: Motivation for contributions of the paper

The paper is structured as follows. After the related work in Sec. 2, we give the definitions of core DCR graphs and Petri Nets with inhibitor arcs, read arcs and pending places in Sec. 3. We then proceed in Sec. 4 to provide the transformation of DCR graphs to Petri Nets, which is done by induction in the number of relations of the DCR graph. We also provide a sketch proof of the bisimilarity between the safe Petri Net and the DCR graph. Next we show how we reduce the size of the mapping in Sec. 5. We exemplify the mapping with a simple e-shop process. As usual we conclude and discuss future work in Sec. 6.

2 Related work

Several notations for declarative process modelling have been developed. In addition to DCR graphs, the Declare [1] and Guard-Stage-Milestone (GSM) notations have also seen broad use in the business process management research community.

Declare provides a set of templates for modelling business constraints that are formalised as LTL formulae (parameterized by activities). A Declare model is the conjunction of a set of instantiated formulae. Given the limited expressiveness of the templates, a mapping from DCR graphs to Declare is not possible. Declare has been formalized in other languages such as coloured automata [22] and SCIFF [23, 24]. Mappings from Declare to Petri Nets and R/I-nets were provided respectively in [31] and [7], however proofs of correctness are missing from each of these.

The GSM notation [20] takes a declarative data-centric approach to modelling processes, where stages of activities in the process are connected through guards that need to be satisfied for their activation and milestones that represent their acceptance criteria. A mapping has been proposed from Petri Nets to GSM [30], in particular with a focus on representing the output of process discovery algorithms (which usually produce Petri Nets) as GSM models. We

are not aware of any direct mappings in the opposite direction. Similarly [10] provides a mapping from DCR graphs to GSM models, an opposite mapping is mentioned as future work but has not yet materialised.

In [12] a subset of the DCR relations and their equivalent Petri Net mapping is presented, without inhibitor arcs without proof of correctness. [27] provides an encoding of DCR graphs as Büchi automata.

Petri Nets are widely used, and therefore there are also many translations to notations outside the declarative process modelling sphere, for example Ladder Logic Diagrams [36], Timed Automata [5] and mCRL2[32].

Similarly much work has gone into mapping other modelling notations into Petri Nets, such as UML activity diagrams [35], UML sequence diagrams [39], UML state charts [19], and BPMN [9, 32].

The work in [23] presents logic-based approaches which formalize regulatory models by relying on the deontic notions of obligations and permissions.

Different classes of ω-language Petri Nets have been introduced in [37] and their complexity has been studied in [11]. The definition of acceptance criteria for infinite words in [37] is based on markings being visited infinitely often, similar to the acceptance criteria of Büchi-automata. This differs from the acceptance criteria introduced in the present paper, which is based on pending places, for which tokens cannot rest infinitely without being consumed by a transition being fired.

3 Preliminaries

In this section we provide the running example and the formal definitions of Dynamic Condition Response graphs and safe Petri Nets with inhibitor and read arcs and pending places.

3.1 Running Example

We consider as running example a simple e-shop application that has the following specification:

(i) If an order is added, a payment for the order must eventually be made.
(ii) Payment information (eg. credit card number) must be provided before a payment can be executed.
(iii) The payment information can be edited any number of times.
(iv) A new order cannot be added before a subsequent payment has been made and payment can only be made if an order has been added and is not yet paid.

We can identify three actions in the system: Edit (or initially provide) Payment Information, Add Order and Make Payment. Below we will see how to model processes that fulfil these requirements as respectively DCR graphs and safe Petri Nets with inhibitor and read arcs and pending places.

3.2 Dynamic Condition Response graphs

We give a formal definition of core Dynamic Condition Response (DCR) graphs as attributed directed graphs.[5] For a set A we write $\mathcal{P}(A)$ for the set of all subsets of A, i.e. the powerset of A and $\mathcal{P}_{ne}(A)$ for the set of all non-empty subsets of A.

Definition 1. *A* DCR graph G *is given by a tuple* $(E, M, R, @, L, l)$ *where*

(i) E *is a finite set of* events
(ii) $M = (Ex, Re, In) \in \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$ *is the* marking
(iii) $R \subseteq E \times E$ *is the set of* relations *between events*
(iv) $@ : R \to \mathcal{P}_{ne}(\{\bullet\!\!\leftarrow, \bullet\!\!\rightarrow, \rightarrow\!\!+, \rightarrow\!\!\%\})$ *is the* relation type assignment
(v) L *is the set of action labels*
(vi) $l : E \to L$ *is the* labelling function *assigning an action label to each event*

The marking $M = (Ex, Re, In)$ *describes the state of an event* e *in the following way. If* e *has been* executed *at least once then* $e \in Ex$*. If* e *is* pending *(i.e. it must eventually be executed) then* $e \in Re$*. If* e *is* included *(i.e. it is currently relevant) then* $e \in In$*.*

Assume a relation $r = (e, e') \in R$ from event e to e'. If $\bullet\!\!\leftarrow \in @r$ we say r is a constraining relation. If $@r \cap \{\bullet\!\!\rightarrow, \rightarrow\!\!+, \rightarrow\!\!\%\} \neq \emptyset$ we say that r is an effect relation. Note that r can be both a constraining and an effect relation at the same time. We write $e \bullet\!\!\leftarrow e'$ (or $e' \rightarrow\!\!\bullet\ e$) if $\bullet\!\!\leftarrow \in @r$ and say there is a *condition* from e' to e. We write $e \bullet\!\!\rightarrow e'$ if $\bullet\!\!\rightarrow \in @r$ and say there is a response from e to e'. We write $e \rightarrow\!\!+ e'$ if $\rightarrow\!\!+ \in @r$ and say there is an include from e to e'. Finally, we write $e \rightarrow\!\!\% e'$ if $\rightarrow\!\!\% \in @r$ and say there is an exclude from e to e'.

The behaviour of a DCR graph is given by a labelled transition system, where the states are markings and the transitions are the execution of a labelled event. Hereto comes a definition of when a finite or infinite execution sequence is accepting or not. We first define when events are enabled, i.e. can be executed.

Definition 2 (Event enabling). *Let* $(E, M, R, @, L, l)$ *be a DCR graph. An event* $e \in E$ *is* enabled for the marking $M = (Ex, Re, In)$*, writing* $\mathsf{enabled}(M, e)$ *if and only if:*

(i) $e \in In$
(ii) $\forall e' \in In.\ e' \rightarrow\!\!\bullet e \implies e' \in Ex$

The conditions for event enabling state that for an event e to be enabled, (i) it must be included. (ii) Whenever e has a condition relation from an included event e', then this e' was executed at least once.

We now define the effect of executing an event e for a given marking M.

[5] The presentation deviates slightly from the original definition given in [13] to facilitate the definition of the mapping to Petri Nets, but defines the same graph structures.

Definition 3. *Let G be a DCR graph with marking $M = (Ex, Re, In)$. The effect of executing an enabled event e is the marking* $\mathsf{effect}_G(M, e) = (Ex', Re', In')$ *where*

$$\begin{aligned} Ex' =& Ex \cup \{e\} \\ Re' =& (Re \setminus \{e\}) \cup \{e' \mid e \bullet\!\!\rightarrow e'\} \\ In' =& (In \setminus \{e' \mid e \rightarrow\!\!\!\!\% \; e'\}) \\ & \cup \{e' \mid e \rightarrow\!\!+ \; e'\} \end{aligned}$$

We are now ready to define the labelled transition semantics for DCR graphs.

Definition 4. *Let $G = (E, M, R, @, L, l)$ be a DCR graph. Define a labelled transition relation between markings by $M \xrightarrow{e}_G \mathsf{effect}_G(M, e)$ if* $\mathsf{enabled}(M, e)$*, where $e \in E$. Write $M \Rightarrow M'$ for $\exists e \in E. M \xrightarrow{e}_G M'$ and write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$. Define $\mathbb{M}_G = \{M' \mid M \Rightarrow^* M'\}$, i.e. the set of all reachable markings from the initial marking M of G. The labelled transition system for G is then defined as $[[G]] = (\mathbb{M}_G, M, \rightarrow_G \subset (\mathbb{M}_G \times E \times \mathbb{M}_G), L, l)$.*

Finally, we define when a finite or infinite execution sequence of a DCR graph is *accepting*. Intuitively, it is required that any included and pending event e in some intermediate state must eventually be executed or no longer included or pending in a later state. If one limits attention to finite execution sequences, the acceptance criteria is that no pending event is included in the final state.

Definition 5. *Let $G = (E, M_0, R, @, L, l)$ be a DCR graph. A finite or infinite sequence of transitions $M_0 \xrightarrow{e_0}_G M_1 \xrightarrow{e_1}_G \ldots$ in $[[G]]$ with $M_i = (Ex_i, Re_i, In_i)$, is* accepting *if $e \in Re_i \cap In_i$ implies $\exists j \geq i.(e_j = e \vee e \notin Re_j \cap In_j)$.*

A DCR graph modelling our running example is shown in Fig 2. Events are depicted as boxes containing the action label of the event and relations as arrows. A relation with multiple types is depicted as multiple arrows between the same two events, one arrow for each type. Events that are included in the initial marking are drawn as boxes with a solid border, events that are excluded in the initial marking are drawn as boxes with a dashed border. Consequently, the events labelled EditPaymentInfo and AddOrder are initially included and the event labelled MakePayment is excluded in the initial marking of the graph.

The first requirement, "If an order is made, a payment for the order must eventually be made" is modelled by a response relation ($\bullet\!\!\rightarrow$ in blue) and an include relation ($\rightarrow\!\!+$ in green) from the event labelled AddOrder to the event labelled MakePayment. (The include relation is needed because of the interplay with the fourth requirement described below).

The second requirement, "Payment information (eg. credit card number) must be provided before a payment can be executed" is modelled by a condition relation ($\rightarrow\!\!\bullet$ or $\bullet\!\!\leftarrow$ in orange) from the event labelled EditPaymentInfo to the event labelled MakePayment.

The third requirement, "The payment information may be provided at any time and any number of times." is modelled by having no condition relations

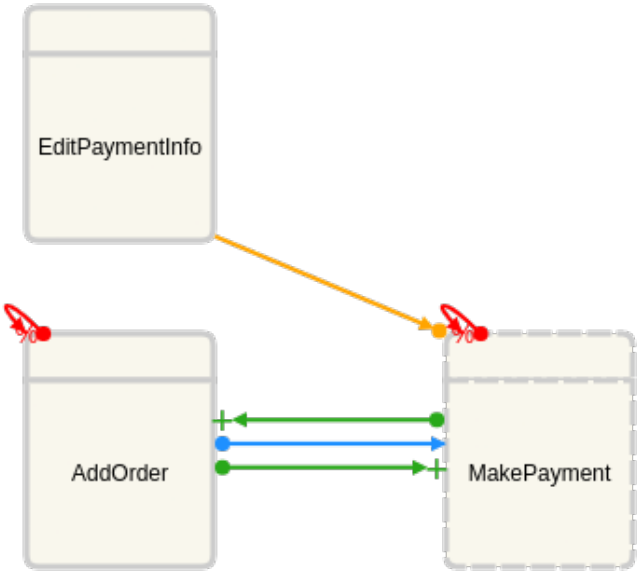


Fig. 2: DCR graph specification for the e-shop process

pointing to the event labelled EditPaymentInfo and making sure that it is included in the initial marking and never excluded.

The forth requirement is in two parts. The first part, "a new order cannot be made before a subsequent payment has been made" is modelled by an exclude relation ($\rightarrow\!\!\!\!\!\!\%$ in red) from AddOrder to itself and an include relation from MakePayment to AddOrder. The effect is that when AddOrder is executed, it excludes itself and is thus no longer available, except if MakePayment is executed, which will include AddOrder again. The second part, "payment can only be made if an order has been made and is not yet paid" is similarly modelled by an exclusion relation from MakePayment to itself and an inclusion relation from MakePayment to AddOrder.

3.3 Petri Nets with inhibitor arcs, read arcs and pending places

There are numerous variants of Petri Nets with different expressive power. As described in the introduction, we use safe Petri Nets with inhibitor and read arcs and a notion of both finite and infinite acceptance criteria. Inhibitor arcs (also called negative contextual arcs) are special arcs between places and transitions specifying the constraint that the transition is only enabled if all places related to it by inhibitor arcs are empty. In general, the addition of inhibitor arcs makes the model of Petri Nets Turing complete [2]. However, with the additional requirement of safeness, which means that places can hold at most one token (also known as the property of all the net places being 1-bounded), the notation is restricted to finite state models.

Read arcs (also called test, activator or positive contextual arcs) [4] specify the constraint that a transition is only enabled if all places related to it by read arcs have a token. A key difference between having a read arc and a pair of input and output arcs between a transition and a place, is that read arcs are not consuming the token. This means that two transitions with read arcs to the same place can occur concurrently [25]. However, if two transitions are connected to the same place by a read arc and a standard input arc respectively, the two transitions will still be in conflict.

The acceptance criteria we introduce is inspired by DCR graphs and allows us to conveniently express the union of regular and ω-regular languages, without needing to refer to explicit markings. The acceptance criteria is defined by indicating a subset of the states to be so-called *pending* places, and then define a finite or infinite execution sequence to be accepting if any token on a pending place is eventually subsequently consumed (but possibly placed back) by the execution of a transition. If one limits attention to finite execution sequences, the acceptance criteria is that all pending places are empty at the end of the execution. Note that the use of read arcs allows us to test, if there is a token on a pending place without consuming it.

We define Petri Nets with inhibitor arcs, read arcs and pending places as follows.

Definition 6. *A Petri Net with inhibitor and read arcs and pending places (PNirp) is a tuple* $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$*, where*

(i) P *is a finite set of places,*
(ii) T *is a finite set of transitions s.t.* $P \cap T = \emptyset$*,*
(iii) $A = IA \sqcup OA$ *is a finite set of input and output arcs, where:*
  (1) $IA \subseteq P \times T$ *is a finite set of input arcs,*
  (2) $OA \subseteq T \times P$ *is a finite set of output arcs,*
(iv) $Inhib$*:* $IA \longrightarrow \{true, false\}$ *is a function defining inhibitor arcs,*
(v) $Read$*:* $IA \longrightarrow \{true, false\}$ *is a function defining read arcs,*
(vi) Act is a set of labels (actions),
(vii) $\lambda : T \to Act$ *is a labelling function,*
(viii) $Pe \subseteq P$ *is the set of pending places,*

and the constraint that if $Inhib((p,t))$ *then* $\neg Read((p,t))$ *and if* $Read((p,t))$ *then* $\neg Inhib((p,t)) \wedge (t,p) \notin OA$*. That is, an input arc cannot be both a read arc and an inhibitor arc. And if there is a read arc from place* p *to transition* t*, then there cannot be an output arc from transition* t *to* p*.*

We only consider 1-bounded places in the present paper, which means that markings can be defined as simply a subset of places (the places containing a token).

Definition 7. ***(safe Marking).*** *Let* $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$ *be a PNirp. A safe marking* M *on* N *is a subset* $M \subseteq P$ *of places. We say there is a token* x *at a place* $p \in P$*, written* $x \in M(p)$*, if* $p \in M$*. The set of all markings over* N *is denoted by* $M(N)$*.*

We say that a Petri Net is safe if the execution of transitions preserves the safeness of markings. In this paper we will work only with safe Petri Nets, in particular we prove that the mapping from DCR graphs to Petri Nets provided in the next section always yields a safe Petri Net.

Assuming the Petri Net to be safe simplifies the definition of enabledness of transitions defined as follows.

Definition 8. ***(Enabledness)****. Let $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$ be a $PNirp$. We say that a transition $t \in T$ is enabled in a marking M, if*

(i) for $t \in T$ we have $\{p \in P \mid (p,t) \in IA \wedge \neg Inhib((p,t)\} \subseteq M$, i.e. for all input arcs except the inhibitor arcs there is a token in the input place,
(ii) for $t \in T$ we have $\{p \in P \mid (p,t) \in IA \wedge Inhib((p,t)\} \cap M = \emptyset$, i.e. for all inhibitor arcs there is not *a token in the input place,*

We abuse notation and, just as for DCR graphs, let $\mathsf{enabled}(M,t)$ *denote that the transition t is enabled in marking M.*

Next we formalise the effect of executing (or firing) a transition. Again it is simplified by the assumption of safeness and we use the same notation as for DCR graphs to denote the result of firing a transition.

Definition 9. ***(Firing rule).*** *Let $N = (P, T, A, Inhib, Read, Act, \lambda, Pe)$ be a $PNirp$, M a marking on N and $t \in T$ a transition. If* $\mathsf{enabled}(M,t)$ *with $Input(t) = \{p \in P \mid (p,t) \in IA \wedge \neg Inhib((p,t)) \wedge \neg Read((p,t))\}$ and $Output(t) = \{p \in P \mid (t,p) \in OA\}$ then t can fire, i.e. be executed, and produce a marking* $\mathsf{effect}_G(M,t) = (M \setminus Input) \cup Output$.

For convenience in the construction, we include the marking M in the $PNirp$ tuple and we use $N = (P, M, T, A, Inhib, Read, Act, \lambda, Pe)$ to refer to a safe marked $PNirp$ with marking $M \subseteq P$.

Similarly to how DCR graphs define labelled transition systems, the firing rule defines a labelled transition system for a $PNirp$ with markings as states and the labelled Petri Net transitions as labels.

Definition 10. *Let $N = (P, M, T, A, Inhib, Read, Act, \lambda, Pe)$ be a $PNirp$ with safe marking M. Define a labelled transition relation between markings by $M \xrightarrow{e}_N$* $\mathsf{effect}_G(M,t)$ *if* $\mathsf{enabled}(M,t)$*, where $t \in T$. Write $M \Rightarrow M'$ for $\exists t \in T. M \xrightarrow{t}_N M'$ and write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$. Define $\mathbb{M}_N = \{M' \mid M \Rightarrow^* M'\}$, i.e. the set of all reachable markings from the initial marking M of N. The labelled transition system for N is then defined as $[[N]] = (\mathbb{M}_N, M, \rightarrow_N \subset (\mathbb{M}_N \times T \times \mathbb{M}_N), Act, \lambda)$.*

Finally, we define when a finite or infinite execution sequence of a $PNirp$ is *accepting*.

Definition 11. *Let $N = (P, M, T, A, Inhib, Read, Act, \lambda, Pe)$ be a $PNirp$ with safe marking M. A finite or infinite sequence of transitions $M_0 \xrightarrow{t_0}_N M_1 \xrightarrow{t_1}_N \ldots$ in $[[N]]$ is* accepting *if $p \in M_i \cap Pe$ implies $\exists j \geq i.(p, t_j) \in IA$.*

Figure 3 shows the safe Petri Net resulting from the implemented optimized transformation of the running example DCR graph. The place pending_included_MakePayment is the only pending place. The arcs between the transition pend_MakePayment and the place executed_EditPaymentInfo are in fact a read arc in the transformation, but represented as a pair of standard input and output arcs in the implementation so we can simulate it in the TAPAAL tool.

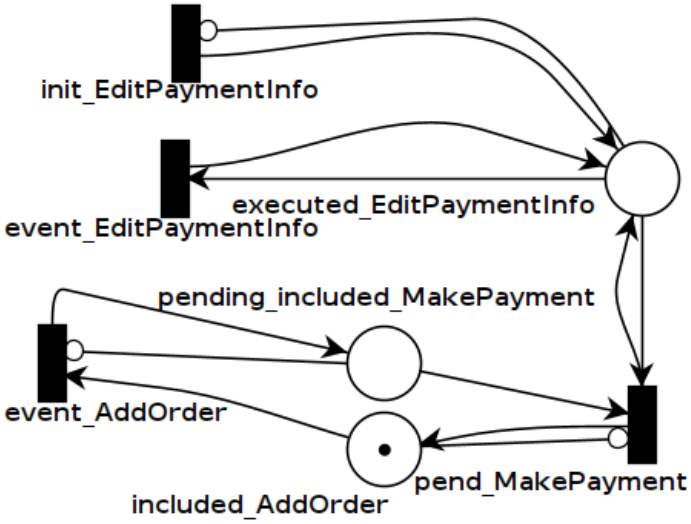


Fig. 3: E-shop Petri Net resulting from the transformation implementation.

It is worth noting that in the Petri Net we need two transitions labelled with the action EditPaymentInfo, namely a transition init_EditPaymentInfo mapping the initial execution (or the initial entry of the payment information) and a transition event_EditPaymentInfo mapping subsequent executions.

4 Mapping DCR graphs to Petri Nets

In this section we provide the mapping from DCR graphs to marked safe Petri Nets with inhibitor and read arcs and pending places and prove that the DCR graph and the Petri Net have bisimilar transition semantics. The mapping has been implemented as a python script, which can be found at: `https://github.com/paul-cvp/dcr-to-tapn.git`, where we also provide some results of our mapping. We support the standard PNML[17] exchange format extended with arc types [16]. A key difference in the code mapping is that read arcs are automatically translated to a pair of input and output arcs. This was a design choice in order to maintain compatibility with a greater number of Petri Net verification tools and the difference only has consequences for the degree of concurrency

and the acceptance criteria for infinite runs. In what follows we use the notation — (a straight line) to refer to read arcs and use the notation <—> to refer to a pair of input and output arcs.

The core part of the mapping is given as a function $DP : DCR \to PNirp$, defined inductively in the number of relations of the DCR graph. The Petri Net $DP(G)$ will have the events of G as labels (actions), since there will in general be more than one transition representing each event of G. To get the same observable behaviour as the DCR graph G, we then subsequently just need to compose the labelling function of of $DP(G)$ with the event labelling function of G. Due to the rich structure of DCR graph markings, the basic inductive mapping in general produces a number of unused places and transitions. These can subsequently be removed by searching for unreachable transitions and places and merging places with the same arcs.

As part of the mapping $DP : DCR \to PNirp$, we also define for every $G \in DCR$ a mapping $DPM_G : \mathbb{M}_G \to \mathbb{M}_{DP(G)}$, i.e. from markings of G to the markings of $DP(G)$. For a DCR graph $G = (E, M, R, @, L, l)$ and $DP(G) = (P_{DP(G)}, M_{DP(G)}, T_{DP(G)}, A_{DP(G)}, Inhib_{DP(G)}, Read_{DP(G)}, Act_{DP(G)}, \lambda_{DP(G)}, Pe_{DP(G)})$ we then have $M_{DP(G)} = DPM_G(M)$ and $Act_{DP(G)} = E$.

The two mappings are defined so we get the following precise semantic correspondence between the two process models. (Note we write $\exists!$ to mean "there exists a unique").

Theorem 1. ***(Bisimilarity)*** *For $G \in DCR$ we have that the relation $Sim_G = \{(M, DPM_G(M)) \mid M \in \mathbb{M}_G\}$ is a bisimulation relation between $[[G]]$ and $[[DP(G)]]$, in the sense that $(M_0, DPM_G(M_0)) \in Sim_G$, where M_0 is the initial marking of G and for all $(M, DPM_G(M)) \in Sim$, we have*

(i) $M \xrightarrow{e} M'$ *implies* $\exists! t \in T_{DP(G)}.DPM_G(M) \xrightarrow{t} DPM_G(M')$ *and* $\lambda(t) = e$,
(ii) $DPM_G(M) \xrightarrow{t} M'$ *and* $\lambda(t) = e$ *implies* $M \xrightarrow{e} M''$ *and* $DPM_G(M'') = M'$.

That is, for every enabled event in a marking M of the DCR graph we have a unique enabled transition in the corresponding marking $DPM_G(M)$ of the Petri Net which is labelled by the event e and firing the transition changes the marking of the Petri Net to the marking corresponding to the DCR marking resulting from executing e - and vice versa. We will see below, that in addition the bisimulation also pairs accepting runs.

We now proceed to define the mapping function and outline the proof of Theorem 1 along the way. For each event $e \in E$ of the DCR graph G, there will be four places in $DP(G)$, which we will write as P_e^{Ex}, P_e^{In}, P_e^{Re}, and P_e^{Rex}. The first two places represent respectively if the event e has been executed and if it has been included. The last two places record the pending response state of the event e by a token in P_e^{Re} if and only if the event e is pending and included, and a token in P_e^{Rex} if and only if the event e is pending and excluded. The places P_e^{Re} will constitute the set $Pe_{DP(G)}$ of pending places.

Definition 12. ***(Places mapping)*** *Let $G = (E, M, R, @, L, l) \in DCR$. Define the corresponding Petri Net places of $DP(G)$ as $P_{DP(G)} = \{P_e^\gamma | e \in E, \gamma \in$*

$\{Ex, In, Re, Rex\}\}$. *Define the corresponding pending places of* $DP(G)$ *as* $Pe_{DP(G)} = \{P_e^{Re} | e \in E\}$.

Definition 13. ***(Markings mapping)*** *Let* $G = (E, M_0, R, @, L, l) \in DCR$ *and* $\mathbb{M}_G$ *be the reachable markings of* G *and* $\mathbb{M}_{DP(G)} = \mathcal{P}(P_{DP(G)})$*, i.e. all safe markings of the places* $P_{DP(G)}$ *defined above. Define* $DPM_G : \mathbb{M}_G \to \mathbb{M}_{DP(G)}$ *as follows. For* $M = (Ex, Re, In) \in \mathbb{M}_G$ *define* $DPM_G(M)$ *such that for any event* $e \in E$*,*

(i) $P_e^{Ex} \in DPM_G(M) \iff e \in Ex$
(ii) $P_e^{In} \in DPM_G(M) \iff e \in In$
(iii) $P_e^{Re} \in DPM_G(M) \iff e \in Re \wedge e \in In$
(iv) $P_e^{Rex} \in DPM_G(M) \iff e \in Re \wedge e \notin In$

The events and relations of a DCR graph are represented by respectively transitions and arcs in the Petri Net. Each event of the DCR graph will be represented by several transitions in the Petri Net. Indeed, the number of transitions representing each event depends on the number of relations in the DCR graph.

We define the corresponding Petri Net transitions $T_{PD(G)}$, arcs $A_{PD(G)}$ and labelling function $\lambda_{PD(G)}$ by induction in the number $k = |R|$ of relations. We will at the same time argue for the proof of Theorem 1, since it also follows from the inductive construction.

In the base case, $k = 0$ i.e. a DCR graph $G = G_0 = (E, M, \emptyset, @, L, l)$ without any relations, each event will be represented by a (sub) Petri Net as shown in Fig. 4 (assuming a marking M, where the event is included, not executed and not pending, i.e. $e \in In, e \notin Ex \cup Re$) which is completely independent of the similar sub Petri Nets representing the other events. We have four transitions for each event $e \in E$ of the DCR graph and arcs as shown in Fig. 4. We use the labelling function of the Petri Net to label all the transitions with the event e and thereby record that the transitions represent this event in the DCR graph. That is, we define the mapping formally as follows.

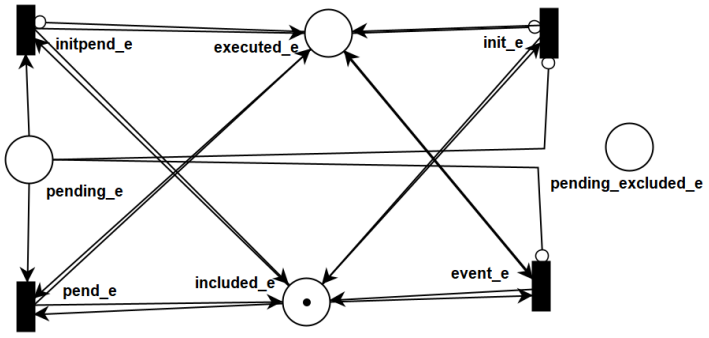


Fig. 4: Base case: Petri Net for a single included DCR event, which is not yet executed nor pending.

Definition 14. ***(Base case: Mapping a DCR graph with no relation)*** *Let $G = (E, M, \emptyset, @, L, l)$ be a DCR Graph with no relations. Then $P_{DP(G)} = \{p_e^\delta | e \in E, \delta \in \{In, Ex, Re, Rex\}\}$, $T_{DP(G)} = \{t_e^\delta | e \in E, \delta \in \{event, init, pend, initpend\}\}$ and $\lambda_{DP(G)}(t_e^\delta) = e$. The set of arcs $A_{DP(G)} = IA_{DP(G)} \cup OA_{DP(G)}$ and $Inhib_{DP(G)} : IA_{DP(G)} \to \{true, false\}$ are defined by Table 1. Each row in the table corresponds to one of the four transitions, and each column to one of the four places. Each entry is an arc in which the left arrow is an input arc and the right arrow is an output arc. That is, the input and output arc pair <—> in the entry of column p_e^{Ex} and row t_e^{event} in the table means that we have $(p_e^{Ex}, t_e^{event}) \in IA_{DP(G)}$ and $(t_e^{event}, p_e^{Ex}) \in OA_{DP(G)}$. A table entry of o— in the entry of column p_e^{Re} and row t_e^{event} means we add an arc $(p_e^{Re}, t_e^{event}) \in IA_{DP(G)}$, which is an inhibitor arc, i.e. $Inhib((p_e^{Re}, t_e^{event})) = true$, and we add no arc in $OA_{DP(G)}$. The read arc — between column p_e^{In} and row t_e^{event} is mapped as $(p_e^{In}, t_e^{event}) \in IA_{DP(G)}$ and $Read((p_e^{In}, t_e^{event})) = true$.*

	p_e^{In}	p_e^{Ex}	p_e^{Re}	p_e^{Rex}
t_e^{event}	—	<—>	o—	
t_e^{init}	—	o—>	o—	
$t_e^{initpend}$	—	o—>	<—	
t_e^{pend}	—	<—>	<—	

Table 1: Arc patterns for an event in the base case

Note that we have no arcs in any directions connected to the place p_e^{Rex}. This means that this place is redundant, unless more relations are added to the DCR graph, which will give rise to more transitions in the Petri Net. We will reuse the same table notation style for arc pattern mappings throughout the paper. Fig. 4 is a visual representation of Table 1.

Proof sketch of Thm.1: base case (1). It follows by a trivial inspection of the event execution cases and the different initial markings that we have the bisimulation property in Thm. 1 for the base case. If an event e is initially included and not executed nor pending, then we have the marking in Fig.4. Observe that only the transition labelled *init_e* can fire, which will read the token at the place *included_e* and put a token at the place *executed_e*. This corresponds to the execution semantics of DCR graphs. Subsequently, only the transition labelled *event_e* can fire and firing the transition will read the tokens at the places *included_e* and *executed_e*. If the event e is initially pending, included and not executed, it will fire first the transition *initpend_e* after which only the transition *event_e* can fire. If the event e is initially pending, included and executed, it will fire first the transition *pend_e* after which only the transition *event_e* that can fire. Finally, if the event is not included in the initial DCR marking, there will be no token in the *inlcuded_e* place and consequently no transition can fire. □

Now consider the induction step. Let $G = (E, M, R, @, L, l)$ be a DCR Graph with $R = \{r_1, \ldots, r_k, r_{k+1}\}$ and assume we have defined the mapping and proven Thm. 1 for $G_k = (E, M, \{r_1, \ldots, r_k\}, @, L, l)$.

We proceed by cases of the type $@r_{k+1}$ of the relation $r_{k+1} = (e, e')$. Effect relations change the marking of e' when e fires, and thus refines the transitions for e and adds arcs connected to the places recording the marking for e'. Dually, constraining relations requires a refinement of the transitions for e', adding arcs connected to the places recording the marking for e.

We first consider the cases where the relation $r_{k+1} = (e, e')$ is a single effect, i.e. $@r_{k+1} \in \{\{\rightarrow\!\!+\}, \{\rightarrow\!\!\!\!\%\}, \{\bullet\!\!\rightarrow\}\}$ and $e \neq e'$.

For $@r_{k+1} = \{\rightarrow\!\!+\}$ we replace each transition t_e^δ representing the event e with three new transitions $t_e^{0,\delta}$, $t_e^{1,\delta}$ and $t_e^{2,\delta}$, which in addition to the arcs connected to t_e^δ also get the new arcs shown in Table 2a. More formally we say that we apply the Def.15 below for relation $@r_{k+1} = \{\rightarrow\!\!+\}$ and the arc pattern table Table 2a.

Definition 15. ***(Inductive step: Mapping a DCR relation)***
Let $G = (E, M, R, @, L, l)$ be a DCR Graph, $APT(@r)$ be the arc pattern table for any given relation $r \in @R$ and $|APT(@r)_r|$ be the number of rows (transition copies) in the arc pattern table. Then $T_{DP(G_{k+1})} = T_{DP(G_k)} \backslash \{t_e^\delta \mid t_e^\delta \in T_{DP(G_k)}$ and $\lambda_{DP(G_k)}(t_e^\delta) = e\} \cup \{t_e^{i,\delta} \mid i \in \{0, 1, .., |APT(@r)_r| - 1\}\}$ and let $(p, t_e^{i,\delta}) \in IA_{DP(G_{k+1})}$ for $i \in \{0, 1.., |APT(@r)_r| - 1\}$ if and only if $(p, t_e^\delta) \in IA_{DP(G_k)}$ and let $(t_e^{i,\delta}, p) \in OA_{DP(G_{k+1})}$ for $i \in \{0, 1.., |APT(@r)_r| - 1\}$ if and only if $(t_e^\delta, p) \in OA_{DP(G_k)}$ or $(t_e^{i,\delta}, p)$ is one of the arcs in $APT(@r)$. Finally, for $t \in A_{DP(G_k)}$ such that $\lambda_{DP(G_k)} \neq e$, let $(p, t) \in IA_{DP(G_{k+1})}$ and $(t, p) \in OA_{DP(G_{k+1})}$ if and only if $(p, t) \in IA_{DP(G_k)}$ or $(t, p) \in OA_{DP(G_k)}$.

$\rightarrow\!\!+$	$p_{e'}^{In}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—		
$t_e^{1,\delta}$	o—>	—>	<—
$t_e^{2,\delta}$	o—>		o—

(a) Arc patterns for $\rightarrow\!\!+$

$\rightarrow\!\!\!\!\%$	$p_{e'}^{In}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	o—		
$t_e^{1,\delta}$	<—	o—	
$t_e^{2,\delta}$	<—	<—	—>

(b) Arc patterns for $\rightarrow\!\!\!\!\%$

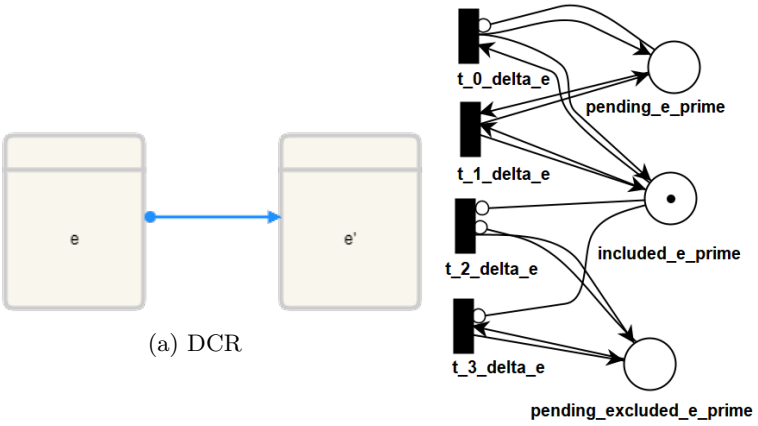
$\bullet\!\!\rightarrow$	$p_{e'}^{In}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—	o—>	
$t_e^{1,\delta}$	—	—	
$t_e^{2,\delta}$	o—		o—>
$t_e^{3,\delta}$	o—		—

(c) Arc patterns for $\bullet\!\!\rightarrow$

Table 2: Arc patterns for $r_{k+1} = (e, e')$ effect relations

The cases for $@r_{k+1} = \{\rightarrow\!\!\!\!\%\}$ and $@r_{k+1} = \{\bullet\!\!\rightarrow\}$ follow the same approach, by applying Def.15. Observe that we need four copies for each existing transition for e in the case of the response relation. The case for response is also illustrated graphically in Fig. 5.

Example 1. (Mapping the response relation) Figure 5 shows how the response relation $e \bullet\!\!\rightarrow e'$ is mapped by replacing each existing transition t_e^δ (t_delta_e) representing e by four new copies, connected to the places representing the marking of the event e'.



(a) DCR

(b) Petri Net

Fig. 5: Mapping (a) DCR response relation to a Petri Net notation (b)

Proof sketch of Thm.1: single effect mapping(2). First note that adding an effect relation from e to e' to the DCR graph only changes the output transitions representing e in $DP(G_k)$. Here we need three transitions, covering the different possibilities of the marking of e'. For the $e \rightarrow\!\!+ e'$, transition $t_e^{0,\delta}$ handles the case, where e' is already included, transition $t_e^{1,\delta}$ handles the case, where e' is not included, but pending and $t_e^{2,\delta}$ handles the case, where e' is not included and not pending. We follow a similar reasoning for $e \rightarrow\!\!+ e'$,$e \not\rightarrow e'$ and $e \bullet\!\!\rightarrow e'$. □

We now consider the constraining relation consisting of a single condition, i.e. $@r_{k+1} = \{\bullet\!\!\leftarrow\}$ and $e' r_{k+1} e$ and $e \neq e'$. For the condition relation we replace all existing transitions of e' with 3 new copies, again keeping the old arcs and adding new arcs to the places of e according to Table 3. This is also illustrated graphically in Fig. 6. Again we apply Def 15.

Proof sketch of Thm.1: single constraint mapping(3). The transition copy $t_e^{0,\delta}$ handles the case where e is included and already executed. The transition $t_e^{1,\delta}$ handles the case where e is excluded and not already executed. Finally, the transition $t_e^{2,\delta}$ handles the case where e is excluded and already executed. □

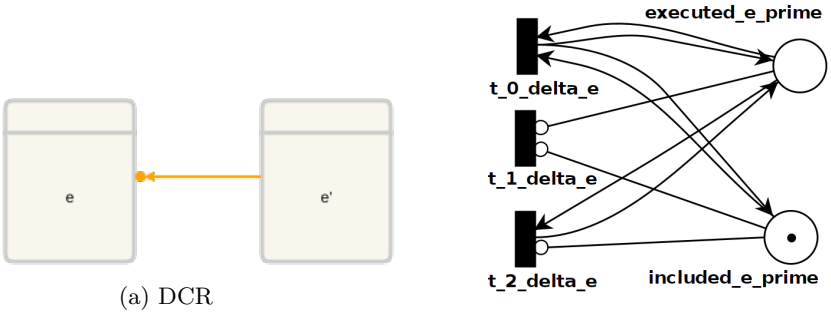
Example 2. (Mapping a condition relation.) Figure 6 shows how a condition relation is mapped between the transitions representing the DCR event e' and the places representing the execution and inclusion marking for the DCR event e.

Now we proceed to describe the cases of relations where the events e and e' are identical, and thereafter the cases of multiple relations between the same two events.

$\bullet\!\!\not\leftarrow$	$p_{e'}^{In}$	$p_{e'}^{Ex}$
$t_e^{0,\delta}$	—	—
$t_e^{1,\delta}$	o—	o—
$t_e^{2,\delta}$	o—	—

(a) Arc patterns for $\bullet\!\!\not\leftarrow$

Table 3: Arc patterns for $r_{k+1} = (e, e')$ constraint relation



(a) DCR

(b) Petri Net

Fig. 6: Mapping (a) DCR condition relation to a Petri Net notation (b)

Case $e = e'$ *and a single relation:* For single relations where the source and target is the same, we do not get the same multiplication of transitions:

(i) $@r = \{\rightarrow\!\!+\}$: Do nothing, since it can only take effect if e is already included.
(ii) $@r = \{\rightarrow\!\!\%\}$: Remove all output arcs from transitions t_e^{δ} (i.e. transitions with label e) to the place p_e^{In}, because the event can only be executed if there is a token at p_e^{In}, and that token should not be put back.
(iii) $@r = \{\bullet\!\!\rightarrow\}$: Add output arcs from all transitions t_e^{δ} to p_e^{Re} and replace all read arcs from p_e^{Re} to transitions t_e^{δ} with standard input arcs.
(iv) $@r = \{\bullet\!\!\not\leftarrow\}$: Remove the transitions $\{t_e^{init}, t_e^{initpend}\}$ and all their associated arcs.

We now consider the cases with multiple relations between the same two events, i.e. $|@r_{k+1}| > 1$. When we have both an include and exclude relation, i.e. $\{\rightarrow\!\!+, \rightarrow\!\!\%\} \subseteq @r_{k+1}$ we only apply the include relation, as the DCR graph semantics stipulate that first the exclusion takes place and then the inclusion.

Case $e = e'$ *and* $@r_{k+1} = \{\bullet\!\!\rightarrow, \rightarrow\!\!\%\}$*:* For all transitions t such that $\lambda_{DP(G_k)}(t) = e$, add output arcs from t to p_e^{Rex} and remove all output arcs from t to p_e^{In}. Define $\lambda_{DP(G_{k+1})} = \lambda_{DP(G_k)}$.

The remaining cases look at multiple relations $|@r_{k+1}| > 1$ between different events $e \neq e'$. We again follow the same reasoning as in part 2 and 3 of the proof, i.e. for a given $e\, r_{k+1}\, e'$ use its arc pattern table to make copies of existing

$\bullet\!\!\leftarrow \wedge \rightarrow\!\!\!+$	$p_{e'}^{In}$	$p_{e'}^{Ex}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—	—		
$t_e^{1,\delta}$	o—>	o—	—>	<—
$t_e^{2,\delta}$	o—>	o—		o—
$t_e^{3,\delta}$	o—>	—	—>	<—
$t_e^{4,\delta}$	o—>	—		o—

(a) Arc patterns for $e' \bullet\!\!\leftarrow e \wedge e \rightarrow\!\!\!+ e'$

$\bullet\!\!\leftarrow \wedge \rightarrow\!\!\!\!\%$	$p_{e'}^{In}$	$p_{e'}^{Ex}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	<—	—	o—	
$t_e^{1,\delta}$	<—	—	<—	—>
$t_e^{2,\delta}$	o—	o—		
$t_e^{3,\delta}$	o—	—		

(b) Arc patterns for $e' \bullet\!\!\leftarrow e \wedge e \rightarrow\!\!\!\!\% e'$

$\bullet\!\!\leftarrow \wedge \bullet\!\!\rightarrow$	$p_{e'}^{In}$	$p_{e'}^{Ex}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—	—	o—>	
$t_e^{1,\delta}$	—	—	—	
$t_e^{2,\delta}$	o—	o—		o—>
$t_e^{3,\delta}$	o—	o—		—
$t_e^{4,\delta}$	o—	—		o—>
$t_e^{5,\delta}$	o—	—		—

(c) Arc patterns for $e' \bullet\!\!\leftarrow e \wedge e \bullet\!\!\rightarrow e'$

Table 4: Effect and constraint pair arc patterns

transitions and their arcs and create new arc mappings to the existing places. Formally we apply Def. 15 for each relation and arc pattern table mentioned.

Case $e \neq e'$ *and* r_{k+1} *is both constraining and effect:* When an effect constraint pair exists, i.e. $\bullet\!\!\leftarrow \in @r_{k+1}$ and $@r_{k+1} \cap \{\bullet\!\!\rightarrow, \rightarrow\!\!\!\!\%, \rightarrow\!\!\!+\} \neq \emptyset$, their mapping produces arcs that both check the necessary places and also change their marking. Given $e\, r_{k+1}\, e'$, we consider the different cases as follows:

(i) $@r_{k+1} = \{\bullet\!\!\leftarrow, \rightarrow\!\!\!+\}$: The arc pattern is shown in Table 4a.
(ii) $@r_{k+1} = \{\bullet\!\!\leftarrow, \rightarrow\!\!\!\!\%\}$: The arc pattern is shown in Table 4b.
(iii) $@r_{k+1} = \{\bullet\!\!\leftarrow, \bullet\!\!\rightarrow\}$: The arc pattern is shown in Table 4c.

Note that brown arcs show the changes done to the arc patterns for the condition relation from Table 3.

$\bullet\!\!\rightarrow \wedge \rightarrow\!\!\!+$	$p_{e'}^{In}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—	o—>	
$t_e^{1,\delta}$	—	—	
$t_e^{2,\delta}$	o—>	o—>	o—
$t_e^{3,\delta}$	o—>	—>	<—

(a) Arc pattern for $e \bullet\!\!\rightarrow e' \wedge e \rightarrow\!\!\!+ e'$

$\bullet\!\!\rightarrow \wedge \rightarrow\!\!\!\!\%$	$p_{e'}^{In}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	<—	o—	—>
$t_e^{1,\delta}$	<—	<—	—>
$t_e^{2,\delta}$	o—		o—>
$t_e^{3,\delta}$	o—		—

(b) Arc pattern for $e \bullet\!\!\rightarrow e' \wedge e \rightarrow\!\!\!\!\% e'$

Table 5: Two effect relations arc patterns

Case *$e \neq e'$, $e\, r_{k+1}\, e'$ and r_{k+1} is composed of 2 effect relations:*

(i) $@r_{k+1} = \{\rightarrow\!\!\!\!\%, \rightarrow\!\!+\}$: Is equivalent to only mapping the include relation.
(ii) $@r_{k+1} = \{\bullet\!\!\rightarrow, \rightarrow\!\!+\}$: The arc pattern is shown in Table 5a.
(iii) $@r_{k+1} = \{\bullet\!\!\rightarrow, \rightarrow\!\!\!\!\%\}$: The arc pattern is shown in Table 5b.
(iv) $@r_{k+1} = \{\bullet\!\!\rightarrow, \rightarrow\!\!+, \rightarrow\!\!\!\!\%\}$: Equivalent to the case $@r_{k+1} = \{\bullet\!\!\rightarrow, \rightarrow\!\!+\}$.

Note that brown arcs show the changes done to the arc patterns for the response relation from Table 2c.

$\bullet\!\!\leftarrow \wedge \bullet\!\!\rightarrow \wedge \rightarrow\!\!+$	p_e^{In}	$p_{e'}^{Ex}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	—	—	o—>	
$t_e^{1,\delta}$	—	—	—	
$t_e^{2,\delta}$	o—>	o—	—>	<—
$t_e^{3,\delta}$	o—>	o—	o—>	o—
$t_e^{4,\delta}$	o—>	—	—>	<—
$t_e^{5,\delta}$	o—>	—	o—>	o—

(a) Arc pattern for $e' \bullet\!\!\leftarrow e \wedge e \bullet\!\!\rightarrow e' \wedge e \rightarrow\!\!+ e'$

$\bullet\!\!\leftarrow \wedge \rightarrow\!\!\!\!\% \wedge \bullet\!\!\rightarrow$	$p_{e'}^{In}$	$p_{e'}^{Ex}$	$p_{e'}^{Re}$	$p_{e'}^{Rex}$
$t_e^{0,\delta}$	<—	—	o—	—>
$t_e^{1,\delta}$	<—	—	<—	—>
$t_e^{2,\delta}$	o—	o—		o—>
$t_e^{3,\delta}$	o—	o—		—
$t_e^{4,\delta}$	o—	—		o—>
$t_e^{5,\delta}$	o—	—		—

(b) Arc pattern for $e' \bullet\!\!\leftarrow e \wedge e \bullet\!\!\rightarrow e' \wedge e \rightarrow\!\!\!\!\% e'$

Table 6: Two effect relations and a condition relation arc patterns

Case *$e \neq e'$, $e\, r_{k+1}\, e'$ and r_{k+1} is a composed of 2 effect relations and a condition relation:*

(i) $@r_{k+1} = \{\bullet\!\!\leftarrow, \rightarrow\!\!+, \rightarrow\!\!\!\!\%\}$: Equivalent to the case $@r_{k+1} = \{\bullet\!\!\leftarrow, \rightarrow\!\!+\}$.
(ii) $@r_{k+1} = \{\bullet\!\!\leftarrow, \rightarrow\!\!+, \bullet\!\!\rightarrow\}$: The arc pattern is shown in Table 6a.
(iii) $@r_{k+1} = \{\bullet\!\!\leftarrow, \bullet\!\!\rightarrow, \rightarrow\!\!\!\!\%\}$: The arc pattern is shown in Table 6b.

Note that brown arcs show the changes done to the arc patterns for the condition response relation mapping from Table 4c.

This completes the inductive definition of the mapping from DCR graphs to safe Petri Nets with inhibitor arcs, read arcs and pending places as there are no other exceptional cases.

Proof sketch of Thm. 1: exhaustive mapping of exceptional cases(4). We follow the same reasoning as part 2 and 3 of the proof i.e. we take each case sub-point and detail all the possible changes in marking of the DCR Graph in order to show that there is a transition and arc pattern that handles this in the mapped $PNirp$. □

Parts 1 to 4 of the proof of Thm.1 show how the strong bisimilarity property is preserved by each induction step in the definition. We believe the reader should be convinced of how the entire Petri Net is constructed by following the inductive transformation.

What remains to show is that the accepting runs in the two models are the same. This follows easily from the correspondence of markings in Def. 13, which is maintained by the bisimulation relation.

Proposition 1. *For a DCR Graph DCR graph* $G = G_0 = (E, M, R, @, L, l)$ *it holds that an execution sequence* $M \rightarrow M_1 \rightarrow M_2 \rightarrow ..$ *is accepting if and only if* $DPM_G(M) \rightarrow DPM_G(M_1) \rightarrow DPM_G(M_2) \rightarrow ..$ *is accepting.*

5 Pruning and reachability analysis

We report the steps needed to reduce the size of the mapped $PNirp$. This is achieved in two ways: pruning away transitions and places based on the DCR Graph relations and marking; and based on a reachability analysis of the $PNirp$.

5.1 Pruning based on the DCR graph

Creating places. Given a DCR Graph we follow these intuitions when creating places. Only create an:

(i) Included Place for events that are included and may become excluded (have an exclusion relation towards them) and events that are not included and may become included (have an inclusion relation towards them);
(ii) Executed Place for events that have a condition from them;
(iii) Pending Place for events that have a response relation to them and events that are initially pending;
(iv) Pending Excluded Place for events that need both an Included Place and a Pending Place.

Creating event transitions. Given a DCR Graph and a $PNirp$ we follow these intuitions when creating transitions. Only create:

(i) init labelled transitions for events that need an Executed Place;
(ii) pend labelled transitions for events that need a Pending Place.

The pruning is done during the inductive construction of the Petri Net. [6]

5.2 Petri Net reachability analysis

Pruning based on the reachability graph. Our mapping creates dead transitions because it preemptively creates arc patterns for both the marked and unmarked state of a place. Then at each induction step we expand the set of dead transitions, either because we need to copy the dead transition or if we create new transitions and map arcs from a place the dead transition should have an effect on.

Reachability analysis is done on the Petri Net reachability graph which is a labelled transition system where the states are the set of places and the transitions represent the set of transitions fired to move from one state to another. The optimization on the $PNirp$ is done by removing all places and transitions that are not part of the reachability graph.

[6] We direct the reader to the Appendix in our repository `https://github.com/paul-cvp/dcr-to-tapn/blob/master/appendix/Appendix.pdf` to see the simplified arc pattern tables.

Merging places. Finally it is possible to merge places that label the same state in the reachability graph of the Petri Net. The merging also requires us to update the set of pending places accordingly. Notice that in the e-shop example AddOrder and MakePayment have the relation the $r@ = \{\bullet\!\!\rightarrow, \rightarrow\!\!+\}$ and the initial marking of AddOrder is not included and not pending. Therefore we merged the pending and included places of MakePayment. Notice that this would not have been possible if the event AddOrder was initially included.

Definition 16. *(Equivalent places) Let $p, p' \in P$. We say that $p \equiv p'$ if the set of input and output arcs is equal and also the arc type. We define a new place p'' with the merged ids of p and p' and copy their input and output arcs and also the arc type. (Updating Pe) If $(p \in Pe \vee p' \in Pe) \wedge p \equiv p' \iff p'' \in Pe$.*

5.3 Space analysis on the running example

The unoptimized Petri Net of our e-shop has 12 places, 56 transitions and 488 arcs. The DCR analysis pruned one has 5 places, 10 transitions and 49 arcs. Doing just the Petri Net reachability analysis yields 7 places, 6 transitions and 42 arcs. The full optimization, as show in Fig.3 has 3 places, 4 transitions and 12 arcs.

6 Conclusion and future work

We presented a transformation from the Dynamic Condition Response (DCR) graph constraint based process specification language to safe Petri Nets with inhibitor arcs and read arcs, generalized with an acceptance crietria for the modelling of ω-regular liveness properties. We outlined the proof for strong bisimilarity between the transition system for the DCR graph and the transition system for the resulting Petri Net, also preserving the acceptance criteria of finite and infinite executions.

We believe the work in the present paper provides a plethora of research avenues, which we aim to explore in future work. Concretely, we plan to extend the transformation from the core DCR relations to include features of later versions, in particular to cover Timed DCR graphs [12], thereby providing a complete mapping from Timed DCR graphs to safe Timed Arc Petri Nets and also extend the strong bisimulation correspondence to support this case. We plan to evaluate the space complexity of the mapping and the complexity of the resulting models by using the DisCoveR [3] miner to mine DCR Graphs from well-known, real-life, public event logs and map these to their Petri Net counter parts. We also aim to improve the optimization step by using DCR Event-Reachability[18] and by detecting handmade rules such as in [38].

As seen by our running example, the mapping nicely captures concurrency between independent events. This could potentially also be combined with a mapping from Petri Nets to BPMN [9], to provide an output following an ISO standard process notation. Finally we aim to integrate the existing mapping from safe Timed Arc Petri Nets to Timed Automata [34] to provide a link from DCR Graphs to Timed Automata.

References

[1] Wil M.P van der Aalst and Maja Pesic. "DecSerFlow: Towards a Truly Declarative Service Flow Language". In: *Proceedings of Web Services and Formal Methods (WS-FM 2006)*. Ed. by M. Bravetti, M. Nunez, and Gianluigi Zavattaro. Vol. 4184. 2006, pp. 1–23.

[2] T Agerwala. "A complete model for representing the coordination of asynchronous processes". In: (1974). Hopkins Computer Research Report 32.

[3] C.O. Back et al. "DisCoveR: accurate and efficient discovery of declarative process models". In: *Int Journal of Software Tools Technology Transfer* 24 (2022), pp. 563–587.

[4] Paolo Baldan et al. "Functional concurrent semantics for petri nets with read and inhibitor arcs". In: *CONCUR 2000—Concurrency Theory: 11th International Conference University Park, PA, USA, August 22–25, 2000 Proceedings 11*. Springer. 2000, pp. 442–457.

[5] Joakim Byg, Kenneth Yrke Jørgensen, and Jiří Srba. "An Efficient Translation of Timed-Arc Petri Nets to Networks of Timed Automata". In: *Formal Methods and Software Engineering*. Ed. by Karin Breitman and Ana Cavalcanti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 698–716. ISBN: 978-3-642-10373-5.

[6] Joakim Byg, Kenneth Yrke Jørgensen, and Jiří Srba. "TAPAAL: Editor, Simulator and Verifier of Timed-Arc Petri Nets". In: *Automated Technology for Verification and Analysis*. Ed. by Zhiming Liu and Anders P. Ravn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 84–89. ISBN: 978-3-642-04761-9.

[7] Johannes De Smedt et al. "A full R/I-net construct lexicon for declare constraints". In: *Available at SSRN 2572869* (2015).

[8] Søren Debois, Thomas T Hildebrandt, and Tijs Slaats. "Replication, refinement & reachability: complexity in dynamic condition-response graphs". In: *Acta Informatica* 55.6 (2018), pp. 489–520.

[9] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. "Formal semantics and analysis of BPMN process models using Petri nets". In: *Queensland University of Technology, Tech. Rep* (2007), pp. 1–30.

[10] Rik Eshuis et al. "Deriving Consistent GSM Schemas from DCR Graphs". In: *Service-Oriented Computing*. Ed. by Quan Z. Sheng et al. Cham: Springer International Publishing, 2016, pp. 467–482. ISBN: 978-3-319-46295-0.

[11] Olivier Finkel. "On the High Complexity of Petri Nets ω-Languages". In: *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer. 2020, pp. 69–88.

[12] Thomas Hildebrandt et al. "Contracts for cross-organizational workflows as timed Dynamic Condition Response Graphs". In: *The Journal of Logic and Algebraic Programming* 82.5 (2013). Formal Languages and Analysis of Contract-Oriented Software (FLACOS'11), pp. 164–185. ISSN: 1567-8326. DOI: https://doi.org/10.1016/j.jlap.2013.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1567832613000283.

[13] Thomas T. Hildebrandt and Raghava Rao Mukkamala. "Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs". In: *PLACES*. 2010, pp. 59–73.

[14] Thomas T. Hildebrandt et al. "Contracts for cross-organizational workflows as timed Dynamic Condition Response Graphs". In: *Journal of Logic and Algebraic Programming* 82.5-7 (2013), pp. 164–185.

[15] Thomas T. Hildebrandt et al. "Decision Modelling in Timed Dynamic Condition Response Graphs with Data". In: *Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6-10, 2021, Revised Selected Papers*. Ed. by Andrea Marrella and Barbara Weber. Vol. 436. Lecture Notes in Business Information Processing. Springer, 2021, pp. 362–374. DOI: 10.1007/978-3-030-94343-1_28. URL: https://doi.org/10.1007/978-3-030-94343-1%5C_28.

[16] Lom-Messan Hillah et al. "Extending PNML scope: A framework to combine Petri nets types". In: *Transactions on Petri Nets and Other Models of Concurrency VI*. Springer, 2012, pp. 46–70.

[17] Lom-Messan Hillah et al. "PNML Framework: an extendable reference implementation of the Petri Net Markup Language". In: *International Conference on Applications and Theory of Petri Nets*. Springer. 2010, pp. 318–327.

[18] Tróndur Høgnason and Søren Debois. "DCR Event-Reachability via Genetic Algorithms". In: *Business Process Management Workshops*. Ed. by Florian Daniel, Quan Z. Sheng, and Hamid Motahari. Cham: Springer International Publishing, 2019, pp. 301–312. ISBN: 978-3-030-11641-5.

[19] Zhaoxia Hu and Sol M Shatz. "Mapping UML Diagrams to a Petri Net Notation for System Simulation." In: *SEKE*. Citeseer. 2004, pp. 213–219.

[20] Richard Hull et al. "Introducing the guard-stage-milestone approach for specifying business entity lifecycles". In: *Proc. of WS-FM'10*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 1–24. ISBN: 978-3-642-19588-4.

[21] Hugo A. López et al. "The Process Highlighter: From Texts to Declarative Processes and Back". In: *Proceedings of the Dissertation Award and Demonstration, Industrial Track at BPM 2018*. Vol. 2196. 2018. URL: CEUR-WS.org/VOL-2196/.

[22] Fabrizio Maria Maggi et al. "Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata". In: *Business Process Management*. Ed. by Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 132–147. ISBN: 978-3-642-23059-2.

[23] Marco Montali. *Specification and verification of declarative open interaction models: a logic-based approach*. Vol. 56. Springer Science & Business Media, 2010.

[24] Marco Montali et al. "Declarative Specification and Verification of Service Choreographiess". In: *ACM Trans. Web* 4.1 (Jan. 2010). ISSN: 1559-1131. DOI: 10.1145/1658373.1658376. URL: https://doi.org/10.1145/1658373.1658376.

[25] Ugo Montanari and Francesca Rossi. "Contextual nets". In: *Acta Informatica* 32 (1995), pp. 545–596.

[26] Raghava Rao Mukkamala. "A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs". PhD thesis. IT University of Copenhagen, June 2012.

[27] Raghava Rao Mukkamala and Thomas T. Hildebrandt. "From Dynamic Condition Response Structures to Büchi Automata". In: *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*. 2010, pp. 187–190. DOI: 10.1109/TASE.2010.22.

[28] Håkon Norman et al. "Zoom and Enhance: Action Refinement via Subprocesses in Timed Declarative Processes". In: *Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings*. Ed. by Artem Polyvyanyy et al. Vol. 12875. Lecture Notes in Computer Science. Springer, 2021, pp. 161–178. DOI: 10.1007/978-3-030-85469-0_12. URL: https://doi.org/10.1007/978-3-030-85469-0%5C_12.

[29] Amir Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[30] Viara Popova and Marlon Dumas. "From Petri Nets to Guard-Stage-Milestone Models". In: *Business Process Management Workshops*. Ed. by Marcello La Rosa and Pnina Soffer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 340–351. ISBN: 978-3-642-36285-9.

[31] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. "From Declarative Processes to Imperative Models." In: *SIMPDA* 1293 (2014), pp. 162–173.

[32] Ivo Raedts et al. "Transformation of BPMN Models for Behaviour Analysis." In: *MSVVEIS* 2007 (2007), pp. 126–137.

[33] Tijs Slaats. "Flexible Process Notations for Cross-organizational Case Management Systems". PhD thesis. IT University of Copenhagen, Jan. 2015.

[34] Jiří Srba. "Timed-Arc Petri Nets vs. Networks of Timed Automata". In: *Applications and Theory of Petri Nets 2005*. Ed. by Gianfranco Ciardo and Philippe Darondeau. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 385–402. ISBN: 978-3-540-31559-9.

[35] Tony Spiteri Staines. "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets". In: *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*. 2008, pp. 191–200. DOI: 10.1109/ECBS.2008.12.

[36] D. Thapa, S. Dangol, and Gi-Nam Wang. "Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique". In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-*

IAWTIC'06). Vol. 2. 2005, pp. 228–233. DOI: 10.1109/CIMCA.2005.1631473.

[37] Rüdiger Valk. "Infinite behaviour of Petri nets". In: *Theoretical computer science* 25.3 (1983), pp. 311–341.

[38] HMW Verbeek et al. "Reduction rules for reset/inhibitor nets". In: *Journal of Computer and System Sciences* 76.2 (2010), pp. 125–143.

[39] Nianhua Yang et al. "Modeling UML Sequence Diagrams Using Extended Petri Nets". In: *2010 International Conference on Information Science and Applications*. 2010, pp. 1–8. DOI: 10.1109/ICISA.2010.5480384.

[40] Dmitry A. Zaitsev. "Toward the Minimal Universal Petri Net". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.1 (2014), pp. 47–58. DOI: 10.1109/TSMC.2012.2237549.

9.2 Static and Dynamic Techniques for Iterative Test-Driven Modelling of Dynamic Condition Response Graphs

Remark 9.2. *This work is an invited journal paper, currently under review, for Data and Knowledge Engineering journal special issue on Augmented Business Process Management*[1]*. It contains and extends material from two previous conference papers of my co-authors:*

1. *Paper 1 [119]: Slaats T, Debois S, Hildebrandt T. Open to change: A theory for iterative test-driven modelling. InBusiness Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16 2018 (pp. 31-47). Springer International Publishing.*

2. *Paper 2 [35]: Christfort AK, Slaats T. Efficient Optimal Alignment Between Dynamic Condition Response Graphs and Traces. InInternational Conference on Business Process Management 2023 Sep 1 (pp. 3-19). Cham: Springer Nature Switzerland.*

[1] `https://www.sciencedirect.com/journal/data-and-knowledge-engineering/about/call-for-papers#augmented-business-process-management`

Static and Dynamic Techniques for Iterative Test-Driven Modelling of Dynamic Condition Response Graphs

Axel K. F. Christfort[a], Vlad Paul Cosma[a], Søren Debois[b], Thomas T. Hildebrandt[a], Tijs Slaats[a]

[a]*Department of Computer Science, Copenhagen University, Copenhagen Ø, 2100, Denmark*
[b]*DCR Solutions, Copenhagen S, 2300, Denmark*

Abstract

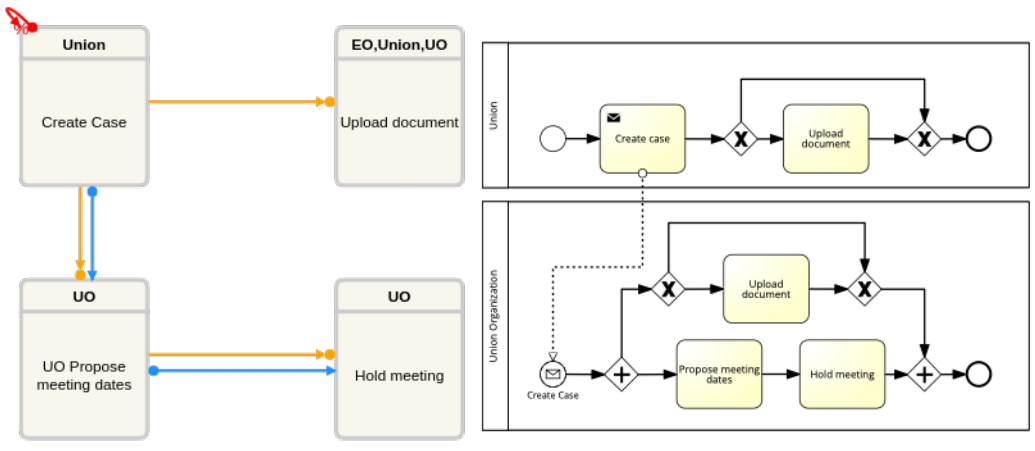
Test-driven declarative process modelling combines process models with test traces and has been introduced as a means to achieve both the flexibility provided by the declarative approach and the comprehensibility of the imperative approach. Open test-driven modelling adds a notion of context to tests, specifying the activities of concern in the model, and has been introduced as a means to support both iterative test-driven modelling, where the model can be extended without having to change all tests, and unit testing, where tests can define desired properties of parts of the process without needing to reason about the details of the whole process. The openness however makes checking a test more demanding, since actions outside the context are allowed at any point in the test execution and therefore many different traces may validate or invalidate an open test. In this paper we combine previously developed static techniques for effective open test-driven modelling for Dynamic Condition Response Graphs with a novel efficient implementation of dynamic checking of open tests based on alignment checking. We illustrate the static techniques on an example based on a real-life cross-organisational case management system and benchmark the dynamic checking on models and tests of varying size.

Keywords: process modelling, declarative, test-driven, iterative, alignment
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Introduction

Declarative process notations [1, 2, 3, 4, 5] and the more traditional imperative process notations [6] represent two complementary approaches to process modelling. Declarative models explicitly define the rules for when activities are allowed and required to be executed, but the possible execution sequences are not explicitly modelled and the reader must by oneself represent the state of the process [7]. In contrast, imperative models do only implicitly represent the rules satisfied by a process, but explicitly define the possible execution sequences, making it more easy for a reader to understand which paths are possible. For this reason, imperative notations are mostly suitable for rigid processes that do not require many different paths and where the underlying rules do not change too often. A declarative process notation on the other hand offers flexibility by default, since an activity can be executed unless it is explicitly forbidden by a rule. Moreover, it is much easier to adapt a declarative process to accommodate changes in the rules, since rules can simply be added or removed from the specification.

As an example, consider the trade union conflict handling process described in [8], which is shown as a Dynamic Condition Response (DCR) Graph [1, 9, 10, 11] in fig. 1a and a BPMN (Business Process Model and Notation) collaboration [6] diagram in fig. 1b, representing respectively a declarative and an imperative process notation supported by industrial design and execution tools.



(a) Declarative DCR Model (b) Imperative BPMN Model

Figure 1: DCR and BPMN models of Union and Employers Organisation conflict handling

The process starts by a trade union (Union) creates a case and send it to the Union Organization (UO), the overarching organization of trade unions. Then the UO must propose tentative meeting dates to the Employer Organization (not shown in the diagrams), after which the UO and the Employer Organization hold a meeting to reach a settlement. At any point after the case was created the Union and the UO can add a document to the case.

While the BPMN diagram shows the execution paths directly, the DCR graph shows the rules: The orange arrow from Create Case to Upload Document with the bullet at the target denotes that Create Case is a *condition* for Upload Document, i.e. Create Case must have been executed at least once before Upload Document can be executed. The blue arrow with the bullet at the source from Create Case to Propose meeting dates denotes that Propose meeting dates is a *response* to Create Case, i.e. after the case is created, the activity Propose meeting dates must eventually be executed. Similarly, the blue arrow from Propose meeting dates to Hold meeting means that a meeting must eventually be hold after meeting dates have been proposed. A unique feature of DCR graphs is that activities dynamically can be included and excluded from the process during execution. Excluded activities cannot be executed and any constraints related to them are ignored. The red arrow with the %-sign at the target from Create Case to itself denotes that the activity Create Case excludes itself when executed, i.e. it can only happen once. The absence of such red arrows for the other three activities means, that they can be repeated any number of times. In contrast, the BPMN process is more rigid, since Propose meeting dates and Hold meeting must happen exactly once and Upload Document at most once for each actor. It would of course be possible to create a BPMN model that offer exactly the same executions as the DCR Graph, essentially by adding more loops to the diagram, but it would then also become more difficult to read.

Inspired by the concept of test-driven development [12, 13] in software engineering, Zugal et al. [7, 14] introduced the concept of *test-driven modelling* (TDM) as a means to support the design and understanding of process models. The idea of TDM is to define a set of desired execution sequences as test cases and then use these test cases to guide the construction of the model with the help of design tools that can automatically check if the model satisfy the test cases. In this way, the user at the same time gets some of the benefits of the declarative and the imperative modelling notations: Both the rules and some of the desired execution sequences are specified explicitly, without having to express all possible sequences.

Some examples of test-cases for the trade union conflict handling process could be the following.

Example 1. In the conflict handling process above, desired execution sequences that can be used as test-cases are the following:

Create Case.Propose meeting dates.Hold meeting (1)

Create Case.Propose meeting dates.Hold meeting.Upload Document (2)

The first sequence represents the simplest (happy) path, where no documents are uploaded. The second sequence represents a path where also a document is uploaded. An example of a test-case which is only satisfied by the DCR Graph model is the following, where meeting dates are proposed twice:

Create Case.Propose meeting dates.Propose meeting dates.Hold meeting (3)

Support for test-driven modelling has been included in the commercial DCR design tools[1] for DCR Graphs [15, 16, 17]. The DCR design tools also allow for the specification of undesired execution sequences, referred to as negative test cases. A negative test case for the running example would be the sequence Create Case.Propose meeting dates, i.e. where the meeting is arranged without it being held. Another example would be the sequence Upload Document, where a document is added to a case before the case is created.

As described in [7, 14], test-driven modelling may also be used during iterative, incremental development of models, where tests are used as guide for the development of the model. However, as pointed out in [18], test cases given as execution sequences then often need to be updated, if the model evolves. Consider for instance the addition of an activity Add metadata to the complaint process and the requirement that the Union must add some meta-data information about the case (at least once) before it is created. This would require the positive test cases to be changed to include at least one Add metadata activity before the Create Case activity. To support incremental model evolution to be used with TDM more seamlessly, so-called *open* tests was introduced in [18]. An open test pairs the test activity trace with a set of activities defining the *context* of relevant activities for that trace, assumed

[1] Available freely for academic use at `dcrsolutions.net`

to include the activities in the trace. All activities outside the context can be carried out at any point when running the test. In this way, test-cases can remain relatively small in size and be defined in a manner that make them robust against safe model extensions, as detailed in Sec. 5. Both positive and negative open tests are allowed in [18]. A positive open test passes if and only if there exists a model trace which is identical to the test case, if one ignores the activities in the trace that are not in the context. A negative test passes if and only if all model traces, projected to activities in the context, are different from the test case.

For instance, we can express the first test case such that it holds both for the original and in the refined model, if we consider the context as {Create Case, Propose meeting dates, Hold meeting}. Similarly the negative test case Upload Document, expressing that a document can not be uploaded before a case has been created, just require the context {Create Case, Upload Document}.

The work in [18] left open the definition of an algorithm for validating tests. Instead, the paper introduced a notion of *transparent* process extensions which preserve positive open tests and a notion of *exclusion-safe* process extensions, which preserve negative open tests. The criteria for the two kinds of extensions are static properties of the graph, which was shown to be verifiable in polynomial time. It was also shown that the problem of validating open tests could be reduced to a model-checking problem.

In the present paper, we show how to define the task of validating open tests as an alignment task, exploiting the recent paper on efficient alignment for DCR graphs [19], and introduce a number of novel search-space pruning techniques that allow to further reduce the run-time cost of checking open tests.

The task of alignment has most notably been used in conformance checking [20] and involves doing synchronous moves, log moves and model moves to show how well a trace, usually originating from an event log, fits to a process model. A synchronous move refers to a move in the model which is matched by a move in the log. A log move refers to a move in the log, without making a move in the model, i.e. skipping over a step in the log. Conversely, a model move refers to making a move in the model without making a move in the log. Each kind of move has a cost as defined by a cost function.

The problem of validating open tests maps neatly to an alignment checking problem, by using the cost function that assigns cost ∞ to log-moves and model-moves inside the context, and cost 0 to synchronous-moves and model-moves outside the context. In this way, the cost function effectively

implements the projection to activities in the context: If an activity is outside the context, it is free to perform this activity in the model without taking a step in the log. If the activity is inside the context, the zero cost of synchronous moves and infinite cost of log and model moves fully disallows deviations between the log and the model.

We structure our paper as follows: In Sec. 2 we briefly discuss related work. In Sec. 3 we first recall the notions of open tests [18] and then recall the definition of Distributed Dynamic Condition Response (DCR) Graphs [1]. We provide an example of iterative test-driven modelling of a DCR Graph inspired by the real case of a trade union process in Sec. 4. In Sec. 5 we then recall the theory of iterative test driven modelling for DCR graphs based on statically verified, safe extensions of DCR graphs [18]. In Sec. 6 we then show how the validation of open tests is mapped to an alignment problem [19]. Sec. 3.1 contain results from experiments. Finally, we conclude and discuss future work in Sec. 7. We provide the complete model of the trade union process used for benchmarking the dynamic techniques in the appendix.

2. Related Work

2.1. Test-Driven Modelling

As described above, test-driven modelling (TDM) was introduced by Zugal et al. in [7, 14] as an application of test-driven development to declarative business processes. Their studies [7] indicate in particular the that simple sequential traces are helpful to domain experts in understanding the underlying declarative models. The present approach generalises that of [7, 14]: We define and study preservation of tests across model updates, alleviating modularity concerns while preserving the core usability benefit of defining tests via traces.

Connections between refinement, testing-equivalence and model-checking was observed in [21]. But where we consider refinements guaranteeing preservation of the projected language, the connection in [21] uses that a refinement of a state based model (Büchi-automaton) satisfies the formula the state based model was derived from. Our approach (and that of [22]) has strong flavours of refinement. Indeed, the iterative development and abstract testing of system models in the present paper is related to the substantial body of work on abstraction and abstract interpretation, e.g., [23, 24, 25]. In particular, an open test can be seen as a test on an abstraction of the system under test, where only actions in the context of the test are visible. In this

respect, the abstraction is given by string projection on free monoids. We leave for future work to study the ramifications of this relationship and the possibilities of exploiting it in employing more involved manipulations than basic extensions of the alphabet in the process of iterative development, such as, e.g., allowing splitting of actions.

The synergy between static analysis and model checking is also being investigated in the context of programming languages and software engineering [26]. In particular there have been proposals for using static analysis to determine test prioritisation [27, 28] when the tests themselves are expensive to run. Our approach takes the novel perspective of analysing the adaptations to a model (or code), instead of analysing the current instance of the model.

Test-driven modelling for declarative models has also been identified as a form of hybrid business process representations [29], falling into the category of representations that combine a declarative model with (more imperative) tool support to aid their understandability [30, 31]. In particular the effect of hybrid approaches towards improving the understandability of DCR Graphs, which are closely related to the test-driven modelling approach presented here, was investigated by Andaloussi et al. in [32, 33]. Other approaches to hybrid business process representations tend to focus on the integration of different process notations, either formal or informal, into a single [34, 35, 36, 37] or hierarchical [38, 39, 40] model.

2.2. Alignment

The term trace alignment [41] was introduced as a pre-processing step to other process mining techniques, where traces where aligned against one another in order to distinguish common from exceptional behaviour.

These days trace alignment primarily refers to the alignment of traces against model behaviour and is seen as a cornerstone of conformance checking [42]. [43] provides a thorough formal treatment of these concepts and defines alignments between traces and models, provides methods for computing trace alignment for petri nets, and shows general applications, namely regarding conformance checking and process enhancement.

Much work has been done on improving the performance of alignment algorithms. Lee et al. [44] recomposes conformance results of sub-problems and present a divide-and-conquer based alignment framework and Reißner et al. [45] propose a method of computing an combining the automatons corresponding to both log and model. [46] proposes solving alignment problems

through optimized SAT-encodings. Other approaches suggest approximating alignments, *i.e.* using simulation [47], using subset selection and edit distance [48], and using Trie data structures [49].

Other efforts have been made in extending alignments, namely computing online alignments of event streams [50], defining partial alignments over partial traces [51], and computing alignments of data aware processes by decomposition [52] and SMT-encoding respectively [53].

Beyond conformance checking, anti-alignments[54] have been introduced to improve precision checking by unveiling model behaviour that deviates from the observed behaviour.

Trace alignment of models has since been extended to cover many process notations, including hybrid [55] and declarative notations, for example through a mapping to the A^* algorithm [56, 57] and a mapping to a planning problem [58].

3. Preliminaries

In this section we first in Sec. 3.1 recall the general notion of open tests as introduced in [18] and then in Sec. 3.2 recall Distributed Dynamic Condition Response Graphs introduced in [8][2].

3.1. Open Tests

The theory of open tests is defined in terms of trace languages. First we define open tests as an activity trace together with a set of activities.

Definition 2 (Open test cases and polarities). *An* open test case (c, Σ) *is a pair consisting of a finite sequence* $c \in \Sigma^*$*, and a set of activities* Σ*. We write* $\mathsf{dom}(c) \subseteq \Sigma$ *for the set of activities appearing in* c*, i.e., when* $c = \langle e_1 \ldots . e_n \rangle$ *we have* $\mathsf{dom}(c) = \{e_1, \ldots, e_n\}$*. For a* polarity ρ *in* $\{+, -\}$*, we refer to* t^+ *as a* positive *open test and* t^- *as a* negative *test.*

Example 3. Consider the set of activities $\Sigma = \{\mathsf{Create}, \mathsf{Propose}, \mathsf{Hold\ Document}\}$, abbreviating respectively "Create case", "Propose meeting dates", "Hold meeting" and "Upload Document". We define an open test case t_0:

$$t_0 = (\langle \mathsf{Create.Propose.Hold.Document} \rangle, \Sigma) \tag{4}$$

[2]We use a slightly modified presentation, that however define the same graphs.

A *positive* test case requires the presence of a trace in the model under test that matches the test, whereas a *negative* test requires the absence of any trace in the model that matches the test. A positive test t_0^+ being *open* means that it can be validated by any traces in a model, over *any* set of activities, whose projection to Σ is exactly $\langle \mathsf{Create.Propose.Hold.Document} \rangle$. For instance, an example of a model trace validating the test is the sequence $\langle \mathsf{Metadata.Create.Propose.Hold.Document} \rangle$, for a model over the activities $\Sigma' = \{\mathsf{Metadata}\} \cup \Sigma$ that in addition to Σ also contains an activity Metadata abbreviating "Add Metadata".

Example 4. Extending our previous example, define a negative open test as follows:

$$t_1^- = (\langle \mathsf{Hold} \rangle, \{\mathsf{Create}, \mathsf{Hold}\})^- \tag{5}$$

Intuitively, the negative open test t_1^- requires that no trace in the model under test, when projected to $\{\mathsf{Create}, \mathsf{Hold}\}$, is exactly Hold. That is, this test models the requirement that one may not Hold a meeting without first Createing a case.

To formalise the validation of open tests we define the system under test as simply a set $L \subseteq \Sigma^*$ of finite sequences over a set of activities Σ.

Definition 5. *A* system $S = (L, \Sigma)$ *is a language* L *of finite sequences over a set of activities* Σ*.*

We can now define under what circumstances positive and negative open tests pass. First we introduce notation.

Notation. Let ϵ denote the empty sequence of activities. Given a sequence s, write s_i for the ith element of s, and $s|_\Sigma$ defined inductively by $\epsilon|_\Sigma = \epsilon$, $(a.s)|_\Sigma = a.(s|_\Sigma)$ if $a \in \Sigma$ and $(a.s)|_\Sigma = s|_\Sigma$ if $a \notin \Sigma$. E.g, if $s = \langle \mathsf{Create.Propose.Hold} \rangle$ is the sequence of test t_0 above, then $s|_{\{\mathsf{Propose},\mathsf{Hold}\}} = \langle \mathsf{Propose.Hold} \rangle$ is the projection of that sequence. We lift projection to sets of sequences point-wise.

Definition 6 (Passing open tests). *Let* $S = (L, \Sigma')$ *be a system and* $t = (c, \Sigma)$ *an open test case. We say that:*

1. S passes *the open test* t^+ *iff there exists* $c' \in L$ *such that* $c'|_\Sigma = c$*.*

2. S passes *the open test* t^- *iff for all* $c' \in L$ *we have* $c'|_\Sigma \neq c$.

S fails *an open test* t^ρ *iff it does not pass it.*

Notice how activities that are not in the context of the open test are ignored when determining if the system passes.

Example 7 (System S, Iteration 1). Consider a system $S = (L, \Sigma)$ with activities $\Sigma = \{\mathsf{Create}, \mathsf{Propose}, \mathsf{Hold}, \mathsf{Document}\}$ and as language L the subset of sequences of Σ^* such that the Hold is always preceded (not necessarily immediately) by Propose, and Create is always succeeded (again not necessarily immediately) by Hold.

Positive tests require existence of a trace that projects to the test case. This system S passes the test t_0^+ for $t_0 = (\langle \mathsf{Create.Propose.Hold.Document} \rangle, \Sigma)$ as defined above, since the sequence $c' = \langle \mathsf{Create.Propose.Hold.Document} \rangle$ in L has $c'|_\Sigma = \langle \mathsf{Create.Propose.Hold.Document} \rangle$.

Negative tests require the absence of any trace that projects to the test case. S also passes the test t_1^- for $t_1 = (\mathsf{Hold}, \{\mathsf{Create}, \mathsf{Hold}\})$ since if there were a $c' \in L$ s.t. $c'|_{\{\mathsf{Create},\mathsf{Hold}\}} = \mathsf{Hold}$ that would contradict that Create should always appear before any occurrence of Hold.

Finally, consider the following positive test.

$$t_2^+ = (\mathsf{Create}, \{\mathsf{Create}, \mathsf{Hold}\})^+$$

The System S fails this test t_2^+, because every sequence in L that contains Create will by definition also have a subsequent Hold, which would then appear in the projection.

We note that a test either passes or fails for a particular system, never both; and that positive and negative tests are dual: t^+ passes iff t^- fails and vice versa.

Lemma 8. *Let* $S = (L, \Sigma)$ *be a system and* t *a test case. Then either (a)* S *passes* t^+ *and fails* t^-*; or (b)* S *fails* t^+ *and* S *passes* t^-.

Example 9 (Iteration 2, Test preservation). We extend our model of Example 7 with the additional requirement that the union must add metadata before creating a case. To this end, we refine our system (L, Σ) to a system $S' = (L', \Sigma' = \Sigma \cup \{\mathsf{Metadata}\})$ where Metadata abbreviates "Add Metadata", and L' is the language over Σ'^* that satisfies the original rules of Example 7 and in addition that Create is always preceded by Metadata.

The explicit context ensures that the tests t_0^+, t_1^-, t_2^+ defined in the previous iteration remain meaningful. The system S' no longer has a trace

$$\langle \mathsf{Create.Propose.Hold.Document} \rangle$$

because Metadata is missing. Nonetheless, S' *still* passes the test t_0^+, because S' *does* have the trace:

$$c' = \langle \mathsf{Metadata.Create.Propose.Hold.Document} \rangle \in L'$$

and the projection $c'|_\Sigma = \langle \mathsf{Create.Propose.Hold.Document} \rangle$ shows that t_0^+ passes S'.

Similarly, S' still passes the test t_1^- since for any $c' \in L'$, if $c'|_\Sigma = \langle \mathsf{Hold} \rangle$ then $c' = \langle c_0.\mathsf{Hold}.c_1 \rangle$ for some $c_0, c_1 \in \Sigma' \backslash \Sigma$, but that contradicts the requirement that Create must appear before any occurrence of Hold.

3.2. Distributed Dynamic Condition Response Graphs

In this section we recall the declarative language of Distributed *Dynamic Condition Response* (DCR) Graphs [1] originally introduced as a formal language for event-based workflows derived as a genernalisation of event structures [59]. The DCR language is a similar to DECLARE [4, 5] in that it is a graph based language for the declaration of temporal constraints between activities. Different from DECLARE is that a DCR graph also represent the run-time state of a process using a so-called marking of activities. This makes it possible to provide an operational semantics of DCR graphs like for Petri Nets, by defining when an activity is enabled in a marking and how the execution of an enabled activity updates the marking.

Formally, we define DCR graphs as attributed directed graphs [3]

Definition 10. *A* DCR graph G *is given by a tuple* $(E, M, R, @, L, l)$ *where*

(i) E *is a finite set of* events

(ii) $M = (Ex, Re, In) \in \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$ *is the* marking

(iii) $R \subseteq E \times E$ *is the set of* relations *between events*

[3] We deviate slightly from the original presentation in[DCR Graph [1]] to be consistent with later articles and defines slightly more general graph structures.

(iv) $@ : R \to \mathcal{P}_{ne}(\{\to\bullet, \bullet\to, \to\diamond, \to+, \to\%\})$ *is the* relation type assignment

(v) L *is the set of action labels*

(vi) $l : E \to L$ *is the* labelling function *assigning an action label to each event*

The marking $M = (Ex, Re, In)$ *describes the state of an event* e *in the following way. If* e *has been* executed *at least once then* $e \in Ex$*. If* e *is* pending *(i.e. it must eventually be executed) then* $e \in Re$*. If* e *is* included *(i.e. it is currently relevant) then* $e \in In$*.*

The relation type assignment assigns one or more relation types to each edge. If $\to\bullet \in @(e, e')$ *we say that there is a* condition *from* e *to* e'*. If* $\bullet\to \in @(e, e')$ *we say that there is a* response *from* e *to* e'*. If* $\to\diamond \in @(e, e')$ *we say that there is a* milestone *from* e *to* e'*. If* $\to+ \in @(e, e')$ *we say that there is an* inclusion *from* e *to* e'*. If* $\to\% \in @(e, e')$ *we say that there is an* exclusion *from* e *to* e'*. We will often write* $e\mathcal{R}e'$ *or* $e\mathcal{R}e' \in R$ *for* $\mathcal{R} \in @(e, e')$*.*

We assume for simplicity in the the development of the static techniques for itereative test-driven modelling that the set of events and activities are the same and the labelling function is the identity. We will therefor also often use the words activity and activities for event and events.

We introduce the notion of Distributed DCR Graph from [8] to facilitate the presentation of the running example, which is an example of a process involving three different roles. The development of the theory extends trivially to include observance of roles, but will for simplicity of presentation ignore the roles in the following.

Definition 11. *A* Distributed DCR graph *is a tuple* $(E, M, R, @, L, l, \mathsf{by}, \mathsf{RI})$

- $(E, M, R, @, L, l)$ *is a DCR Graph*
- L *is the set of* activities
- $\ell : \mathsf{E} \to L$ *is the* labelling function*, assigning activities to events*
- RI *is a set of roles*
- $\mathsf{by} : L \to \mathcal{P}(\mathsf{RI})$ *is the* role assignment function*, assigning zero or more roles to activities.*

We can now explain the DCR graph model of the Union process formally.

Example 12 (DCR graph of the trade union process). The distributed DCR graph for the Union process shown in Figure 1a uses the response, condition and exclude relations. The condition from Create Case to Upload Document models the requirement that a case must be created before documents can be added to it. The response and condition from Create Case to Propose meeting dates models respectively that meeting dates must eventually be proposed after a case has been created and that a case must be created before meeting dates can be proposed. Similarly for the condition and response relations from Propose meeting dates to Hold meeting. The marking of the graph is $(\emptyset, \emptyset, \{\mathsf{Create\ Case}, \mathsf{Propose\ meeting\ dates}, \mathsf{Upload\ Document}, \mathsf{Hold\ meeting}\})$, i.e. no activities have been executed, no activities are yet pending responses and all activities are included. The roles are $\mathsf{Rl} = \{\mathsf{uo}, \mathsf{union}, \mathsf{eo}\}$, i.e. union organisation, union and employer organisation respectively. The role assignment is given by $\mathsf{by}(\mathsf{Create\ Case}) = \{\mathsf{union}\}$, $\mathsf{by}(\mathsf{Upload\ Document}) = \mathsf{Rl}$, $\mathsf{by}(\mathsf{Propose\ meeting\ dates}) = \{\mathsf{uo}\}$ and $\mathsf{by}(\mathsf{Hold\ meeting}) = \{\mathsf{uo}\}$.

The enabledness of an event can be determined by looking at the marking of events immediately related to the activity by an incoming condition or milestone relation the graph. To define enabledness formally we will use the following notation.

Notation. When G is a DCR graph, we write, e.g., $\mathsf{E}(G)$ for the set of activities of G, $\mathsf{Ex}(G)$ for the executed activities in the marking of G, etc. In particular, we write $\mathsf{M}(e)$ for the triple of boolean values $(e \in \mathsf{Ex}, e \in \mathsf{Re}, e \in \mathsf{In})$. We write $(\rightarrow\!\bullet\, e)$ for the set $\{e' \in \mathsf{E} \mid e' \rightarrow\!\bullet\ e\}$, write $(e \bullet\!\rightarrow)$ for the set $\{e' \in \mathsf{E} \mid e \bullet\!\rightarrow e'\}$ and similarly for $(e \rightarrow\!+)$, $(e \rightarrow\!\%)$ and $(\rightarrow\!\diamond e)$.

Definition 13 (Enabled activities [1]). *Let $G = (E, \mathsf{M}, R, @, L, l)$ be a DCR graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. An event $e \in \mathsf{E}$ is* enabled*, written $e \in \mathsf{enabled}(G)$, iff (a) $e \in \mathsf{In}$ and (b) $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$ and (c) $(\mathsf{Re} \cap \mathsf{In}) \cap (\rightarrow\!\diamond e) = \emptyset$.*

That is, enabled activities (a) are included, (b) their included conditions have already been executed, and (c) have no pending included milestones.

The effect of executing an activity only changes the marking of the executed activity and activities related to it by outgoing response, inclusion or exclusion edge from the executed activity.

Definition 14. *Let G be a DCR graph with marking $\mathsf{M} = (Ex, Re, In)$. The effect of executing an enabled event e is the marking $\mathsf{effect}_G(\mathsf{M}, e) = (Ex', Re', In')$ where*

$$\begin{aligned} Ex' =& Ex \cup \{e\} \\ Re' =& (Re \setminus \{e\}) \cup (e \bullet\!\!\rightarrow) \\ In' =& (In \setminus (e \rightarrow\!\!\%) \cup (e \rightarrow\!\!+) \end{aligned}$$

We write $\mathsf{execute}(G, e)$ for the DCR Graph obtained by replacing the marking of G with $\mathsf{effect}_G(\mathsf{M}, e)$. We write $G \rightarrow_e G'$ for $G' = \mathsf{execute}(G, e)$. For a sequence of events $\phi = \langle e_1, ..., e_n \rangle$, we write $G \rightarrow^*_\phi G_n$ as shorthand for $G \rightarrow_{e_1} G_1 ... \rightarrow_{e_n} G_n$.

Definition 15 (Accepting). *Let G be a DCR graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. We say that G is* accepting*, written $\mathsf{accepting}(G)$, iff $\mathsf{In} \cap \mathsf{Re} = \emptyset$.*

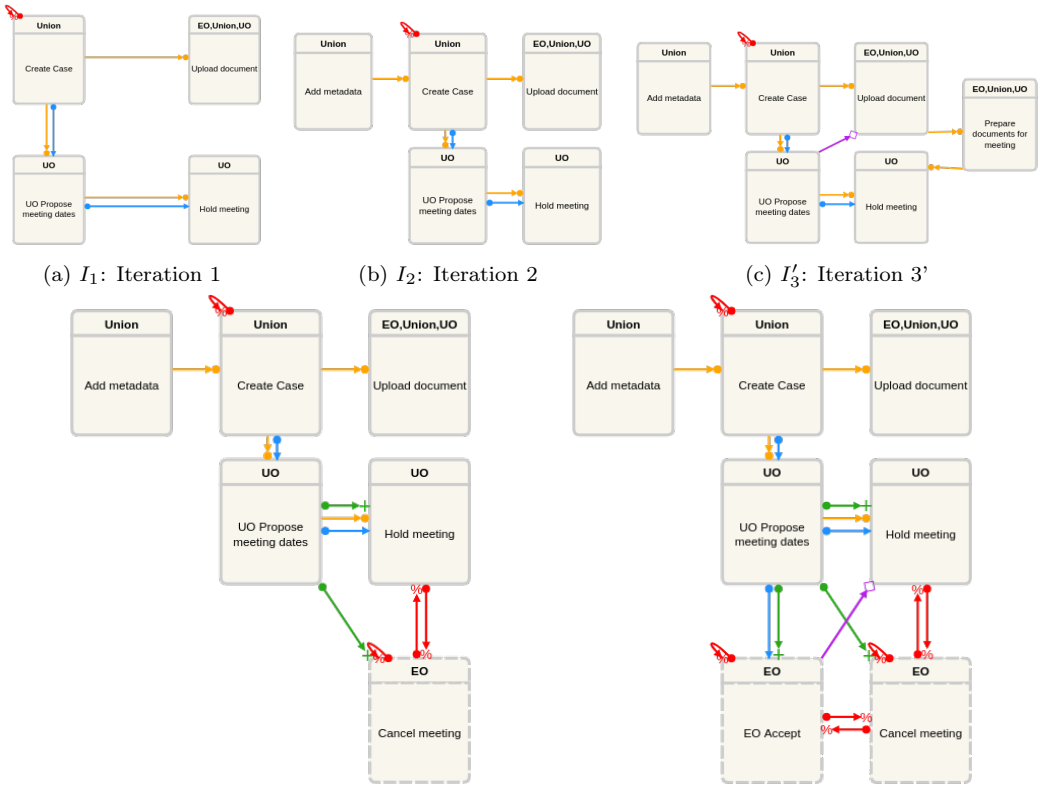
Definition 16 (Runs). *A sequence of events ϕ is a* run *of a DCR Graph iff $G \rightarrow^*_\phi G'$. It is an* accepting run *iff $\mathsf{accepting}(G')$. . The* language $\mathsf{lang}(G_0)$ *of G_0 is then the set of all such accepting runs. Write $\hat{G}$ for the corresponding system $\hat{G} = (\mathsf{lang}(G), \mathsf{E})$ (viz. Definition 5). When no confusion is possible, we denote by simply G both a DCR graph and its corresponding system $\hat{G}$.*

Notation. We lift the labelling function to also work on runs, giving us the shorthand $\ell(\phi) = \langle \ell(e_1) ... \ell(e_n) \rangle$ for a run $\phi = \langle e_1, ..., e_n \rangle$.

4. Example of Iterative Open Test-Driven Modelling

We will consider a series of extensions of the running example shown in Fig. 2 to exemplify an iterative development of a distributed DCR Graph model. Fig. 2a shows the initial model shown in Fig. 1, which we refer to as the iteration 1 model. The process we eventually develop conforms to the currently running process at the Union Organization. The process exists to ensure that the Union, UO and EO handle complaints from union members following the agreed on protocol.

Example 17 (DCR Iteration 2). Following Ex. 9, we extend the iteration 1 model of Fig. 2a to the iteration 2 model in Fig. 2b. We model the new requirement that metadata must be provided before creating a case by adding a new activity Metadata and a condition relation from Metadata to Create.



(a) I_1: Iteration 1 (b) I_2: Iteration 2 (c) I_3': Iteration 3'

(d) I_3: Iteration 3 (e) I_4: Iteration 4

Figure 2: DCR Graph models of the case handling Examples 3–20.

Example 18 (Non-preservation of non-open tests). We emphasize that if we interpret the trace $s = \langle \mathsf{Create.Propose.Hold.Document} \rangle$ underlying the test t_0^+ as a test in the sense of [7, 14], that test is *not* preserved when we extend the system from (L, Σ) to (L', Σ'): The original system L has the behaviour s, but the extension L' does not.

Example 19 (Iteration 2, Additional tests). We add the following additional tests for the new requirements of Iteration 2.

$$t_3^- = (\langle \mathsf{Create}, \mathsf{Propose}, \mathsf{Hold}, \mathsf{Document} \rangle, \Sigma')^-$$

Note that the trace of t_3 is *the same* as the original test t_0; the two tests differ only in their context. This new test says that in a context where we know about the Metadata activity, omitting it is not allowed.

The use of a context in open tests means that changes to a model will as in the above cases often preserve open test outcomes, obviating the need to re-check tests after the change. This is however not generally the case, as illustrated by the following example.

Example 20 (Iteration 3). The meeting may be cancelled, e.g. if the union decides to drop the case, therefore no meeting will be held. We model this by adding an activity "Cancel meeting", abbreviated as Cancel, to the set of activities Σ' and constrain the language L' accordingly. Now the system will pass the test $t_2^+ = (\mathsf{Create}, \{\mathsf{Create}, \mathsf{Hold}\})^+$ defined above, because it has a trace that contains Create but no Hold: the sequence in which the Union Organization proposes a meeting date and subsequently cancels it.

Example 21 (DCR Iteration 3). Following Example 20, we extend the iteration 2 model of Figure 2b to the iteration 3 model in Figure 2d. To model the cancellation of a proposed meeting, we add the activity Cancel and exclude-relations between Cancel and Hold. This models the choice between those two activities: Once one is executed, the other is excluded and no longer present in the model. To model subsequent meeting proposals, we add an include relation from Propose to Cancel and Hold: if the union proposes new meeting dates, the activities Cancel and Hold become included once again, re-enabling the decision to hold or cancel a meeting.

Example 22 (DCR Iteration 4). We now add a new activity "EO Accept" to the model of Figure 2d in Figure 2e. Using response and include relation pair from Propose to Accept we model that after UO proposes a meeting date EO must Accept the meeting date; and by adding a milestone from Accept to Hold we model that a meeting may not be held while we are waiting for EO to Accept the meeting date. Finally, we add exclude-relations between Accept and Cancel to model the choice EO has at this step in the process.

We proceed in the next section to recall from [18] the static techniques for iterative open test-driven modelling of DCR Graphs.

5. Static Techniques for Iterative Open Test-Driven Modelling of DCR Graphs

The following Proposition from [18] gives general conditions for when outcomes of positive (resp. negative) tests for a system S are preserved when the system is changed to a new system S'.

Proposition 23 (Preservation of tests[18]). *Let $S = (L, \Sigma)$ and $S' = (L', \Sigma')$ be systems, and let $t = (c_t, \Sigma_t)$ be a test case. Assume that $\Sigma' \cap \Sigma_t \subseteq \Sigma' \cap \Sigma$. Then:*

1. *If $L'|_\Sigma \supseteq L$ and S passes t^+, then so does S'.*
2. *If $L'|_\Sigma \subseteq L$ and S passes t^-, then so does S'.*

In words, the assumption $\Sigma' \cap \Sigma_t \subseteq \Sigma' \cap \Sigma$ states that the changed system S' does not introduce activities appearing in the context of the test that did not already appear in the original system S. Condition 1 (resp. 2) expresses that positive (resp. negative) tests are preserved if the language of the original system S is included in (resp. including) the language of the changed system S' projected to the activities in the original system. Now, if one can find static properties of changes to process models for a particular notation that implies the conditions of Proposition 23 then these properties can be checked instead of relying on model-checking to infer preservation of tests. We identified such static properties for the Dynamic Condition Response (DCR) graphs in [18]

We consider the situation that a graph G' extends a graph G by adding activities and relations. Recall the notation $\mathsf{M}(e) = (e \in \mathsf{Ex}, e \in \mathsf{Re}, e \in \mathsf{In})$.

Definition 24 (Extensions). *Let*

$$G = (E, \mathsf{M}, R, @, L, l) \text{ and } G' = (E', \mathsf{M}', R', @', L', l')$$

be DCR graphs. We say that G' statically extends *G and write $G \sqsubseteq G'$ iff $\mathsf{E} \subseteq \mathsf{E}'$, $\mathsf{R} \subseteq \mathsf{R}'$, $@ \subseteq @'$, $L \subseteq L'$ and $\ell \subseteq \ell'$. If also $e \in \mathsf{E}$ implies $\mathsf{M}(e) = \mathsf{M}'(e)$, we say that G'* dynamically extends *G and write $G \preceq G'$.*

Our main analysis technique will be the application of Proposition 23. To this end, we need ways to establish the preconditions of that Theorem, that is:

$$\mathsf{lang}(G')|_\mathsf{E} \supseteq \mathsf{lang}(G) \qquad (\dagger)$$

$$\mathsf{lang}(G')|_\mathsf{E} \subseteq \mathsf{lang}(G) \qquad (\ddagger)$$

Example 25. Consider the graphs I_1 of Figure 2a and I_2 of Figure 2b. Clearly $I_1 \sqsubseteq I_2$ since I_2 contains all the activities and relations of I_1. Moreover, since the markings of I_1 and I_2 agree, also $I_1 \preceq I_2$. Similarly, $I_3 \sqsubseteq I_4$, where I_3 and I_4 are the graphs of Figures 2d and 2e. On the other hand, neither $I_2 \sqsubseteq I_3$ nor $I_2 \sqsubseteq I_4$, since the former graphs have relations (e.g., the exclusions and the milestone) not in the latter.

We note that DCR activity execution preserves static extensions, i.e. if $G \sqsubseteq G'$ and an activity e is enabled in both G and G' then $G_1 \sqsubseteq G'_1$, if G_1 and G'_1 are the results of executing e in G and G' respectively. Dynamic extension is generally *not* preserved by execution, because an execution might make markings between the original and extended graph differ *on the original activities*, e.g., if G' adds an exclusion, inclusion or response constraint between activities of E.

5.1. Positive tests

We first establish a syntactic condition for a modification of a DCR graph to preserve positive tests. The condition will be, roughly, that the only new relations are either (a) between new activities, or (b) conditions or milestones from new to old activities. For the latter, we will need to be sure we can find a way to execute enough new activities to satisfy such conditions and milestones. To this end, we introduce the notion of dependency graph, inspired by [60].

Definition 26 (Dependency graph). *Let $G = (E, \mathsf{M}, R, @, L, l)$ be a DCR graph, and let $e, f, g \in \mathsf{E}$ be activities of G. Write $e \to f$ whenever $e \to\bullet f \in \mathsf{R}$ or $e \to\diamond f \in \mathsf{R}$ and $\to^*$ for the transitive closure. The* dependency graph *$D(G, e)$ for e is the directed graph which has nodes $\{f \mid g \to^* e \wedge g \bullet\to^* f\}$ and an edge from node f to node g iff $f \to g$ or $f \bullet\to g$ in G.*

With the notion of dependency graph, we can define the notion of "safe" activities, intuitively those that can be relied upon to be executed without having undue side effects on a given (other) set of nodes X. The principle underlying this definition is inspired by the notion of dependable activity from [60].

Definition 27 (Safety). *Let $G = (E, \mathsf{M}, R, @, L, l)$ be a DCR graph, let $e \in \mathsf{E}$ be an activity of G, and let $X \subseteq \mathsf{E}$ be a subset of the activities of G. We say that e is* safe *for X iff*

1. *$D(G, e)$ is acyclic,*
2. *no $f \in D(G, e)$ has an include, exclude, or response relation to any $x \in X$.*
3. *for any $f \in D(G, e)$, if f has a condition or milestone to some $f' \in \mathsf{E}$, then f' is reachable from f in $D(G, e)$.*

The notion of safe activity really captures activities that can reliably (without side effects) be executed if they are conditions or milestones for other activities. We use this to define a notion of transparent process extensions: a process extension which we shall see preserves positive tests.

Definition 28 (Transparent). *Let*

$$G = (E, \mathsf{M}, R, @, L, l) \text{ and } G' = (E', \mathsf{M}', R', @', L', l')$$

be DCR graphs with $G \sqsubseteq G'$. *We say that* G' *is* transparent *for* G *iff for all* $e, f \in \mathsf{E}$ *and* $e', f' \in \mathsf{E}'$ *we have:*

1. *if* $e'\mathcal{R}f' \in \mathsf{R}'$ *for* $\mathcal{R} \in \{\rightarrow\bullet, \rightarrow\diamond\}$ *then* either $e'\mathcal{R}f' \in \mathsf{R}$ or *(a)* $e' \notin \mathsf{E}$, *(b)* e' *is safe for* E, *and (c)* $\mathsf{E}(D(G', e')) \subseteq \mathsf{E}' \setminus \mathsf{E}$,
2. *for* $\mathcal{R} \in \{\rightarrow+, \rightarrow\%, \bullet\rightarrow\}$ *we have* $e\mathcal{R}f \in \mathsf{R}'$ *iff* $e\mathcal{R}f \in \mathsf{R}$.
3. *for* $\mathcal{R} \in \{\rightarrow+, \bullet\rightarrow\}$ *we have if* $e\mathcal{R}e' \in \mathsf{R}'$ *or* $e' \in \mathsf{Re}(G')$ *then* $e' \in \mathsf{E}$

We rephrase these conditions more intuitively. Call an activity $e \in \mathsf{E}$ an *old* activity, and an activity $e' \in \mathsf{E}' \setminus \mathsf{E}$ a *new* activity. The first item then says that we can never add conditions or milestones from old activities and only add a condition or milestone to an old activity when the new activity is safe, that is, we can rely on being able to discharge that milestone or condition. The second item says that we cannot add exclusions, inclusions or responses between old activities. The third says that we also cannot add inclusions or responses from old to new activities, or add a new activity which is initially pending in the marking, which could cause the new graph to be less accepting than the old. Inclusions, exclusions and responses may be added from a new to an old activity; the interplay of condition 1 of Definition 28 and condition 2 of Definition 27 then implies that this can only happen if the new activity is not in the dependency graph of any old activity. The reason is, that such constraints can be *vacuously* satisfied since the new activity at the source of the constraint is *irrelevant* with respect to passing any of the positive tests.

Example 29. It is instructive to see how violations of transparency may lead to non-preservation of positive tests. An extension such as the one from I_2 to I_3' consisting of the addition of the new activity "Prepare documents for meeting" along with the relations Document $\rightarrow\bullet$ Prepare $\rightarrow\bullet$ Hold, Propose $\rightarrow\diamond$ Document which break the positive test t_0^+. In I_3' we now force

Document to happen before the first Hold and any time after Propose. These changes violate item 1 from Definition 28.

Also consider the change from I_2 to I_3. Although we have a new inclusion relation Propose $\rightarrow+$ Hold between two old activities, it does not violate transparency for this specific marking. Consider however a different marking where Hold $\notin$ In is not included. In this case the new include relation would break any positive tests. This violation breaks items 2 from definition 28.

Theorem 30. *Let $G \preceq G'$ with G' transparent for G, and let $t^+ = (c_t, \Sigma_t)^+$ be a positive test with $\Sigma_t \subseteq \mathsf{E}$. If G passes t then so does G'.*

Example 31 (Preservation). Consider the change from the graph I_1 of Figure 2a to the graph I_2 of Figure 2b: We have added the activity Metadata and the condition Metadata $\rightarrow\bullet$ Create. In this case, I_2 is transparent for I_1: The new activity Metadata satisfies Definition 28 part (1c): even though a new condition dependency is added for Create, the dependency graph for the new Create remains acyclic. By Theorem 30, it follows that *any* positive test whose context is contained in {Create, Propose, Hold, Document} will pass I_2 if it passes I_1. In particular, we saw in Example 7 that I_1 passes the test t_0^+, so necessarily also I_2 passes t_0^+.

Now consider the changes as observed when extending I_3 to I_4. The new activity is Accept. Because we add a milestone relations between a new and an old activity we have to check item 1 from Definition 28. Therefore we ask (a) is Accept a new activity? Yes; (b) is Accept safe? Yes (we construct the dependency graph D(I_4, Accept) and apply Definition 27); (c) Are the activities in the dependency graph $D(I_4, \mathsf{Accept})$ a subset of only new activities? Yes (the dependency graph is the empty graph). We can therefore conclude that the extension I_4 is a transparent extension of I_3.

5.2. Negative tests

For negative tests we must establish the inclusion (‡) stated after Definition 24. This inclusion was investigated previously in [61, 22], with the aim of establishing more general refinement of DCR graphs. Definition 24 is a special case of refinement by merging, investigated in the above papers. Hence, we use the sufficient condition for such a merge to be a refinement from [22] to establish a sufficient condition, *exclusion-safety* for an extension to preserves negative tests.

Definition 32 (Exclusion-safe). *Suppose* $G = (E, \mathsf{M}, R, @, L, l)$ *and* $G' = (E', \mathsf{M}', R', @', L', l')$ *are DCR graphs and that* G' *dynamically extends* G*. We say that* G' *is* exclusion-safe *for* G *iff for all* $e \in \mathsf{E}$ *and* $e' \in \mathsf{E}'$ *we have that:*

1. *if* $e' \rightarrow\% \; e \in \mathsf{R}'$ *then* $e' \rightarrow\% \; e \in \mathsf{R}$.
2. *if* $e' \rightarrow+ \; e \in \mathsf{R}'$ *then* $e' \rightarrow+ \; e \in \mathsf{R}$.

Theorem 33. *Suppose* $G \preceq G'$ *are DCR graphs with* G' *exclusion-safe for* G*, and suppose* $t^- = (c_t, \Sigma_t)^-$ *is a negative test with with* $\Sigma_t \subseteq E$*. If* G *passes* t *then so does* G'*.*

Example 34 (Application). Consider again the change from I_1 to I_2 in Figure 2a and 2b. Since neither contains inclusions or exclusions, clearly I_2 is exclusion-safe for I_1. By Theorem 33 it follows that any negative test whose context is contained in $\{\mathsf{Create}, \mathsf{Propose}, \mathsf{Hold}, \mathsf{Document}\}$ which passes I_1 will also pass I_2. In particular, the negative test $t_1^- = (\langle \mathsf{Hold} \rangle, \{\mathsf{Create}, \mathsf{Hold}\})^-$ of Example 7 passes I_1, so by Theorem 33 it passes also I_2.

Example 35 (Non-application). The changes from I_2 to I_3 (Figures 2b and 2d), where amongst other changes we have added an activity Cancel and a relation $\mathsf{Cancel} \rightarrow\% \; \mathsf{Hold}$ violate exclusion-safety.

In this case, we can find a negative test that passes I_2 but not I_3: $t_2^- = \langle \mathsf{Create} \rangle, \{\mathsf{Create}, \mathsf{Hold}\}^-$. It passes both I_1 and I_2, because in both of these, Create leaves Propose pending and which subsequently leaves Hold pending, whence one needs to execute both Propose and Hold to get a trace of the process. But in I_3 (and I_4), we can use Cancel to exclude the pending Hold. So I_3 has a trace $\langle \mathsf{Create}, \mathsf{Propose}, \mathsf{Cancel} \rangle$, and the projection of this trace to the context $\{\mathsf{Create}, \mathsf{Hold}\}$ of our test is the string $\langle \mathsf{Create} \rangle$: The test fails in I_3 (and I_4).

The prerequisites of both Theorem 30 and 33 are effectively computable.

Theorem 36. *Let* $G \preceq G'$ *be DCR graphs. It is decidable in time polynomial in the maximum size of* G, G' *whether (1)* G' *is exclusion-safe for* G *and (2)* G' *is transparent for* G*.*

Should a model update fail to satisfy the prerequisites for Theorem36, we proceed to "re-run" tests. We now show how one can use the theory of alignments for DCR Graphs [19] instead of the model-checking technique proposed in [18]. Using alignments has the added benefit of alleviating the burden of

mapping the DCR Graph into a Büchi-automaton for model checking tools and avoids the exponential space blowout of such a mapping [62]. Using alignments also allow us to implement a way of pruning the search-space based on the static constraints of the DCR graph, which will be introduced in the coming section.

6. Applying Alignment to Test Driven Modeling

We first recall the definition of alignments between traces and DCR graphs as defined in [19].

Definition 37 (Alignment and complete alignment). *Let L be a set of activity names, $\sigma \in L^*$ a trace, and $G = (E, \mathsf{M}, R, @, L, l)$ a DCR graph.*

A pair $(l, e) \in (L_{\gg} \times E_{\gg}) \setminus \{\gg, \gg\}$ is

- *a* move *in log if $l \in L$ and $e = \gg$;*
- *a* move *in model if $l = \gg$ and $e \in E$;*
- *a synchronous* move *if $l \in L$, $e \in E$ and $l = \ell(e)$.*

Let $L_A = (L_{\gg} \times E_{\gg}) \setminus \{\gg, \gg\}$ be the set of legal moves. The alignment of a trace σ and run $\phi \in E^*$ is a sequence $\gamma = \langle (l_1, e_1)...(l_n, e_n) \rangle \in L_A^*$, s.t. $\gamma_\sigma = \langle l_1...l_n \rangle$ and $\gamma_\phi = \langle e_1...e_n \rangle$ (ignoring $\gg$). We say that γ is a complete alignment *of σ and G iff $\gamma_\sigma = \sigma$ and γ_ϕ is an accepting run of G.*

Definition 38 (Optimal alignment). *Let σ be a trace and G a DCR graph. Let $\Gamma_{(\sigma,G)}$ be the set of all complete alignments of σ and G. Given a cost function $\mathcal{K}$, we now define an alignment $\gamma \in \Gamma_{(\sigma,G)}$ to be* optimal, *iff $\forall \gamma' \in \Gamma_{(\sigma,G)}. \mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$.*

We now make the novel observation that the model checking task of Test Driven Modelling maps neatly to an alignment checking problem when using the cost function that assigns cost ∞ to log-moves and model-moves inside the context, and cost 0 to synchronous-moves and model-moves outside the context. We define the cost function here specifically for DCR Graphs but note that it straightforwardly generalizes to any modelling notation by removing the labelling function.

Definition 39. *For a DCR graph $G = (E, \mathsf{M}, R, @, L, l)$ and the context of a test case Σ, we define cost function $\mathcal{K}^{\Sigma}_{G}$ as follows:*

$$\mathcal{K}^{\Sigma}_{G}(l, e) = \begin{cases} 0 & \textit{when } \ell(e) \in \Sigma \wedge l = \ell(e) \\ 0 & \textit{when } \ell(e) \notin \Sigma \wedge l = \gg \\ \infty & \textit{otherwise} \end{cases}$$

We can now go about proving this property with an auxilliary lemma, tying together alignment cost and the underlying runs projection onto a context.

Lemma 40. *Let G be a DCR graph and $t = (c, \Sigma)$ a test case. For any alignment γ of c and G under $\mathcal{K}^{\Sigma}_{G}$, we have that $\mathcal{K}^{\Sigma}_{G}(\gamma) = 0$ iff $\gamma_{\phi}|_{\Sigma} = c$.*

Proof (sketch). By induction on c. It is sufficient to consider a single step l, e of the alignment.

($\Longrightarrow$) Assume that $\mathcal{K}^{\Sigma}_{G}(\gamma) = 0$. By definition of $\mathcal{K}^{\Sigma}_{G}$, every move $(l, e) \in \gamma$ has either $l = \ell(e) \in \Sigma$ or $l = \gg \wedge \ell(e) \notin \Sigma$. If $\ell(e) \in \Sigma$ then by definition of $\mathcal{K}^{\Sigma}_{G}$ we have $l = \ell(e) \in \Sigma$ and so $\ell(e)|_{\Sigma} = \ell(e) = l$. If instead $\ell(e) \notin \Sigma$ then $l = \gg$ and $\ell(e)|_{\Sigma} = \cdot$.

($\Longleftarrow$) Assume instead that $\gamma_{\phi}|_{\Sigma} = c$. Consider some step l, e of γ. If $\ell(e) \in \Sigma$ then $\ell(e)|_{\Sigma} = \ell(e)$. By definition also $l \in \Sigma$ but then by assumption $l = \ell(e) = \ell(e)|_{\Sigma}$ and the step has cost zero. If instead $\ell(e) \notin \Sigma$ then $\ell(e)|_{\Sigma} = \cdot$. But then clearly $l = \gg$, and again the step has cost zero. □

With this lemma, we can now state and prove that for a test case $t = (c, \Sigma)$, the cost of alignment between c and a graph G using $\mathcal{K}^{\Sigma}_{G}$ matches exactly with whether or not G passes the test t.

Theorem 41. *Let $G = (E, \mathsf{M}, R, @, L, l)$ be a system modelled as a DCR Graph and $t = (c, \Sigma)$ a test case. Under the cost function $\mathcal{K}^{\Sigma}_{G}$ we have that:*

1. *G* passes *the open test t^{+} iff any optimal alignment of c, G has cost 0.*
2. *G* passes *the open test t^{-} iff any optimal alignment of c, G has cost ∞.*

Proof. (1). G passes the open test t^{+} iff there exists a trace p s.t. $p|_{\Sigma} = c$ iff (Lemma 40) there exists an alignment of c, G with cost 0 iff the optimal alignment of c, G has cost 0. (2). G passes the open test t^{-} iff forall traces x $x|_{\Sigma} \neq c$ iff (Lemma 40) forall traces x the alignment γ of G, c where $\gamma_{l} = x$ has cost ∞ iff any optimal alignment of G, c has cost ∞. □

Having successfully mapped the problem of checking open tests, we can now start to define a static check that prunes the search-space during alignment. In order to do this, we define the following check for when an event is not reachable under a context.

Definition 42 (Non-reachability under context). *Let e be an event in a graph G with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, and let X be a set of events G. We say that* the event e is inaccessible under X in G *iff $e \in X$ or one of the following holds:*

(a) $e \notin \mathsf{In}$ and forall $e' \in (\rightarrow\!+ e)$ we have e' inaccessible in G under $X \cup \{e\}$.

(b) There exists $e' \in (\rightarrow\!\bullet\, e)$ with $e' \notin \mathsf{Ex}$, $e' \in \mathsf{In}$, e' inaccessible in G under $X \cup e$; and forall $e'' \in (\rightarrow\!\%\, e')$ also e'' is inaccessible in G under $X \cup \{e, e'\}$.

(c) There exists $e' \in (\rightarrow\!\diamond\, e)$ with $e' \in \mathsf{Re}$, $e' \in \mathsf{In}$, e' not accessible in G under $X \cup \{e\}$, and forall $e'' \in (\rightarrow\!\%\, e')$ we have e'' inaccessible in G under $X \cup \{e, e'\}$.

We now need to correlate this definition to the execution semantics of DCR graphs, to which we start with enabledness.

Lemma 43. *If an event e with $e \notin X$ is inaccessible in G under some X, then $e \notin \mathsf{enabled}(G)$.*

Proof. Because $e \notin X$, either (a), (b), or (c) of Definition 42 is true. It is straightforward to verify that in either case, e is not enabled. □

We can now show that this definition of non-reachability under a context holds over any model-step outside the context.

Lemma 44 (Preservation of inaccessibility). *let e, j be events such that e is inaccessible in G under X, and $j \notin X$. If $G \rightarrow_j G'$, then e is inaccessible in G' under X.*

Proof. By induction on the $|\mathsf{E}(G)| - |X|$, that is, the number of events of G *not* in X. For the base case, if 0 events of X are not in G, then $X = \mathsf{E}(G)$, and every event is inaccessible in G, clearly also after executing some event. For the inductive step, suppose the statement holds for any $|\mathsf{E}(G)| - |X| < k$ for some $k > 0$.

If $e \notin X$ we are done. Because e is inaccessible in G under X, (a), (b), or (c) of Definition 42 holds for G and X. We consider each in turn. Note first that clearly j is not inaccessible in G because otherwise it was not enabled.

(a). It is enough to show that $e \notin \mathsf{In}$ and inaccessible under $X \cup e$ also in G'. For the former, suppose not. Then $j \rightarrow\!\!\!\!\% \; e$. But then $j \notin \mathsf{enabled}(G)$; contradiction. For the latter, we have e' inaccessible in G under $X \cup \{e\}$; by IH e' then inaccessible also in G' under $X \cup \{e\}$.

(b). We have some $e' \in (\rightarrow\!\!\bullet\, e)$ with $e' \notin \mathsf{Ex}$, $e' \in \mathsf{In}$, e' inaccessible in G under $X \cup e$; and forall $e'' \in (\rightarrow\!\!\!\!\% \, e')$ also e'' is inaccessible in G under $X \cup \{e, e'\}$. Clearly $j \neq e'$ because e' was not enabled in G, so $e' \notin \mathsf{Ex}'$. Similarly, j did not exclude e', because then j would not have inaccessible in G and so not enabled, so $e \in \mathsf{In}'$. By IH, e' is inaccessible in G under $X \cup \{e\}$ and also by IH, so is any e'' excluding e''.

(c). We have some $e' \in (\rightarrow\!\!\diamond e)$ with $e' \in \mathsf{Re}$, $e' \in \mathsf{In}$, e' not accessible in G under $X \cup \{e\}$, and forall $e'' \in (\rightarrow\!\!\!\!\% \, e')$ we have e'' inaccessible in G under $X \cup \{e, e'\}$. Again, $e' \in \mathsf{In}'$ because otherwise j would have been inaccessible and thus not enabled in G; and for that reason also $j \neq e'$ and so $e' \in \mathsf{Re}'$. By IH, e' is inaccessible in G under $X \cup \{e\}$ and also by IH, so is any e'' excluding e''. □

Using this lemma, we can now show that non-reachability under a context also holds for any run outside the context.

Lemma 45. *let G be a graph, e an event of G, Σ a context, and assume that $\ell(e) \notin \Sigma$. Take $X = \{e \mid \ell(e) \in \Sigma\}$, and assume further that e is inaccessible in G under X. Then any for run $G \rightarrow^*_\phi G'$ with $\ell(\phi) \in \Sigma^{\complement *}$, we have e inaccessible under X in G'.*

Proof. By induction on the length of ϕ. The base case is trivial, so consider $G \rightarrow^*_\phi G_k \rightarrow_{\phi_k} G_{k+1}$. By IH, we have e inaccessible under X in G_k. Wlog, pick a j with $\ell(j) = \phi_k$ causing the final transition. By assumption $\ell(j) = \phi_k \in \Sigma^{\complement}$, so $j \notin X$. By Lemma 45, e is inaccessible in G_{k+1} under X. □

Finally we can state a theorem, saying that if the next event of the test-case is inaccessible under the context, the optimal alignment must have cost ∞.

Theorem 46. *Let $t = (c, \Sigma)$ be a test with sequence $c = \langle \ell(e_1), ..., \ell(e_n) \rangle$ and G a DCR graph. Write $X = \{e \mid \ell(e) \in \Sigma\}$ Suppose a partial alignment*

γ_i contains synchronous moves for $c_{1..i}$ with $G \rightarrow^*_{\gamma_i} G_i$. Then any optimal alignment of c_{i+1} and G_i has cost ∞ under cost function $\mathcal{K}^{\Sigma}_{G}$ if either:

1. *$i < n$ and e_{i+1} is inaccessible under $X \setminus \{e_{i+1}\}$ in G_i, or*
2. *$i = n$ and $\exists e \in \mathsf{Re}_i$, s.t. $e \in \mathsf{In}_i$, e is inaccessible under X in G_i, and $\forall e' \in (\rightarrow\!\!\%\, e)$ we have e' is inaccessible under X in G_i.*

Proof. By contradiction. Assume γ_{i+1} is an optimal alignment of c_{i+1} and G_i with cost 0.

1. Assume $i < n$ etc. Because γ_{i+1} is an optimal alignment of c_{i+1} and G_i with cost 0, we must have $(\ell(e_{i+1}), e_{i+1}) \in \gamma_{i+1}$. But e_{i+1} is inaccessible under $X \setminus \{e_{i+1}\}$, so by Lemma 43 not enabled in G_i. It follows that we must have some alignment $\gamma' \subset \gamma_{i+1}$, s.t. $\ell(\gamma'_\phi) \in \Sigma^{\complement *}$ with $G_i \rightarrow^*_{\gamma'_\phi} G'$ with $e_{i+1} \in \mathsf{enabled}(G')$. But by Lemma 45, e_{i+1} is inaccessible in G' under $X \setminus \{e_{i+1}\}$, and therefore not enabled in G'; contradiction.
2. Assume $n = i$ etc. We have $c_{i+1} = \langle\rangle$ meaning any move in the alignment of c_{i+1} and G_i must have $l = \gg$. As $e \in \mathsf{Re}_i$ and $e \in \mathsf{In}_i$, we must execute or exclude e for G_i to become accepting, thus γ_{i+1} contains a move $(\gg, e)$ or $(\gg, e')$ with $e' \in (\rightarrow\!\!\%\, e)$, and as γ_{i+1} has cost 0, any move $(\gg, j) \in \gamma_{i+1}$ must have $\ell(j) \notin \Sigma$. As both e and e' are inacessible in G_i under $`X$, and any move contains j with $\ell(j) \notin \Sigma$ we have by Lemma 45 that e and e' are inaccessible, and therefore by Lemma 43 not enabled, at any point during the alignment. This contradicts either of them being in γ_{i+1}.

□

In practice, this means that for each step in our search we can perform a relatively simple and computationally inexpensive static check, which verifies if the next event in the test trace is inaccessible under the context. If this is the case then it does not matter to further branch out the search space with additional model moves, even if these are outside the context and therefore have cost 0, as we know from the structure of the graph that they will never enable the event we need to synchronize on. In this way we can prune large branches of our search space.

	Std / Std	Std / Pruning	Bit / Std	Bit / Pruning
t0pos	0.50	0.68	0.20	0.50
t1neg	0.33	0.02	0.18	0.02
t2pos	2.44	2.72	0.69	1.13
t3neg	0.14	0.01	0.09	0.01
t4pos	0.60	0.65	0.24	0.36
t5pos	0.82	0.91	0.30	0.54
t6neg	2.80	0.83	0.81	0.56
t7neg	20.13	0.01	5.56	0.01
t8neg	5.01	1.72	1.37	1.38
t9neg	0.26	0.20	0.14	0.17
t10neg	117.79	0.36	28.07	0.36
t11pos	2.43	2.48	0.70	0.93
t12neg	57.40	34.29	13.95	7.96
t13pos	0.26	0.29	0.14	0.35
t14neg	0.37	0.37	0.19	0.35
t15pos	2.24	2.28	0.69	0.97
t16neg	0.14	0.10	0.09	0.13
t17neg	33.08	15.26	8.22	2.35
t18neg	4.41	1.68	1.26	0.84
Avg positive	1.33	1.43	0.42	0.68
Avg negative	20.15	4.57	4.99	1.18
Avg all	13.22	3.41	3.31	1.00

Table 1: Results for running the test-suite on the full model as seen in Appendix A. All results denoted in milliseconds. All tests pass. Results are shown for all combinations of standard vs bit implementation and with / without pruning based on theorem 46.

6.1. Benchmarking

We benchmark the above alignment mapping by checking 18 test cases[4] against the full model as seen in Appendix A. We present the results for both the standard alignment algorithm for DCR graphs [19] implemented in Typescript and a new Typescript implementation[5] implementing set-operations as bit-operations as described in more detail in [63]. Both implementations are run with and without online pruning of the search-space based on the result

[4]https://github.com/Axel0087/BitDCRAlign/blob/main/tests.json
[5]https://github.com/Axel0087/BitDCRAlign

of theorem 46. We show the full results seen in Table 1.

We note that for both positive and negative tests, using the bit implementation provides a significant speedup from an avg of 13.22ms down to 3.31ms. When looking at the effects of the pruning, we note that utilizing this approach is slightly slower for test-cases where an alignment can be found. This slowdown is to be expected as the pruning is intended for cases where the entire search-space must be exhausted, which is shown in the case of the negative tests, where the average runtime is significantly reduced for both implementations. Finally, it should be noted that this reduction in the runtime on negative cases severely outweighs the increased runtime on the positive cases as the negative tests are much more computationally expensive.

7. Conclusion and Discussion

In this paper we extended the general theory for testing abstractions of process models based on a notion of open tests as introduced in [18] by showing how the problem of dynamically checking open tests against a model can be mapped to an alignment-checking problem. We applied this mapping to DCR Graphs, using efficient alignment-checking techniques from [19], introduced additional novel search-space pruning techniques that are particularly well suited to finding alignments for open tests, and implemented these algorithms using efficient bit-operations. Finally we showed through experimentation that (1) open tests on DCR Graphs of reasonable size can be checked in milliseconds, (2) our pruning technique significantly reduces the run-time of alignment checking by an factor of 4 in cases where the entire state-space of the model needs to be explored, and (3) the use of bit-operation further reduces the run-time by an additional factor of 4.

In future work we aim to investigate other techniques from the model checking world such as ample sets or stubborn sets to further improve the alignment performance. In addition we currently only support the basic definition of DCR graphs and in the future we plan to extend our test approach to handle also subprocesses, time and data. We also intend to further prune the search-space of the alignment checking algorithm by taking into consideration the independence relation between events [64].

Appendix A. Appendix: Complete UO/EO process

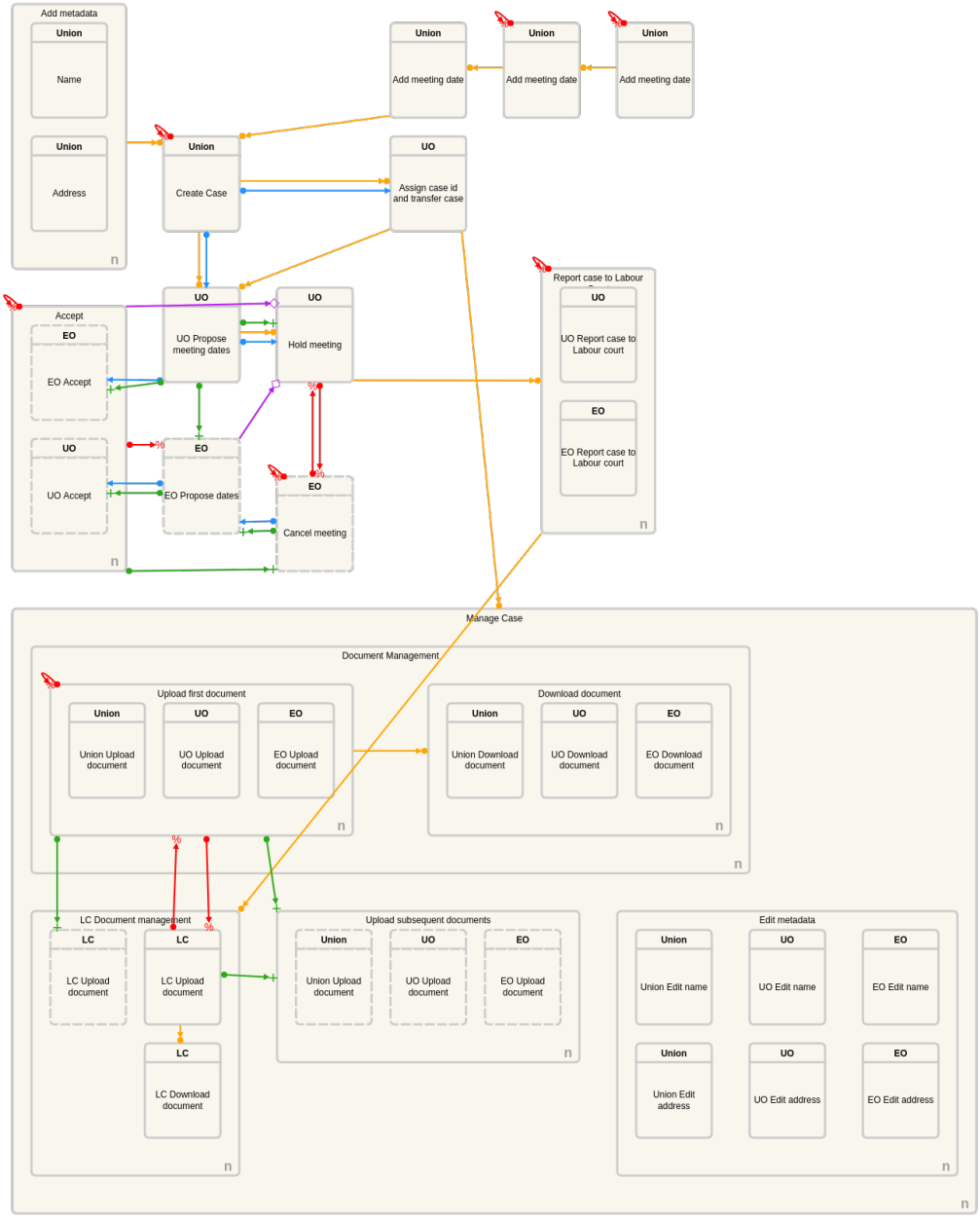


Figure A.3: Complete DCR Graph model of the case handling process

References

[1] T. Hildebrandt, R. R. Mukkamala, Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs, in: Post-proceedings of PLACES 2010, Vol. 69 of EPTCS, 2010, pp. 59–73. doi:10.4204/EPTCS.69.5.

[2] H. A. Reijers, T. Slaats, C. Stahl, Declarative modeling–an academic dream or the future for bpm?, in: Business Process Management, Springer, 2013, pp. 307–322.

[3] Object Management Group, Case Management Model and Notation, Tech. Rep. formal/2014-05-05, Object Management Group, version 1.0 (May 2014).

[4] M. Pesic, W. M. Van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management, 2006, pp. 169–180.

[5] M. Pesic, H. Schonenberg, W. M. P. v. d. Aalst, DECLARE: Full Support for Loosely-Structured Processes, in: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, IEEE, 2007, pp. 287–300.

[6] Object Management Group BPMN Technical Committee, Business Process Model and Notation, Version 2.0 (2013).

[7] S. Zugal, J. Pinggera, B. Weber, The impact of testcases on the maintainability of declarative process models, Enterprise, Business-Process and Information Systems Modeling (2011) 163–177.

[8] T. Hildebrandt, R. R. Mukkamala, T. Slaats, Designing a cross-organizational case management system using dynamic condition response graphs, in: 2011 IEEE 15th international enterprise distributed object computing conference, IEEE, 2011, pp. 161–170.

[9] R. R. Mukkamala, A formal model for declarative workflows - dynamic condition response graphs, Ph.D. thesis, IT University of Copenhagen (March 2012).

[10] T. Slaats, Flexible process notations for cross-organizational case management systems, Ph.D. thesis, IT University of Copenhagen (January 2015).

[11] S. Debois, T. Hildebrandt, The DCR Workbench: Declarative Choreographies for Collaborative Processes, in: Behavioural Types: from Theory to Tools, River Publishers, 2017, pp. 99–124.

[12] K. Beck, Test-driven development: by example (2003).

[13] D. Janzen, H. Saiedian, Test-driven development concepts, taxonomy, and future direction, Computer 38 (9) (2005) 43–50.

[14] S. Zugal, J. Pinggera, B. Weber, Creating declarative process models using test driven modeling suite, in: CAiSE Forum 2011, 2012, pp. 16–32. doi:10.1007/978-3-642-29749-6_2.

[15] M. Marquard, M. Shahzad, T. Slaats, Web-based modelling and collaborative simulation of declarative processes, in: Business Process Management, Springer, 2015, pp. 209–225.

[16] T. Hildebrandt, S. Debois, T. Slaats, M. Marquard, Managing complexity in process digitalisation with dynamic condition response graphs, Vol. 1898, CEUR Workshop Proceedings, 2017, 2nd Workshop on Managed Complexity, ManComp 2017 ; Conference date: 28-08-2017 Through 28-08-2017.
URL https://wwwswt.informatik.uni-rostock.de/ManComp2017/

[17] S. Debois, T. T. Hildebrandt, M. Marquard, T. Slaats, The DCR graphs process portal, in: L. Azevedo, C. Cabanillas (Eds.), Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016, Vol. 1789 of CEUR Workshop Proceedings, CEUR-WS.org, 2016, pp. 7–11.
URL https://ceur-ws.org/Vol-1789/bpm-demo-2016-paper2.pdf

[18] T. Slaats, S. Debois, T. Hildebrandt, Open to change: A theory for iterative test-driven modelling, in: M. Weske, M. Montali, I. Weber, J. vom Brocke (Eds.), Business Process Management, Springer International Publishing, Cham, 2018, pp. 31–47.

[19] A. K. F. Christfort, T. Slaats, Efficient optimal alignment between dynamic condition response graphs and traces, in: C. Di Francescomarino, A. Burattin, C. Janiesch, S. Sadiq (Eds.), Business Process Management, Springer Nature Switzerland, Cham, 2023, pp. 3–19.

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

[21] D. M. Bushnell, Research Conducted at the Institute for Computer Applications in Science and Engineering for the Period October 1, 1999 through March 31, 2000, Technical Report NASA/CR-2000-210105, NAS 1.26:210105, NASA (2000).

[22] S. Debois, T. T. Hildebrandt, T. Slaats, Replication, Refinement & Reachability: Complexity in Dynamic Condition-Response Graphs, Acta Informatica (2017).

[23] J. C. Baeten, R. J. van Glabbeek, Another look at abstraction in process algebra, in: International Colloquium on Automata, Languages, and Programming, Springer, 1987, pp. 84–94.

[24] E. M. Clarke, O. Grumberg, D. E. Long, Model checking and abstraction, ACM transactions on Programming Languages and Systems 16 (5) (1994) 1512–1542.

[25] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM, 1979, pp. 269–282.

[26] M. D. Ernst, Static and dynamic analysis: Synergy and duality, in: ICSE Workshop on Dynamic Analysis, 2003, pp. 24–27.

[27] L. Zhang, J. Zhou, D. Hao, L. Zhang, H. Mei, Prioritizing junit test cases in absence of coverage information, in: Software Maintenance, IEEE, 2009, pp. 19–28.

[28] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, G. Rothermel, A static approach to prioritizing junit test cases, IEEE Transactions on Software Engineering 38 (6) (2012) 1258–1275.

[29] A. A. Andaloussi, A. Burattin, T. Slaats, E. Kindler, B. Weber, On the declarative paradigm in hybrid business process representations: A conceptual framework and a systematic literature study, Inf. Syst. 91 (2020) 101505. doi:10.1016/J.IS.2020.101505.
URL https://doi.org/10.1016/j.is.2020.101505

[30] A. A. Andaloussi, C. J. Davis, A. Burattin, H. A. López, T. Slaats, B. Weber, Understanding quality in declarative process modeling through the mental models of experts, in: D. Fahland, C. Ghidini, J. Becker, M. Dumas (Eds.), Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings, Vol. 12168 of Lecture Notes in Computer Science, Springer, 2020, pp. 417–434. doi:10.1007/978-3-030-58666-9_24.
URL https://doi.org/10.1007/978-3-030-58666-9_24

[31] A. A. Andaloussi, A. Burattin, T. Slaats, E. Kindler, B. Weber, Complexity in declarative process models: Metrics and multi-modal assessment of cognitive load, Expert Syst. Appl. 233 (2023) 120924. doi:10.1016/J.ESWA.2023.120924.
URL https://doi.org/10.1016/j.eswa.2023.120924

[32] A. A. Andaloussi, A. Burattin, T. Slaats, A. C. M. Petersen, T. T. Hildebrandt, B. Weber, Exploring the understandability of a hybrid process design artifact based on DCR graphs, in: I. Reinhartz-Berger, J. Zdravkovic, J. Gulden, R. Schmidt (Eds.), Enterprise, Business-Process and Information Systems Modeling - 20th International Conference, BPMDS 2019, 24th International Conference, EMMSAD 2019, Held at CAiSE 2019, Rome, Italy, June 3-4, 2019, Proceedings, Vol. 352 of Lecture Notes in Business Information Processing, Springer, 2019, pp. 69–84. doi:10.1007/978-3-030-20618-5_5.
URL https://doi.org/10.1007/978-3-030-20618-5_5

[33] A. A. Andaloussi, F. Zerbato, A. Burattin, T. Slaats, T. T. Hildebrandt, B. Weber, Exploring how users engage with hybrid process artifacts based on declarative process models: a behavioral analysis based on eye-tracking and think-aloud, Softw. Syst. Model. 20 (5) (2021) 1437–1464. doi:10.1007/S10270-020-00811-8.
URL https://doi.org/10.1007/s10270-020-00811-8

[34] M. Westergaard, T. Slaats, Mixing paradigms for more comprehensible models, in: Business Process Management, Springer, 2013, pp. 283–290.

[35] J. De Smedt, J. De Weerdt, J. Vanthienen, G. Poels, Mixed-paradigm process modeling with intertwined state spaces, Business and Information Systems Engineering 58 (1) (2016) 19–29. doi:10.1007/s12599-015-0416-y.

[36] J. De Smedt, Studies on declarative process modeling and its relation to procedural techniques. (2016).

[37] K. Kluza, G. J. Nalepa, Formal Model of Business Processes Integrated with Business Rules, Information Systems Frontiers (feb 2018). doi:10.1007/s10796-018-9826-y.
URL http://link.springer.com/10.1007/s10796-018-9826-y

[38] S. Sadiq, W. Sadiq, M. Orlowska, Pockets of Flexibility in Workflow Specification, 2001, pp. 513–526. doi:10.1007/3-540-45581-7_38.
URL http://link.springer.com/10.1007/3-540-45581-7_38

[39] W. M. P. van der Aalst, M. Adams, A. H. M. ter Hofstede, M. Pesic, H. Schonenberg, Flexibility as a Service, 2009, pp. 319–333. doi:10.1007/978-3-642-04205-8_27.
URL http://link.springer.com/10.1007/978-3-642-04205-8_27

[40] T. Slaats, D. M. Schunselaar, F. M. Maggi, H. A. Reijers, The semantics of hybrid process models, in: Cooperative Information Systems, 2016, pp. 531–551.

[41] R. Jagadeesh Chandra Bose, W. M. van der Aalst, Process diagnostics using trace alignment: Opportunities, issues, and challenges, Information Systems 37 (2) (2012) 117–141, management and Engineering of Process-Aware Information Systems.

[42] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance checking, Springer. (2018).

[43] A. Adriansyah, Aligning observed and modeled behavior, Ph.D. thesis, Mathematics and Computer Science (2014).

[44] W. L. J. Lee, H. Verbeek, J. Munoz-Gama, W. M. van der Aalst, M. Sepúlveda, Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining, Information Sciences 466 (2018) 55–91.

[45] D. Reißner, R. Conforti, M. Dumas, M. La Rosa, A. Armas-Cervantes, Scalable conformance checking of business processes, in: On the Move to Meaningful Internet Systems. OTM 2017 Conferences, 2017, pp. 607–627.

[46] M. Boltenhagen, T. Chatain, J. Carmona, Optimized sat encoding of conformance checking artefacts, Computing 103 (1) (2021) 29–50.

[47] M. F. Sani, J. J. G. Gonzalez, S. J. van Zelst, W. M. van der Aalst, Conformance checking approximation using simulation, in: 2020 2nd International Conference on Process Mining (ICPM), 2020, pp. 105–112.

[48] M. Fani Sani, S. J. van Zelst, W. M. P. van der Aalst, Conformance checking approximation using subset selection and edit distance, in: Advanced Information Systems Engineering, 2020, pp. 234–251.

[49] A. Awad, K. Raun, M. Weidlich, Efficient approximate conformance checking using trie data structures, in: 2021 3rd International Conference on Process Mining (ICPM), 2021, pp. 1–8.

[50] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, W. M. P. van der Aalst, Online conformance checking: relating event streams to process models using prefix-alignments, International Journal of Data Science and Analytics 8 (3) (2019) 269–284.

[51] X. Lu, D. Fahland, W. M. P. van der Aalst, Conformance checking based on partially ordered event data, in: Business Process Management Workshops, 2015, pp. 75–88.

[52] M. de Leoni, J. Munoz-Gama, J. Carmona, W. M. P. van der Aalst, Decomposing alignment-based conformance checking of data-aware process models, in: On the Move to Meaningful Internet Systems: OTM 2014 Conferences, 2014, pp. 3–20.

[53] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Cocomot: Conformance checking of multi-perspective processes via smt, in: Business Process Management, 2021, pp. 217–234.

[54] T. Chatain, J. Carmona, Anti-alignments in conformance checking – the dark side of process models, in: Application and Theory of Petri Nets and Concurrency, 2016, pp. 240–258.

[55] B. F. van Dongen, J. De Smedt, C. Di Ciccio, J. Mendling, Conformance checking of mixed-paradigm process models, Information Systems 102 (2021) 101685.

[56] M. de Leoni, F. M. Maggi, W. M. van der Aalst, An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems 47 (2015) 258–277.

[57] M. de Leoni, F. M. Maggi, W. M. P. van der Aalst, Aligning event logs and declarative process models for conformance checking, in: Business Process Management, 2012, pp. 82–97.

[58] G. De Giacomo, F. M. Maggi, A. Marrella, S. Sardina, Computing trace alignment against declarative process models through planning, Proceedings of the International Conference on Automated Planning and Scheduling 26 (1) (2016) 367–375.

[59] G. Winskel, Events in computation, Ph.D. thesis, Computer Science Deptartment, University of Edinburgh (1980).

[60] D. A. Basin, S. Debois, T. T. Hildebrandt, In the Nick of Time: Proactive Prevention of Obligation Violations, in: Computer Security Foundations, 2016, pp. 120–134.

[61] S. Debois, T. T. Hildebrandt, T. Slaats, Hierarchical Declarative Modelling with Refinement and Sub-processes, in: Business Process Management, 2014, pp. 18–33.

[62] R. R. Mukkamala, T. T. Hildebrandt, From dynamic condition response structures to büchi automata, in: J. Liu, D. A. Peled, B. Wang, F. Wang (Eds.), 4th IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2010, Taipei, Taiwan, 25-27 August 2010, IEEE Computer Society, 2010, pp. 187–190. doi:10.1109/TASE.2010.

22.
URL https://doi.org/10.1109/TASE.2010.22

[63] C. O. Back, T. Slaats, T. T. Hildebrandt, M. Marquard, Discover: accurate and efficient discovery of declarative process models, International Journal on Software Tools for Technology Transfer 24 (4) (2022) 563–587. doi:10.1007/s10009-021-00616-0.
URL https://doi.org/10.1007/s10009-021-00616-0

[64] S. Debois, T. T. Hildebrandt, T. Slaats, Concurrency and Asynchrony in Declarative Workflows, in: Business Process Management, 2015, pp. 72–89. doi:10.1007/978-3-319-23063-4_5.

Chapter 10

Handling time

10.1 DD-DisCoveR: Mining timed DCR Graphs using the pm4py DisCoveR DCR extension

Remark 10.1. *This work is a preprint draft paper for the International Conference on Process Mining 2024 demo track*[1].

[1] `https://icpmconference.org/2024/call-for-demos/`

DD-DisCoveR: Mining timed DCR Graphs using the pm4py DisCoveR DCR extension

Vlad Paul Cosma[1], Tijs Slaats[1] and Thomas T. Hildebrandt[1]

[1]*University of Copenhagen, Denmark*

Abstract

We present DD-DisCoveR an extension for declarative process miners to mine delays for condition relations and deadlines for responses.

Keywords

Time, Delay, Deadline, Dynamic Condition Response Graphs

Metadata description	Value
Tool name	DD-DisCoveR
Current version	1.0
Legal code license	Apache 3.0
Languages, tools and services used	Python, Jupyter Notebooks, Matplotlib
Supported operating environment	Microsoft Windows, Mac, GNU/Linux
Download/Demo URL	https://github.com/paul-cvp/delay-deadline-miner
Documentation URL	https://github.com/paul-cvp/delay-deadline-miner
Source code repository	https://github.com/paul-cvp/delay-deadline-miner
Screencast video	https://github.com/paul-cvp/delay-deadline-miner

1. Introduction

We present DD-DisCoveR (Delay Deadline DisCoveR) the first tool and technique that given an event log and a Dynamic Condition Response (DCR) Graph extracts delays and deadlines between pairs of events. We extract the timing values from event timestamps by applying the definition of a delay on a condition and a deadline on a response as given in the formal definition of a Timed Dynamic Condition Response Graph [1]. As such the tool can be seen as an extension with time of the award winning DisCoveR miner [2].

We show how the tool can provide an extensive description of timing distributions as histograms as well as fitting parametric probability distributions on top of the histograms. Given a family of parametric distribution it selects the one with the smallest residual sum of squares error between the binned data and the parametric distribution. Our technique complements simulation tools by allowing us to sample timings from the fitted parametric distributions. We apply this technique on the Road Traffic Fine Management Process (RTFMP) event log[3].

ICPM 2024 Tool Demonstration Track, October 14-18, 2024, Kongens Lyngby, Denmark

vco@di.ku.dk (V. P. Cosma); slaats@di.ku.dk (T. Slaats); hilde@di.ku.dk (T. T. Hildebrandt)

0000-0001-8022-6402 (V. P. Cosma); 0000-0001-6244-6970 (T. Slaats); 0000-0002-7435-5563 (T. T. Hildebrandt)

Related work Mining timing information has been done on all major process modelling formalism: Declare [4], time BPMN[5, 6, 7], timed Petri Nets[8] and timed automata [9]. In [6] lead and lag time are equivalent to the minimum delay and maximum deadline. In [7] the authors mine extraneous delays. [8] mines probabilistic delays on Stochastic Petri Nets. Work in the [10] shows how durations are overlayed on Petri Nets on a per case basis.

2. Preliminaries

We recall the definition of timed Dynamic Condition Response Graphs [1]. The definition is simplified to only consider timed relations, namely conditions and responses. Moreover, instead of nodes being events labelled by activity names we let nodes of the graphs be simply activities, since the DisCoveR process miner in any case produces graphs where every event is uniquely labelled. In the formal definition, time is represented as discrete ticks, that is natural numbers, which in the algorithm is set to the smallest time step needed to represent the delays and deadlines found in the log.

A *timed Condition Response Graph* G is given by a tuple $(A, M, \twoheadrightarrow, \bullet\!\!\rightarrow)$where

(i) A is a finite set of *activities*,

(ii) $M = (Ex, Re, In) \in \big((A \rightharpoonup \omega) \times (A \rightharpoonup \infty) \times A\big)$ is the *timed marking*,

(iii) $\twoheadrightarrow \subseteq A \times \omega \times A$, is the *timed condition relation*,

(iv) $\bullet\!\!\rightarrow \subseteq A \times \infty \times A$, is the timed response relation.

Compared to core DCR graphs, timed DCR Graphs add a natural number $k \in \omega$ to the condition relations, denoting the delay, and a natural number or infinity $d \in \infty$ to the response relations, denoting the deadline. Infinity is used to represent responses with infinite deadlines, i.e. a response that must eventually happen but not within a given time, these are shown as response arrows without a deadline. The timed marking represents the state of the DCR graph and consists a triple (Ex, Re, In), where $Ex(a) = k$ if the last execution of activity a was k time steps ago, and $Re(a) = d$ if a is a pending response and must happen within d time steps if $a \in In$, i.e. a is included. We will abuse notation and also sometimes consider Re as the set $\{a \mid \exists d.Re(a) = d\}$, i.e. the set of activities pending with some deadline.

We now give the definition of an event log. An event log over a given set of activities A and time domain TD being ISO8601 time stamps is defined as $L = (E, C, \alpha, \gamma, \beta, \succ)$ where:

(i) E is a finite set of events,

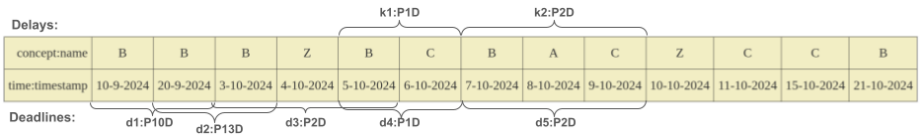
(ii) C is a finite set of cases (process instances),

(iii) $\alpha : E \to A$ maps each event to an activity,

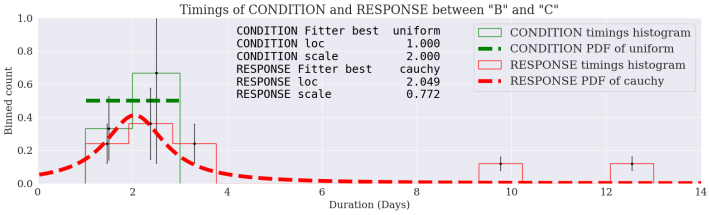
(iv) $\gamma : E \to TD$ maps each event to a timestamp,

(v) $\beta : E \to C$ maps each event to a case and is surjective,

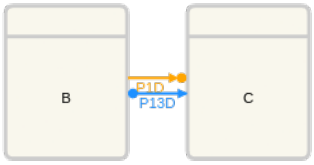
(vi) $\succ \subseteq E \times E$ is the succession relation, which for every case $c \in C$ is a total ordering on the set $\beta^{-1}(c)$



(a) A timed trace with delays (k_1, k_2) and deadlines $(d_1, .., d_5)$ between B and C



(b) Timing data as histograms and fitted functions



(c) Example DCR Graph

Figure 1: Example trace, DCR Graph and extracted timing data

Given an event log L we define the trace function $Tr_L : C \to \mathcal{P}(E)$ by $Tr_L(c) = \{e \mid \beta(e) = c\}$, i.e. returning all events belonging to the same case. We require that for events belonging to the same case the succession relation respect the time ordering, i.e. $\forall c \in C, \forall e_1, e_2 \in Tr_L(c)$ if $e_1 \succ e_2$ then $\gamma(e_1) \leq \gamma(e_2)$. We assume a function $\Delta : TD \times TD \to \omega$ returning an absolute number of time steps between two timestamps.

3. Mining delays and deadlines

Given an event log we mine a timed DCR Graph by first mining an untimed DCR Graph and then mine the minimum observed delays for condition relations and the maximum observed deadlines for response relations.

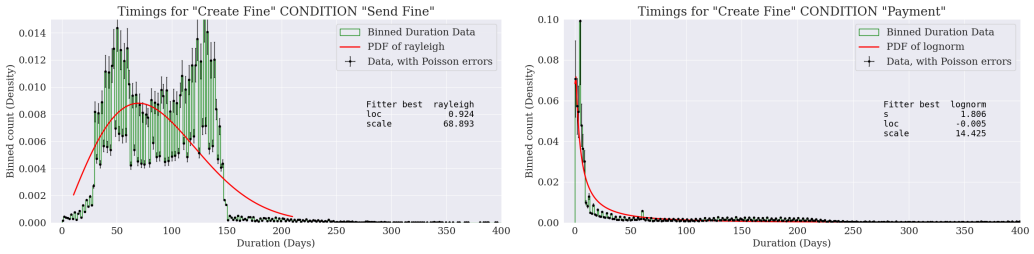
[Delay and deadline mining] Given an event log L and an untimed DCR graph G mined from the log L. We extend the condition and response relations to timed relations as follows:

(i) For $a \rightarrowtriangle b$ let $a \xrightarrow{k} b$, where
$k = MIN\{\Delta(\gamma(e_j), \gamma(e_i)) | \exists c \in C. e_i \succ e_j \in Tr_L(c). \alpha(e_i) = a, \alpha(e_j) = b$ and $\forall e \in Tr_L(c). e_i \succ e \succ e_j \implies \alpha(e) \notin \{a, b\}\}$

(ii) For $a \bullet\!\!\to b$ let $a \xrightarrow{d} b$, where
$d = MAX\{\Delta(\gamma(e_j), \gamma(e_i)) | \exists c \in C. e_i \succ e_j \in Tr_L(c). (\alpha(e_i) = a, \alpha(e_j) = b \vee \alpha(e_i) = \alpha(e_j) = a)$
and $\forall e \in Tr_L(c). e_i \succ e \succ e_j \implies \alpha(e) \notin \{a, b\}\}$

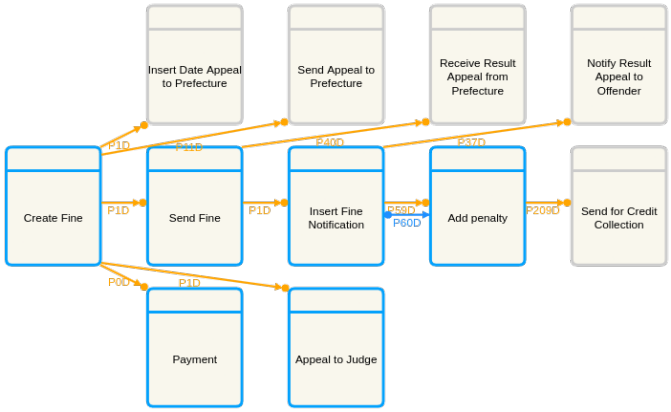
The idea can be explained looking at the example trace in Fig. 1a. For any two activities B and C related by a condition we consider every pair in a trace for which there exists no other activity B or C in between. The delay for the condition between B and C is then defined as the minimum time difference between any such pair of events. In the example trace, we have three such pairs with time differences 1, 2, and the minimum time difference is 1 day, represented in the ISO8601 standard[1] as P1D in the DCR Graph 1c. Mining a deadline for the response relation

[1] www.iso.org/iso-8601-date-and-time-format.html



(a) Create Fine →Send Fine

(b) Create Fine →Payment



(c) Timed Condition Response Graph mined from the RTFMP log

Figure 2: Running DD-DisCoveR on the Road Traffic Fine Management Process event log

between B and C is similar. As before we consider the pair B and C with no B or C in between, and also pairs of activities B and B with no B or C in between. The rationale is that if B occurs twice without the occurrence of an intermediate C, then the response relation allows us to not do C for the duration of time between the two executions of B. In the example trace, we have 5 such pairs with time differences $10, 13, 2, 1, 2$, and the maximum time difference is 13 days, represented in the ISO8601 standard as P13D. We mine different timing data for the condition and the response between activities B and C. We see this both in the histograms and the fitted parametric distributions in fig. 1b (uniform for the condition and cauchy for the response).

Descriptive statistics for mining time Once we extract the timing information we are able to bin it in histograms and fit[2] the parametric distribution which best describes the behaviour between the two activities of interest. The goodness of fit is calculated as the residual sum of square errors.

RTFMP In fig. 2 we show the result of DD-DisCoveR on the RTFMP log [3]. The delays and deadlines are extracted from the timing data which is also shown as histograms (two examples are shown in figs. 2a and 2b). Note that between "Insert Fine Notification" and "Add Penalty" we mine a condition delay of P59D and a response deadline of P60D. Upon closer inspection of

[2]using the fitter python package https://github.com/cokelaer/fitter

the timing data we believe this difference to be a rounding error induced by daylight saving time changes of ±1 hour from the 60 day standard delay/deadline.

4. Conclusion

We presented DD-DisCoveR, a tool that extracts the minimum delay and maximum deadline between pairs of activities that are part of conditions and responses respectively. In future work we intend to use mixtures of distributions create better fits to the data, for example when an exponential plus a Gaussian best describe the behaviour between two activities. We also aim at creating an interactive filtering tool such that the mined delays and deadlines can be tweaked according to the timing distributions.

References

[1] T. Hildebrandt, R. R. Mukkamala, T. Slaats, F. Zanitti, Contracts for cross-organizational workflows as timed dynamic condition response graphs, The Journal of Logic and Algebraic Programming 82 (2013).

[2] C. O. Back, T. Slaats, T. T. Hildebrandt, M. Marquard, Discover: accurate and efficient discovery of declarative process models, in: International Journal on Software Tools for Technology Transfer, 2021. doi:10.1007/s10009-021-00616-0.

[3] F. Mannhardt, M. De Leoni, H. A. Reijers, W. M. Van Der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (2016).

[4] J. M. E. M. van der Werf, R. Mans, W. M. P. van der Aalst, Mining declarative models using time intervals, in: D. Moldt (Ed.), Joint Proceedings of PNSE and ModBE, volume 989 of *CEUR Workshop Proceedings*, 2013. URL: http://ceur-ws.org/Vol-989/paper04b.pdf.

[5] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Toward a time-centric modeling of business processes in bpmn 2.0, in: Proceedings of International Conference on Information Integration and Web-Based Applications &; Services, IIWAS '13, Association for Computing Machinery, USA, ???? doi:10.1145/2539150.2539182.

[6] D. Gagne, A. Trudel, Time-bpmn, in: 2009 IEEE Conference on Commerce and Enterprise Computing, 2009, pp. 361–367. doi:10.1109/CEC.2009.71.

[7] D. Chapela-Campa, M. Dumas, Modeling extraneous activity delays in business process simulation, in: 2022 4th International Conference on Process Mining (ICPM), IEEE, 2022, pp. 72–79.

[8] in: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops : BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, Lecture Notes in Business Information Processing, Springer, Germany, 2014, pp. 15–27. doi:10.1007/978-3-319-06257-0-2.

[9] L. Cornanguer, C. Largouët, L. Rozé, A. Termier, TAG: Learning Timed Automata from Logs, in: AAAI 2022 - 36th AAAI Conference on Artificial Intelligence, Canada, 2022. URL: https://hal.inria.fr/hal-03564455.

[10] W. van der Aalst, Mining Additional Perspectives, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 275–300. doi:10.1007/978-3-662-49851-49.

10.2 Transforming Timed Dynamic Condition Response Graphs to safe Timed-arc Petri Nets

Remark 10.2. *This work is a draft paper containing and extending material from Section 9.1 previously published as [44]: Cosma VP, Hildebrandt TT, Slaats T. Transforming Dynamic Condition Response Graphs to Safe Petri Nets. International Conference on Applications and Theory of Petri Nets and Concurrency 2023 May 28 (pp. 417-439). Cham: Springer Nature Switzerland.*

This Section therefore has many repetitions from Section 9.1.

Mapping Timed Declarative DCR Graph Specifications to safe Timed-arc Petri Nets

Vlad Paul Cosma[1], Thomas T. Hildebrandt[1], and Tijs Slaats[1]

Computer Science Department, Copenhagen University, Denmark
{vco,hilde,slaats}@di.ku.dk

Abstract. We present a semantics preserving mapping from the timed Dynamic Condition Response (DCR) graph constraint based process specification language to 1-safe timed-arc Petri nets with read arcs, extending previous work on a mapping of untimed specifications. The timed DCR graph notation is supported by mature design tools, also including a natural language transformation based on ChatGPT, that domain experts today use to formalize requirements and validate the formalization without being trained in formal logic. We prove that the mapping respects the semantics by showing a bisimulation correspondence between any timed DCR graph specification and its corresponding timed-arc Petri net. The result is supported by a prototype implementation, which transforms a timed DCR graph represented in the DCR XML export format supported by the DCR design tools to the TAPN format. This enables import of the Petri net processes in a range of other tools, in particular the TAPAAL Petri net tool. The mapping is illustrated on a simple running example of an e-shop specification but also tested on a larger and more realistic railroad crossing example, resulting in a Petri Net with 34 places and 99 transitions. Both examples are available online together with the prototype implementation.

Keywords: Petri Nets · Time · DCR graphs · TAPAAL · Bisimilarity

1 Introduction

Formal model-driven engineering is based on the idea that domain requirements and constraints for IT systems are described in a formal model suitable for rigorous analysis, e.g. using model checking tools, and subsequently transformed into code by a property preserving transformation. For this idea to work in practice, it should be possible for people understanding the domain to describe the requirements in a formal notation.

Roughly speaking, formal models come in two kinds: Declarative and imperative models. Intuitively, a declarative model describes which constraints a system must satisfy, while an imperative model describes the sequencing of actions implementing the required behavior. For this reason, declarative modeling notations are often used for the formalization of requirements, while imperative modeling notations are used for the formalization of implementations. Classical

examples of declarative notations are temporal logics such as Linear-time temporal logic (LTL) [30] and extensions with time, such as Monadic First-order Temporal Logic (MFOTL) [7]. Examples of timed imperative modeling notations are finite automata extended with time, such as Timed automata [4, 10]. In between logics and automata, we find the classical Petri Nets model, which both have a declarative flavor of places representing conditions for transitions and an imperative flavor of tokens being moved between places. Petri nets extended with time come in two main variants: Time Petri nets [8], and Timed-arc Petri nets [34].

In this paper, we consider the transformation from process requirements presented in the declarative formal model of Timed Dynamic Condition Response (DCR) graphs to processes expressed as Timed-arc Petri Nets (TAPN) [42]. The DCR graphs notation was introduced in [27, 17] as a formal specification language for distributed workflows. It was shown [27] that DCR graphs can express all formal languages that are the union of a regular and an ω-regular language, and thereby more expressive than LTL. In subsequent papers it has been extended in a range of papers, e.g. adding time, sub-processes, and data (see e.g. [18, 29, 19, 36]) and since 2018 the modelling language has been supported by mature, commercial design tools[1] and workflow tools[2].

The nodes of a DCR graph denote actions (or events) of the process and the directed edges between nodes denote constraints and effects between actions.

We consider as running example a simple e-shop application that has the following specification:

(i) Once an order is added, a payment for the order must be made within 10 time steps.
(ii) Payment information (eg. credit card number) must be provided before a payment can be executed.
(iii) The payment information can be edited any number of times. For security reasons at least 5 time steps must pass between editing the payment information and making the payment.
(iv) A new order cannot be added before a subsequent payment has been made and payment can only be made if an order has been added and is not yet paid.

We can identify 3-4 actions in the system: Add Order, Provide/Edit Payment Information, Make Payment. Below in Fig. 1 we show a timed DCR graph of these requirements modelled in the DCR Solutions design tool and Fig. 2 shows an equivalent Timed Arc Petri Nets with pending places obtained from our transformation as represented in the TAPAAL tool [22]. One may note that the DCR graph represent the initial provision of payment information and subsequent editing of payment information as the same action (EditPaymentInfo), while the Petri net uses two separate transitions. As we will see in the following, a key

[1] Freely available for academic use at `DCRSolutions.net`
[2] See `www.kmd.net/solutions-and-services/case-management-and-document-management/kmd-workzone`

difference between the two notations is that the state of a (timed arc) Petri Net is represented as (timed) tokens on places, while the state of a DCR graph is represented as a marking of each action/event. Related to this, constraints in DCR graphs are expressed directly between actions while constraints in Petri Nets are expressed between transitions and places. This makes DCR Graphs closer to natural language specifications, which do not talk about tokens and places. Indeed, the version 7.0.0 of the DCR Solutions design tool[3] released December 29th, 2023, includes a ChatGPT powered agent that can help identify rules from natural language specifications. We will provide the formal definitions of the two notations in Sec. 3.

In our previous work [11] we transformed core declarative Dynamic Condition Response (DCR) graphs [27, 17] to Petri Nets with inhibitor, read arcs and a new kind of *pending* places used to express an acceptance criteria for infinitary runs needed to express ω-regular properties. We also proved that the labelled transition system of the resulting Petri Net is bisimilar to the lebelled transition system defined by the DCR graph. In the present paper we extend the transformation and proof with time. Concretely our contribution is that we provide a transformation from *extended Timed* DCR graphs [18] to safe (or 1-bounded) Timed-arc Petri Nets with read arcs and pending places (TAPNrp). The read-arcs and pending places can be removed if only finitary runs are of interest, thus giving a standard safe Timed Arc Petri Net (TAPN) which can be analysed in the TAPAAL tool and mapped to Networks of Timed Automata (TA) [10].

2 Related work

Several notations for declarative process modeling have been developed. In addition to DCR graphs, the Declare [1] and Guard-Stage-Milestone (GSM) notations have also seen broad use in the business process management research community.

Declare provides a set of templates for modeling business constraints that are formalized as LTL formulae (parameterized by activities). A Declare model is the conjunction of a set of instantiated formulae. Given the limited expressiveness of the templates, a mapping from DCR graphs to Declare is not possible. Declare has been formalized in other languages such as colored automata [23] and SCIFF [24, 25]. Mappings from Declare to Petri Nets and R/I-nets were provided respectively in [32] and [12], however, proofs of correctness are missing from each of these.

Time constraints were added to the Declare notation in the context of multi-perspective (MP-)Declare [9], which also includes the ability to express data constraints. To facilitate the execution of MP-Declare models a mapping to the specification language Alloy was defined [2].

The GSM notation [21] takes a declarative data-centric approach to modelling processes, where stages of activities in the process are connected through

[3] `documentation.dcr.design/release_notes/release-7-0-0/`.

guards that need to be satisfied for their activation and milestones that represent their acceptance criteria. A mapping has been proposed from Petri Nets to GSM [31], in particular with a focus on representing the output of process discovery algorithms (which usually produce Petri Nets) as GSM models. We are not aware of any direct mappings in the opposite direction. Similarly [14] provides a mapping from DCR graphs to GSM models, an opposite mapping is mentioned as future work but has not yet materialised.

In [16] a subset of the DCR relations and their equivalent Petri Net mapping is presented, without inhibitor arcs and without proof of correctness. [28] provides an encoding of DCR graphs as Büchi automata.

Petri Nets are widely used, and therefore there are also many translations to notations outside the declarative process modeling sphere, for example, Ladder Logic Diagrams [39], Timed Automata [10] and mCRL2[33].

Similarly much work has gone into mapping other modeling notations into Petri Nets, such as UML activity diagrams [38], UML sequence diagrams [41], UML state charts [20], and BPMN [13, 33].

The work in [24] presents logic-based approaches that formalize regulatory models by relying on the deontic notions of obligations and permissions.

Different classes of ω-language Petri Nets have been introduced in [40] and their complexity has been studied in [15]. The definition of acceptance criteria for infinite words in [40] is based on markings being visited infinitely often, similar to the acceptance criteria of Büchi-automata. This differs from the acceptance criteria introduced in the present paper, which is based on pending places, for which tokens cannot rest infinitely without being consumed by a transition being fired. Finally we built on our previous[11] work on mapping untimed DCR Graphs to Petri Nets with inhibitor, read arcs and pending places.

3 Preliminaries

In this section we provide the formal definitions of Timed Dynamic Condition Response graphs and safe Timed-arc Petri Nets with inhibitor and read arcs and pending places.

3.1 Timed Dynamic Condition Response graphs

We give a formal definition of core Dynamic Condition Response (DCR) graphs as attributed directed graphs.[4] For a set A we write $\mathcal{P}(A)$ for the set of all subsets of A, i.e. the powerset of A and $\mathcal{P}_{ne}(A)$ for the set of all non-empty subsets of A. Finally, let $\mathcal{N}$ refer to the set of natural numbers (including zero) and $\mathcal{N}_\omega = \mathcal{N} \cup \{\omega\}$ refer to the set of natural numbers (including zero) and infinity (with infinity written as ω).

[4] The presentation deviates slightly from the original definition given in [16] to facilitate the definition of the mapping to Petri Nets, but defines the same graph structures.

Definition 1. *A* Timed DCR graph G *is given by a tuple* $(E, M, R, @, L, l)$ *where*

(i) E *is a finite set of* events,
(ii) $M = (Ex, Re, In) \in \big((E \rightharpoonup \omega) \times (E \rightharpoonup \infty) \times E\big)$ *is the* timed marking,
(iii) $R \subseteq E \times E$ *is the set of* relations *between events*
(iv) $@ : R \to \mathcal{P}_{ne}(\{\overset{k}{\bullet\!\!\leftarrow}, \overset{d}{\bullet\!\!\rightarrow}, \rightarrow\!\!+, \rightarrow\!\!\%\})$ *is the* relation type assignment *and* $k \in \mathcal{N}$ *is the* delay *and* $d \in \mathcal{N}_\omega$ *is the* deadline.
(v) L *is the set of event labels,*
(vi) $l : E \to L$ *is the labelling function between events and labels.*

The timed marking $M = (Ex, Re, In)$ describes the state of the DCR Graph process by assigning execution times, deadlines and inclusion status to each event in the following way. If an event e has been executed at least once then $Ex(e) = k$, where $k \in \mathcal{N}$ is the number of time steps since the last execution of e. If e has not been executed then $Ex(e)$ is undefined. If $Re(e) = d$ then we say that e is pending with deadline $d \in \mathcal{N}_\omega$, which means that it must be executed within d time-steps or stay forever excluded. The deadline ω represents "eventually", which corresponds to the semantics of untimed DCR graphs. If $e \in In$ we say that it is included and otherwise it is excluded.

Assume a relation $r = (e, e') \in R$ from event e to e'. If $\overset{k}{\bullet\!\!\leftarrow} \in @r$ we say r is a *constraining* relation. If $@r \cap \{\overset{d}{\bullet\!\!\rightarrow}, \rightarrow\!\!+, \rightarrow\!\!\%\} \neq \emptyset$ we say that r is an *effect* relation. Note that r can be both a constraining and an effect relation at the same time.

We write $e \overset{k}{\bullet\!\!\leftarrow} e'$ (or $e' \overset{k}{\rightarrow\!\!\bullet} e$) if $\overset{k}{\bullet\!\!\leftarrow} \in @r$ and say there is a *condition* from e' to e with delay k. The meaning is that at least k time steps must happen after the last execution of e before e' can be executed. If $k = 0$ the condition relation simply states that the event e' must have been executed at least once before e can be executed, which corresponds to the condition relation of untimed DCR graphs. In this case we often omit the delay and simply write $e \bullet\!\!\leftarrow e'$.

We write $e \overset{d}{\bullet\!\!\rightarrow} e'$ if $\overset{d}{\bullet\!\!\rightarrow} \in @r$ and say there is a response from e to e' with deadline d. The meaning is that e' must happen within d time steps after the last execution of e or before that stay excluded.

Finally, we write $e \rightarrow\!\!+ e'$ if $\rightarrow\!\!+ \in @r$ and say there is an inclusion from e to e', and we write $e \rightarrow\!\!\% e'$ if $\rightarrow\!\!\% \in @r$ and say there is an exclusion from e to e'.

We can now explain the details of the timed DCR graph for our running example shown in Fig 1. Events are depicted as boxes containing the action label of the event and relations as arrows. A relation with multiple types is depicted as multiple arrows between the same two events, one arrow for each type. Events that are included in the initial marking are drawn as boxes with a solid border, events that are excluded in the initial marking are drawn as boxes with a dashed border. Consequently, the events labelled EditPaymentInfo and AddOrder are initially included and the event labelled MakePayment is excluded in the initial marking of the graph.

The first requirement, "If an order is made, a payment for the order must be made within 10 time steps" is modelled by a timed response relation ($\overset{10}{\bullet\!\!\rightarrow}$ in

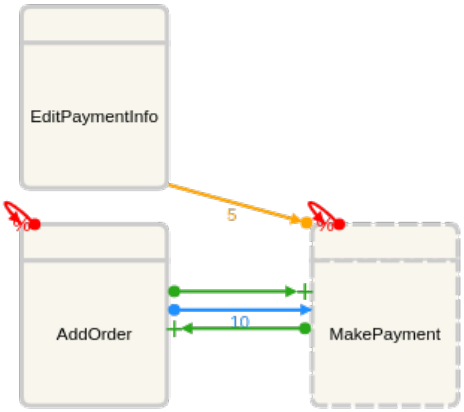


Fig. 1: Timed DCR graph specification for the e-shop process

blue) and an include relation ($\rightarrow\!\!+$ in green) from the event labelled AddOrder to the event labelled MakePayment. The include relation is needed because of the interplay with the fourth requirement described below.

The second requirement, "Payment information (eg. credit card number) must be provided before a payment can be executed" is modelled by a condition relation ($\rightarrow\!\!\bullet$ or $\bullet\!\!\leftarrow$ in orange) from the event labelled EditPaymentInfo to the event labelled MakePayment.

The third requirement, "The payment information may be provided at any time and any number of times." is modelled by having no condition relations pointing to the event labelled EditPaymentInfo and making sure that it is included in the initial marking and never excluded. The security requirement that "5 time steps must pass between editing the payment information and making the payment" is modelled as a delay on the condition relation $\overset{5}{\bullet\!\!\leftarrow}$.

The forth requirement is in two parts. The first part, "a new order cannot be made before a subsequent payment has been made" is modelled by an exclude relation ($\rightarrow\!\!\%$ in red) from AddOrder to itself and an include relation from MakePayment to AddOrder. The effect is that when AddOrder is executed, it excludes itself and is thus no longer available, except if MakePayment is executed, which will include AddOrder again. The second part, "payment can only be made if an order has been made and is not yet paid" is similarly modelled by an exclusion relation from MakePayment to itself and an inclusion relation from MakePayment to AddOrder.

We now define the behaviour of timed DCR graphs. First we introduce some notation for updating markings. When $f : X \to Y$ is a (possibly partial) function, we write $f[x \mapsto y]$ for the function $f' : X \to Y$ which is identical to f, except $f'(x) = y$. We apply this notation also to sets, taking $f[x \mapsto y \mid P(x, y)]$ to be the function f' which is identical to f except that $f'(x) = y$ for all x, y satisfying the given predicate $P(x, y)$.

We define when events are enabled, i.e. can be executed, as follows.

Definition 2 (Event enabling). *Let $(E, M, R, @, L, l)$ be a timed DCR graph. An event $e \in E$ is* enabled for the marking $M = (Ex, Re, In)$*, writing* $\mathsf{enabled}(M, e)$ *if and only if:*

(i) $e \in In$
(ii) $\forall e' \in In.\ e' \overset{k}{\bullet\!\!\rightarrow} e \implies Ex(e') \geq k$
(iii) $\forall e' \notin In.\ e \rightarrow\!\!+ e' \implies Re(e') \geq 0$

The conditions for event enabling state that for an event e to be enabled, (i) it must be included. (ii) Whenever e has a condition relation with delay k from an included event e', then this e' was executed at least k time steps ago. Finally (iii) express that an event is not allowed to include an excluded event where the deadline has passed. This is an adaptation according to the original definition of timed DCR graphs, where such an inclusion was allowed and simply reset the deadline.

A time step n denotes that time advances n steps and can only happen if there are no included pending events with deadline less than n. Formally, we define that a time step n is enabled in marking M, written $\mathsf{enabled}(M, n)$, if $n \leq \mathsf{min}\{d \mid \exists e.e \in In \wedge Re(e) = d\}$. We define the effect of executing an enabled time step n in marking $M = (Ex, Re, In)$ to be the marking $\mathsf{effect}_G(M, n) = (Ex', Re', In)$, where $Ex'(e) = Ex(e) + n$ and $Re'(e) = Re(e) - n$.

We now define the effect of executing an event e for a given marking M.

Definition 3. *Let G be a timed DCR graph with marking $M = (Ex, Re, In)$. The effect of executing an enabled event e is the marking* $\mathsf{effect}_G(M, e) = (Ex', Re', In')$ *where*

$$\begin{aligned} Ex' =& Ex[e \mapsto 0] \\ Re' =& Re[e' \mapsto k \mid \exists k'.\ e \overset{k'}{\bullet\!\!\rightarrow} e' \wedge k = min\{k' \mid e \overset{k'}{\bullet\!\!\rightarrow} e'\}] \\ In' =& (In \setminus \{e' \mid e \rightarrow\!\!\% e'\}) \cup \{e' \mid e \rightarrow\!\!+ e'\} \end{aligned}$$

The timed labelled transition system for timed DCR graphs can now be defined as follows.

Definition 4. *Let $G = (E, M, R, @, L, l)$ be a timed DCR graph. Define a labelled transition relation between markings by $M \overset{\alpha}{\rightarrow}_G \mathsf{effect}_G(M, \alpha)$ if $\mathsf{enabled}(M, \alpha)$, where $\alpha \in E \cup \mathcal{N}$. Write $M \Rightarrow M'$ for $\exists \alpha \in E \cup \mathcal{N}.M \overset{\alpha}{\rightarrow}_G M'$ and write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$. Define $\mathbb{M}_G = \{M' \mid M \Rightarrow^* M'\}$, i.e. the set of all reachable markings from the initial marking M of G. The timed labelled transition system for G is then defined as $[[G]] = (\mathbb{M}_G, M, \rightarrow_G \subset (\mathbb{M}_G \times (E \cup \mathcal{N}) \times \mathbb{M}_G), L, l)$.*

Finally, we define when a finite or infinite execution sequence of a DCR graph is *accepting*. Intuitively, it is required that any included and pending event e in some intermediate state must eventually be executed or no longer included or pending in a later state and infinite execution sequences must contain infinitely many time steps. If one limits attention to finite execution sequences, the acceptance criteria is that no pending event is included in the final state.

Definition 5. *Let* $G = (E, M_0, R, @, L, l)$ *be a timed DCR graph. A finite or infinite sequence of transitions* $M_0 \xrightarrow{e_0}_G M_1 \xrightarrow{e_1}_G \ldots$ *in* $[[G]]$ *with* $M_i = (Ex_i, Re_i, In_i)$*, is* accepting *if* $e \in In_i$ *and* $Re_i(e)$ *defined implies* $\exists j \geq i.(e_j = e \vee e \notin In_j)$.

3.2 Timed arc Petri Nets with read arcs and pending places

There are numerous variants of Petri Nets with different expressive power. As described in the introduction, we use Timed-arc Petri Nets with read arcs and a notion of both finite and infinite acceptance criteria (TAPNrp). Inhibitor arcs (also called negative contextual arcs) are special arcs between places and transitions specifying the constraint that the transition is only enabled if all places related to it by inhibitor arcs are empty. In general, the addition of inhibitor arcs makes the model of Petri Nets Turing complete [3]. However, with the additional requirement of safeness, which means that places can hold at most one token (also known as the property of all the net places being 1-bounded), the notation is restricted to finite state models which also means that inhibitor arcs can be transformed using a complement place and a read arc.

Read arcs (also called test, activator or positive contextual arcs) [6] specify the constraint that a transition is only enabled if all places related to it by read arcs have a token. A key difference between having a read arc and a pair of input and output arcs between a transition and a place, is that read arcs are not consuming the token. This means that two transitions with read arcs to the same place can occur concurrently [26]. However, if two transitions are connected to the same place by a read arc and a standard input arc respectively, the two transitions will still be in conflict. With the use of timed tokens, read arcs also have the property of preserving the token age, whereas a pair of two input and output arcs will reset the age.

The acceptance criteria we introduce is inspired by DCR graphs and allows us to conveniently express the union of regular and ω-regular languages, without needing to refer to explicit markings. The acceptance criteria is defined by indicating a subset of the states to be so-called *pending* places, and then define a finite or infinite execution sequence to be accepting if any token on a pending place is eventually subsequently consumed (but possibly placed back) by the execution of a transition. If one limits attention to finite execution sequences, the acceptance criteria is that all pending places are empty at the end of the execution. Note that the use of read arcs allows us to test, if there is a token on a pending place without consuming it. We define Timed-arc Petri Nets with read arcs and pending places as follows. The definition is adapted from Def.1[22]. We restrict time to discrete natural number time steps as in timed DCR Graphs and the set of timed intervals. Let $\mathsf{Int}_{\geq \delta}$ represent the set of (time) intervals $[\delta, \infty)$ for $\delta \in \mathcal{N}$. Let $\mathsf{Int}_{\leq \Delta}$ represent the set of (time) intervals $[0, \Delta]$ where $\Delta \in \mathcal{N}$. We restrict the intervals $\mathsf{Int}_{\geq \delta}$ to be used on transport arcs, which is used to represent delays and we restrict the intervals $\mathsf{Int}_{\leq \Delta}$ to be used on places, which represent deadlines.

Definition 6 ([22, Def. 1]).
A TAPNrp is a tuple

$$N = (P, T, A, Inhib, Read, Transport, Inv^T, Inv^P, Act, \lambda, Pe),$$

where

(i) P *is a finite set of places,*
(ii) T *is a finite set of transitions s.t.* $P \cap T = \emptyset$,
(iii) $A = IA \sqcup OA$ *is a finite set of input and output arcs, where:*
(1) $IA \subseteq P \times T$ *is a finite set of input arcs,*
(2) $OA \subseteq T \times P$ *is a finite set of output arcs,*
(iv) $Inhib$*:* $IA \to \{true, false\}$ *define inhibitor arcs,*
(v) $Read$*:* $IA \to \{true, false\}$ *define read arcs,*
(vi) $Transport : IA \times OA \to \{true, false\}$ *where* $\forall (p,t) \in IA \wedge (t',p') \in OA$ *such that* $Transport((p,t),(t',p'))$ *we require that* $t = t'$,
(vii) $Inv^T : IA \rightharpoonup \mathsf{Int}_{\geq \delta}$ *where we require* $Inv^T(a) = [\delta, \infty) \implies \exists a'.Transport(a, a')$
(viii) $Inv^P : P \to \mathsf{Int}_{\leq \Delta}$
(ix) Act is a set of labels (actions),
(x) $\lambda : T \to Act$ *is a labelling function,*
(xi) $Pe \subseteq P$ *is the set of pending places,*

and the following constraints:

1. *if* $Inhib((p,t))$ *then*

$$\{\neg Read((p,t))\} \wedge \{\neg Transport((p,t),(t,p'))|\forall p' \in P\}$$

2. *if* $Read((p,t))$ *then*

$$\{(t,p) \notin OA\} \wedge \{\neg Inhib((p,t))\} \wedge \{\neg Transport((p,t),(t,p'))|\forall p' \in P\}$$

3. *if* $Transport((p,t),(t,p'))$ *then* $\{\neg Read((p,t))\} \wedge \{\neg Inhib((p,t))\}$;

We only consider 1-bounded places in the present paper, which means that markings can be defined as simply a subset of places (the places containing a token).

Definition 7. ***(Marking)*** *Let N be a TAPNrp. A safe marking M on N is a subset $M \subseteq P$ of places. We say there is a token x at a place $p \in P$, written $x \in M(p)$, if $p \in M$. The set of all markings over N is denoted by $M(N)$. We define the token age as a function on places $M_t : P \to \mathcal{N}$. We say a token has age n written $M_t(p) = n$ where $n \in \mathcal{N}$.*

We say that a Petri Net is safe if the execution of transitions preserves the safeness of markings. In this paper we will work only with safe Petri Nets, in particular we prove that the mapping from timed DCR graphs to timed-arc Petri Nets provided in the next section always yields a safe timed-arc Petri Net.

Assuming the Petri Net to be safe simplifies the definition of enabledness of transitions defined as follows.

Definition 8. ***(Enabledness)*** *Let N be a $TAPNrp$. We say that a transition $t \in T$ is enabled in a marking M, if for $t \in T$ we have*

(i) $\{p \in P \mid (p,t) \in IA \wedge \neg Inhib((p,t) \wedge \neg Transport((p,t)(t,p'))\} \subseteq M$*, i.e. for all input arcs except the inhibitor and transport arcs there is a token in the input place,*

(ii) $\{p \in P \mid (p,t) \in IA \wedge Inhib((p,t)\} \cap M = \emptyset$*, i.e. for all inhibitor arcs there is* not *a token in the input place,*

(iii) $\{p \in P \mid M_t(p) \geq Inv^T((p,t)) \wedge M_t(p) \leq Inv^P(p') \forall Transport((p,t),(t,p'))\} \subseteq M$*, i.e. for all transport arcs the age of the token is greater or equal to the age guard on the input arc and less or equal to the age on the output place.*

We abuse notation and, just as for DCR graphs, let $\mathsf{enabled}(M,t)$ *denote that the transition t is enabled in marking M.*

Next we formalise the effect of executing (or firing) a transition. Again it is simplified by the assumption of safeness and we use the same notation as for DCR graphs to denote the result of firing a transition.

Definition 9. ***(Firing rule)*** *Let N be a $TAPNrp$, M a marking on N and $t \in T$ a transition. If* $\mathsf{enabled}(M,t)$ *with* $Input(t) = \{p \in P \mid (p,t) \in IA \wedge \neg Inhib((p,t)) \wedge \neg Read((p,t))\}$ *and* $Output(t) = \{p \in P \mid (t,p) \in OA\}$ *then t can fire, i.e. be executed, and produce a marking* $\mathsf{effect}_G(M,t) = (M \backslash Input) \cup Output$.

Definition 10. ***(Time Delay)*** *Let N be a $TAPNrp$, M a marking on N. A time delay $d \in \mathcal{N}$ for $d > 0$ is allowed in M if $(x+d) \in Inv^P(p) \forall p \in P \wedge x \in M_t(p)$, i.e. the delay of d time units does not violate any place invariants. In this case we write* $\mathsf{enabled}(M,d)$ *and refer to the delay as executing a time step d, reaching a new marking $M' =$* $\mathsf{effect}_G(M,d)$ *with the updated age on tokens defined as $M'_t(p) = M_t(p) + d \forall p \in P$.*

For convenience in the construction, we include the marking M and age on marking function M_t in the $TAPNrp$ tuple and we use

$$N_M = (P, T, A, Inhib, Read, Transport, Inv^T, Inv^P, Act, \lambda, Pe, M, M_t)$$

to refer to a safe marked $TAPNrp$ with marking $M \subseteq P$ and age on marking function M_t.

The firing rule and time delay define a timed labelled transition system for a $TAPNrp$ with markings as states and the labelled Petri Net transitions as labels.

Definition 11. *Let N_M be a $TAPNrp$ with marking M. Define a timed transition relation between markings by* $M \xrightarrow{\alpha}_N \mathsf{effect}_G(M,\alpha)$ *if* $\mathsf{enabled}(M,\alpha)$ *where $\alpha \in T \cup \mathcal{N}$. Write $M \Rightarrow M'$ for $\exists \alpha \in T \cup \mathcal{N}. M \xrightarrow{\alpha}_N M'$ and write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$. Define $\mathbb{M}_N = \{M' \mid M \Rightarrow^* M'\}$, i.e. the set of all reachable markings from the initial marking M of N. The timed labelled transition system for N is then defined as* $[[N]] = (\mathbb{M}_N, M, \rightarrow_N \subset \mathbb{M}_N \times (T \cup \mathcal{N}) \times \mathbb{M}_N), Act, \lambda)$.

Finally, we define when a finite or infinite execution sequence of a $TAPNrp$ is *accepting*.

Definition 12. *Let N be a $TAPNrp$ with safe marking M. A finite or infinite sequence of transitions or time delays $M_0 \xrightarrow{\alpha_0}_N M_1 \xrightarrow{\alpha_1}_N \ldots$ in $[[N]]$ is* accepting *if $p \in M_i \cap Pe$ implies $\exists j \geq i.\alpha_j = t_j \wedge (p, t_j) \in IA$. Moreover, if the sequence is infinite then there are infinitely many time steps.*

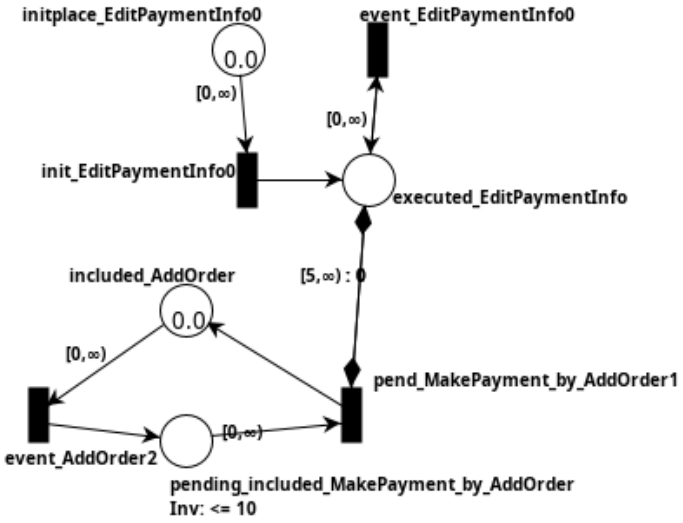


Fig. 2: E-shop Petri Net resulting from the transformation implementation.

Figure 2 shows the safe $TAPNrp$ resulting from the implemented optimized transformation of the running example DCR graph. The place pending_included_MakePayment_by_AddOrder is the only pending place and has the place invariant of 10 time units, representing the deadline until the transition pend_MakePayment_by_AddOrder1 must fire. The arcs between the transition pend_MakePayment_by_AddOrder1 and the place executed_EditPaymentInfo are transport arcs where the input arc only allows the transition to fire if the token in the place is at least 5 time units old.

As already noted in the introduction, the Petri Net need two transitions to represent the action Provide PaymentInfo, namely a transition init_EditPaymentInfo mapping the initial execution (or the initial entry of the payment information) and a transition event_EditPaymentInfo mapping subsequent executions.

4 Mapping Timed DCR graphs to Timed Arc Petri Nets

In this section, we establish a mapping from timed DCR graphs to marked safe Timed-arc Petri Nets with read arcs and pending places. We substantiate the

equivalence in transition semantics between the DCR graph and the Petri Net. Our implementation of this mapping is available as a Python script on GitHub[5] (together with examples). Furthermore, we export our net in the TAPN format, compatible with TAPAAL.

The mapping build upon our prior work [11], where we demonstrated the mapping for the untimed DCR Graphs case to Petri Nets with inhibitor, read arcs, and pending places, along with a proof of bisimilarity. Emphasizing the differences from the untimed version, we provide definitions and proofs for the timed extension, while maintaining the reasoning and approach from the prior work.

As part of the mapping $DP : DCR \to TAPNrp$, we define for every $G \in DCR$ a mapping $DPM_G : \mathbb{M}_G \to \mathbb{M}_{DP(G)}$, i.e. from markings of G to the markings of $DP(G)$. For a timed DCR graph $G = (E, M, R, @, L, l)$ and

$$\begin{aligned} DP(G) =&(P_{DP(G)}, M_{DP(G)}, T_{DP(G)}, A_{DP(G)}, Inhib_{DP(G)}, Read_{DP(G)}, \\ &Transport_{DP(G)}, Inv^T_{DP(G)}, Inv^P_{DP(G)}, Act_{DP(G)}, \lambda_{DP(G)}, Pe_{DP(G)}) \end{aligned}$$

we then have $M_{DP(G)} = DPM_G(M)$ and $Act_{DP(G)} = E$.

Definition 13. ***(Places mapping)*** *Let $G \in tDCR$. Define the corresponding Petri Net places of $DP(G)$ as $P_{DP(G)} = \{P^\gamma_e | e \in E, \gamma \in \{Ex, In\} \cup \{P^\gamma_{(e,e')} | e \in E, e' = \{\phi\} \cup e' \bullet\!\!\to e, \gamma \in \{Re, Rex\}\}$. Define the corresponding pending places of $DP(G)$ as $Pe_{DP(G)} = \{P^{Re}_{(e,e')} | e \in E, e' \in E\}$.*

Definition 14. ***(Response deadline mapping)*** *Let $G \in tDCR$. Define the corresponding Petri Net place invariants of $DP(G)$ as $Inv^P_{DP(G)}(p) = [0, d]$ where $p \in P^{Re}_{(e,e')}$ and $e \overset{d}{\bullet\!\!\to} e'$.*

The following mapping defines the invariant maintained between a marking M of a timed DCR graph and the corresponding marking $DPM_G(M)$ timed-arc Petri Net.

Definition 15. ***(Markings mapping)*** *Let $G \in tDCR$ and $\mathbb{M}_G$ be the timed markings of G and $\mathbb{M}_{DP(G)} = \mathcal{P}(P_{DP(G)})$, i.e. all safe markings of the places $P_{DP(G)}$ defined above. Define $DPM_G : \mathbb{M}_G \to \mathbb{M}_{DP(G)}$ as follows. For $M = (In, Ex, Re) \in \mathbb{M}_G$ define $DPM_G(M)$ such that for any event $e \in E$,*

(i) $P^{Ex}_e \in DPM_G(M) \wedge DPM_G(M)_t(P^{Ex}_e) = n \iff Ex(e) = n$
(ii) $P^{In}_e \in DPM_G(M) \iff e \in In$
(iii) $P^{Re}_{(e,e')} \in DPM_G(M) \wedge Inv^P(P^{Re}_{(e,e')}) = [0, d]$
$\iff Re(e) = d - DPM_G(M)_t(P^{Re}_{(e,e')}) \wedge e \in In$
(iv) $P^{Rex}_{(e,e')} \in DPM_G(M) \wedge Inv^P(P^{Re}_{(e,e')}) = [0, d]$
$\iff Re(e) = d - DPM_G(M)_t(P^{Re}_{(e,e')}) \wedge e \notin In$

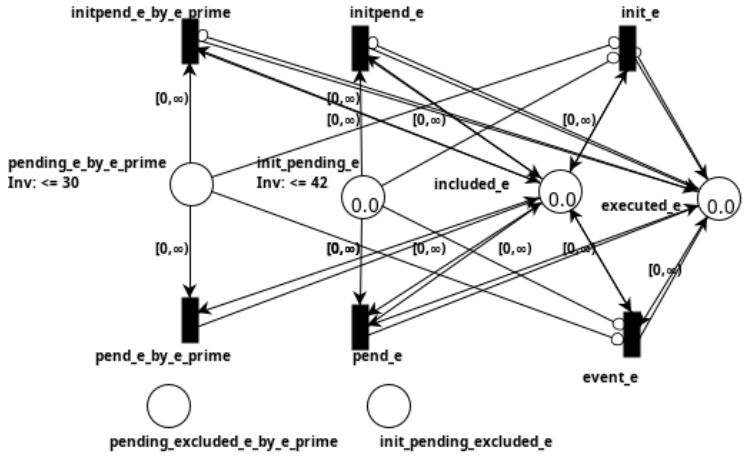


Fig. 3: Base case: Petri Net for a single included DCR event, which is not yet executed nor pending.

We shall see that by construction at most one of the set of places $\{P^{Re}_{(e,e')}\} \cup \{P^{Rex}_{(e,e')}\}$ can be marked for any given marking of $TAPNrp$.

We define an arc pattern table as a table with places as column headers and transitions as row indexes. Each cell defines the arcs mapped between the row index representing the transition and the column header representing the place. There can be 0, 1 or 2 arcs between each transition place pair. An empty cell represents no arcs, a single line drawn as "—" represents a read arc, a "<—" represents an input arc (p, t), a "—>" represents an output arc (t, p), "<—>" represents a pair of input and output arcs. Input and output transport arcs are drawn as "◇—" are "—◇" respectively. Transport input arcs with invariants are shown as "◇$\overset{k}{—}$" and "◇$\overset{k}{—}$◇". Inhibitor arcs are represented as "o—" and a pair of inhibitor and output arcs is shown as "o—>". When a pending Re or pending excluded Rex place has arcs to and from a transition we distinguish when the arc maps pairwise to the transition with the same label by adding a $<\!—_{(e,e')}$ subscript. When the subscript is not present it indicates that the transition maps to all Re or Rex places regardless of their subscript. Finally, we remark an important distinction between a read arc "—", a pair of input and output arcs "<—>" and a pair of transport arcs "◇$\overset{k}{—}$◇". Read arcs maintain token age, a pair of input and output arcs will reset the token age and a pair of transport arcs will maintain the token age, but also allow for an invariant to be set on the input transport arc.

The first arc pattern table is defined for the base structure of an event and can be seen in Table 1. The application of the arc pattern table for a given event e can be seen in Fig. 3. The event has an initial marking $e \in In, e \in Ex, e \in Re$

[5] `https://github.com/paul-cvp/pm4py-dcr/tree/feature/dcrtotapn`

and is initially pending with a deadline of 42 time units. Additionally e is affected by a response relation from another event e', i.e. $e' \overset{30}{\bullet\!\!\rightarrow} e$, which has a deadline of 30 time units.

Definition 16. ***(Base case: Mapping a timed DCR graph with no relation)*** *Let G be a tDCR Graph with no relations. Then $P_{DP(G)} = \{p_e^\delta | e \in E, \delta \in \{In, Ex\}\} \cup \{p^\delta_{(e,e')} | e \in E, e' = e \vee e' \bullet\!\!\rightarrow e, \delta \in \{Re, Rex\}\}$, $T_{DP(G)} = \{t_e^\delta | e \in E, \delta \in \{event, init\}\} \cup \{t^\delta_{(e,e')} | e \in E, e' = \{\phi\} \cup e' \bullet\!\!\rightarrow e, \delta \in \{pend, initpend\}\}$ and $\lambda_{DP(G)}(t_e^\delta) = e$. The set of arcs $A_{DP(G)} = IA_{DP(G)} \cup OA_{DP(G)}$, $Inhib_{DP(G)} : IA_{DP(G)} \to \{true, false\}$, $Read_{DP(G)} : IA_{DP(G)} \to \{true, false\}$ and $Transport_{DP(G)} : IA_{DP(G)} \times OA_{DP(G)} \to \{true, false\}$ are defined by Table 1.*

Definition 16 and the first row of Table 1 tell us that for each event $e \in E$ we create one transition labelled t_e^{event} that fires when there is a token in the places p_e^{In} and p_e^{Ex} and no token in all $p^{Re}_{(e,e')}$ places. This is equivalent to $e \in In, Ex(e) = k$ and $Re(e)$ is undefined for the marking of the timed DCR Graph G. The only effect t_e^{event} has on the marking is to reset the age of the token on p_e^{Ex}. This corresponds to the timed DCR Graph semantics setting the execution time $Ex(e) = 0$ when e is executed. The second row shows that we create another transition labelled t_e^{init} that can fire (and then puts one token in p_e^{Ex}) if and only if there is a token in p_e^{In} and no token in p_e^{Ex} and all $p^{Re}_{(e,e')}$. This corresponds to the marking $e \in In$ and both $Ex(e)$ and $Re(e)$ undefined in G. The third row in Table 1 tells us that we have to create as many copies of $t^{pend}_{(e,e')}$ for as many e' we defined, i.e. $e' = \{\phi\} \cup e' \bullet\!\!\rightarrow e$. $\{\phi\}$ represents the initially pending place which can be marked if $e \in Re$ in the initial marking of G, the remaining e' represent the set of events which have a response relation to e in G. Notice that these arcs are pairwise mapped to their equivalently labelled places. This can be seen Fig. 3 between the transitions labelled **pend_e**, **pend_e_by_e_prime** and their respective places **init_pending_e**, **pending_e_by_e_prime**. The effect of firing any $t^{pend}_{(e,e')}$ is to remove the token from their respective pending place. The last set of transitions $t^{initpend}_{(e,e')}$ follow the same reasoning as $t^{pend}_{(e,e')}$ only it has the additional effect of adding a token to p_e^{Ex}.

e	p_e^{In}	p_e^{Ex}	$p^{Re}_{(e,e')}$	$p^{Rex}_{(e,e')}$
t_e^{event}	—	<—>	o—	
t_e^{init}	—	o—>	o—	
$t^{pend}_{(e,e')}$	—	<—>	$<\!\!-_{(e,e')}$	
$t^{initpend}_{(e,e')}$	—	o—>	$<\!\!-_{(e,e')}$	

Table 1: Arc patterns for an event

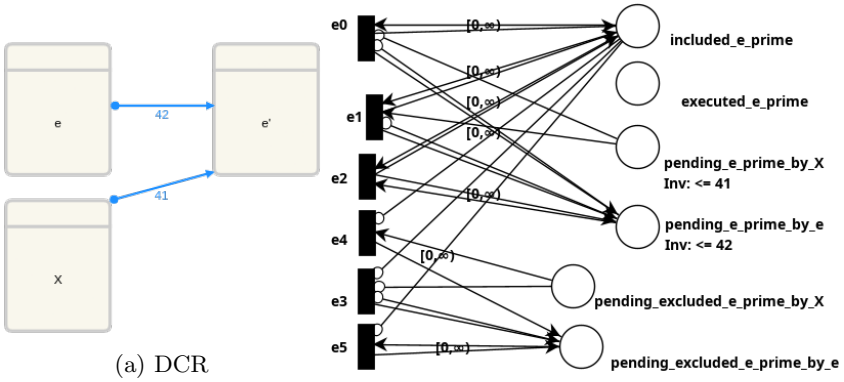
After mapping the TAPNrp for each event $e \in E$ we proceed to map each relation $r \in R$. Let $G = (E, M, R, @, L, l)$ be a DCR Graph with $R = \{r_1, \ldots, r_k, r_{k+1}\}$. We first consider the cases where the relation $r_{k+1} = (e, e')$ is a single relation between distinct events, i.e. $@r_{k+1} \in \{\{\rightarrow\!\!+\}, \{\rightarrow\!\!\%\}, \{\overset{d}{\bullet\!\!\rightarrow}\}, \{\overset{k}{\bullet\!\!-}\}\}$ and $e \neq e'$.

Definition 17. ***(Mapping a DCR relation)***

Let $G = (E, M, R, @, L, l)$ be a DCR Graph, $APT(@r)$ be the arc pattern table for any given relation $r \in @R$ and $|APT(@r)_r|$ be the number of rows (transition copies) in the arc pattern table. Then $T_{DP(G_{k+1})} = T_{DP(G_k)} \backslash \{t_e^\delta \mid t_e^\delta \in T_{DP(G_k)}$ and $\lambda_{DP(G_k)}(t_e^\delta) = e\} \cup \{t_e^{i,\delta} \mid i \in \{0, 1, .., |APT(@r)_r| - 1\}\}$ and let $(p, t_e^{i,\delta}) \in IA_{DP(G_{k+1})}$ for $i \in \{0, 1.., |APT(@r)_r| - 1\}$ if and only if $(p, t_e^\delta) \in IA_{DP(G_k)}$ and let $(t_e^{i,\delta}, p) \in OA_{DP(G_{k+1})}$ for $i \in \{0, 1.., |APT(@r)_r| - 1\}$ if and only if $(t_e^\delta, p) \in OA_{DP(G_k)}$ or $(t_e^{i,\delta}, p)$ is one of the arcs in $APT(@r)$. Finally, for $t \in A_{DP(G_k)}$ such that $\lambda_{DP(G_k)} \neq e$, let $(p, t) \in IA_{DP(G_{k+1})}$ and $(t, p) \in OA_{DP(G_{k+1})}$ if and only if $(p, t) \in IA_{DP(G_k)}$ or $(t, p) \in OA_{DP(G_k)}$.

The set of arcs $A_{DP(G)} = IA_{DP(G)} \cup OA_{DP(G)}$, $Inhib_{DP(G)} : IA_{DP(G)} \to \{true, false\}$, $Read_{DP(G)} : IA_{DP(G)} \to \{true, false\}$ and $Transport_{DP(G)} : IA_{DP(G)} \times OA_{DP(G)} \to \{true, false\}$ are defined according to $APT(@r)$.

For each relation we define its arc pattern table $APT(@r)$ in Table 2. Notice that we are now referring to sets of existing transitions in the row indexes of the tables. Intuitively, Definition 17 tells us that for any given relation $e@r_k e'$ we take *only* the set of existing transitions labelled with e together with the set of existing arcs to and from the transitions and create as many copies of these transitions and arcs as there are rows in the arc pattern table $APT(@r_k)$ for the given relation. Then for each set of transitions we copy, we add the new arcs as defined in the arc pattern table $APT(@r_k)$. We then remove the original set of transitions together with their arcs and keep only the new copies. When we move to the next relation $e@r_{k+1}e'$ we apply the same definition again *only* looking at the set of existing transitions labelled with e and their arcs.



(a) DCR

(b) Petri Net

Fig. 4: Mapping (a) DCR timed response relation to a Petri Net notation (b)

Example 1. (Mapping the timed response) Figure 4 shows how the response relation $e \overset{d}{\bullet\!\!\rightarrow} e'$ is mapped by replacing the set of existing transition T_e representing

$\rightarrow\!\!\!\!+$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',\{e\}\cup e'')}$	$p^{Rex}_{(e',\{e\}\cup e'')}$
$t^{0,\delta}_e \in T_e$	—			
$t^{1,\delta}_e \in T_e$	o—>		$\diamond$—$_{(e',e'')}$	—$\diamond_{(e',e'')}$
$t^{2,\delta}_e \in T_e$	o—>			o—

(a) Arc patterns for $\rightarrow\!\!\!\!+$

$\rightarrow\!\!\!\!\%$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',\{e\}\cup e'')}$	$p^{Rex}_{(e',\{e\}\cup e'')}$
$t^{0,\delta}_e \in T_e$	o—			
$t^{1,\delta}_e \in T_e$	<—		o—	
$t^{2,\delta}_e \in T_e$	<—		$\diamond$—$_{(e',e'')}$	—$\diamond_{(e',e'')}$

(b) Arc patterns for $\rightarrow\!\!\!\!\%$

$\bullet\!\!\rightarrow^{d}$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	—		o—>		o—	
$t^{1,\delta}_e \in T_e$	—		—>$_{(e',e'')}$		<—$_{(e',e'')}$	
$t^{2,\delta}_e \in T_e$	—		<—>			
$t^{3,\delta}_e \in T_e$	o—			o—>		o—
$t^{4,\delta}_e \in T_e$	o—			—>$_{(e',e'')}$		<—$_{(e',e'')}$
$t^{5,\delta}_e \in T_e$	o—			<—>		

(c) Arc patterns for $\bullet\!\!\rightarrow^{d}$

$\bullet\!\!\leftarrow^{k}$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',\{e\}\cup e'')}$	$p^{Rex}_{(e',\{e\}\cup e'')}$
$t^{0,\delta}_e$	—	$\diamond\overset{k}{—}\diamond$		
$t^{1,\delta}_e$	o—			

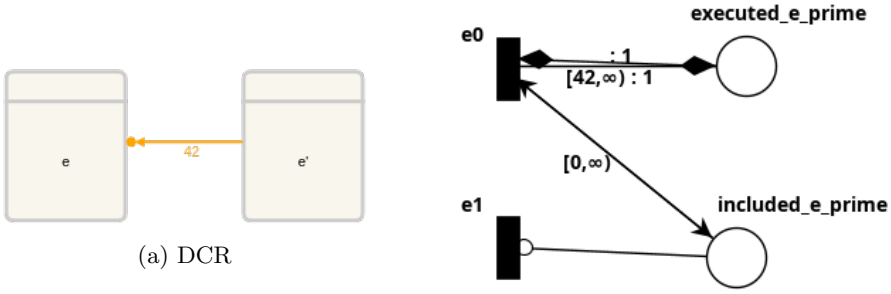
(d) Arc patterns for $\bullet\!\!\leftarrow^{k}$

Table 2: Arc patterns for $r_{k+1} = (e, e')$ single relations

e by six new copies, connected to the places representing the event e'. Notice that in the arc pattern table 2c $APT(\overset{d}{\bullet\!\!\rightarrow})$ of the timed response we distinguish specifically when mapping arcs between the pending place of e' on which e has an effect on and all the other pending places on which other events e'' have an effect on (e'' also includes the initially pending place).

Note that we have already captured the response deadline of 42 time units in the place invariant of **pending_e_prime_by_e** ($p^{Re}_{(e',e)}$). The Figure 4 also shows how a second response from another event X with a deadline of 41 is handled from the point of view of the current relation $e \overset{42}{\bullet\!\!\rightarrow} e'$.

If the event e' is also affected by $\rightarrow\!\!+$ or $\rightarrow\!\!\%$ relations from other events e'', the respective inclusion and exclusion events will transport the tokens between the $p^{Re}_{(e',e)}$ and $p^{Rex}_{(e',e)}$ as defined in row $t^{1,\delta}_{e}$ of the arc pattern table 2a and row $t^{2,\delta}_{e}$ of the arc pattern table 2b respectively.



(a) DCR

(b) Petri Net

Fig. 5: Mapping (a) DCR timed condition relation to a Petri Net notation (b)

Example 2. (Mapping a timed condition) Figure 5 shows how a condition relation is mapped between the transitions representing the DCR event e and the places representing the execution and inclusion marking for the DCR event e'. The delay on the condition is mapped as the invariant on the input transport arc between **e0** and **executed_e_prime**.

Handling exceptions. Timed DCR Graphs allow for multiple relations to exist between the same events as well as one or more relations between an event and itself. The behaviour of the compound relations and of the self relations is different than the execution semantics of single relations between different events. The exceptional cases are the same as for the untimed mapping from DCR to $PNirp$ described in [11]. The only changes in the exceptions for the timed mapping are observed in the new arc pattern tables from Table 3. Note that we still apply Definition 17 on these new tables. All other exceptions not

defined in Table 3 either are the same as for the untimed mapping or reduce to the ones already defined.

We are now ready to extend the bisimilarity Theorem 1 from [11] to a bisimulation between the timed DCR Graph and the mapped Timed-arc Petri Net with read arcs and pending places. The proof follows very much the same structure as, except we also have to show that the time invariants between markings in the DCR graph and the corresponding Petri net are also preserved. In [11] we showed how the strong bisimilarity property is preserved by construction in the induction steps, here we simply refer to the new TAPNrp construction Definitions 13 to 17 to show that the strong timed bisimilarity property holds.

Theorem 1. ***(Strong timed bisimilarity)*** *For $G \in tDCR$ we have that the relation $Sim_G = \{(M, DPM_G(M)) \mid M \in \mathbb{M}_G\}$ is a bisimulation relation between $[[G]]$ and $[[DP(G)]]$, in the sense that $(M_0, DPM_G(M_0)) \in Sim_G$, where M_0 is the initial marking of G and for all $(M, DPM_G(M)) \in Sim$, we have*

(i) $M \xrightarrow{e} M'$ *implies* $\exists! t \in T_{DP(G)}. DPM_G(M) \xrightarrow{t} DPM_G(M')$ *and* $\lambda(t) = e$,
(ii) $DPM_G(M) \xrightarrow{t} M'$ *and* $\lambda(t) = e$ *implies* $M \xrightarrow{e} M''$ *and* $DPM_G(M'') = M'$,
(iii) $M \xrightarrow{n} M'$ *implies* $DPM_G(M) \xrightarrow{n} DPM_G(M')$
(iv) $DPM_G(M) \xrightarrow{n} M'$ *implies* $M \xrightarrow{n} M''$ *and* $DPM_G(M'') = M'$.

What remains to show is that the accepting runs in the two models are the same. This follows easily from the correspondence of markings in Def. 15, which is maintained by the bisimulation relation.

Proposition 1. *For a DCR Graph DCR graph $G = G_0 = (E, M, R, @, L, l)$ it holds that an execution sequence $M \to M_1 \to M_2 \to ..$ is accepting if and only if $DPM_G(M) \to DPM_G(M_1) \to DPM_G(M_2) \to ..$ is accepting.*

5 Conclusion and future work

We presented a transformation from timed Dynamic Condition Response (DCR) graph constraint based process specification language to safe timed-arc Petri Nets with read arcs, generalized with an acceptance criteria for the modelling of ω-regular liveness properties. The mapping extends previous work [11] providing a mapping from untimed DCR graphs to Petri Nets. The difficulties in the extension was in setting up the mapping of the constraints. The proof for strong bisimilarity between the transition system for the timed DCR graph and the transition system for the resulting timed-arc Petri Net follows the same structure as the proof in [11], except that it includes a time invariance between the marking of the timed DCR Graph and the timed-arc Petri Net. The mapping is supported by an open source prototype implementation available in GitHub at `https://github.com/paul-cvp/pm4py-dcr/tree/feature/dcrtotapn` (together with examples).

Given the existence of end-user no-code design tools for timed DCR Graphs, the mapping provides a novel pathway for domain experts who are not familiar

$\overset{k}{\bullet\!\!\leftarrow} \wedge \rightarrow\!\!\!+$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',\{e\}\cup e'')}$	$p^{Rex}_{(e',\{e\}\cup e'')}$
$t^{0,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$		
$t^{1,\delta}_e \in T_e$	o—>		—$\diamond_{(e',e'')}$	$\diamond$—$_{(e',e'')}$
$t^{2,\delta}_e \in T_e$	o—>			o—

(a) Arc patterns for $e' \overset{k}{\bullet\!\!\leftarrow} e \wedge e \rightarrow\!\!\!+ e'$

$\overset{k}{\bullet\!\!\leftarrow} \wedge \rightarrow\!\!\!\!\!\!\%$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',\{e\}\cup e'')}$	$p^{Rex}_{(e',\{e\}\cup e'')}$
$t^{0,\delta}_e \in T_e$	<—	$\diamond\overset{k}{—}\diamond$	o—	
$t^{1,\delta}_e \in T_e$	<—	$\diamond\overset{k}{—}\diamond$	$\diamond$—$_{(e',e'')}$	—$\diamond_{(e',e'')}$
$t^{2,\delta}_e \in T_e$	o—			

(b) Arc patterns for $e' \overset{k}{\bullet\!\!\leftarrow} e \wedge e \rightarrow\!\!\!\!\!\!\% e'$

$\overset{k}{\bullet\!\!\leftarrow} \wedge \overset{d}{\bullet\!\!\rightarrow}$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	o—>		o—	
$t^{1,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	<—>			
$t^{2,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	—>$_{(e',e'')}$		<—$_{(e',e'')}$	
$t^{3,\delta}_e \in T_e$	o—			—>$_{(e',e'')}$		<—$_{(e',e'')}$
$t^{4,\delta}_e \in T_e$	o—			o—>		o—
$t^{5,\delta}_e \in T_e$	o—			<—>		

(c) Arc patterns for $e' \overset{k}{\bullet\!\!\leftarrow} e \wedge e \overset{d}{\bullet\!\!\rightarrow} e'$

$\overset{d}{\bullet\!\!\rightarrow} \wedge \rightarrow\!\!\!+$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	—		o—>		o—	
$t^{1,\delta}_e \in T_e$	—		—>$_{(e',e'')}$		<—$_{(e',e'')}$	
$t^{2,\delta}_e \in T_e$	—		<—>			
$t^{3,\delta}_e \in T_e$	o—>		o—>	o—		o—
$t^{4,\delta}_e \in T_e$	o—>		—>	<—		
$t^{4,\delta}_e \in T_e$	o—>		—>$_{(e',e'')}$			<—$_{(e',e'')}$

(d) Arc pattern for $e \overset{d}{\bullet\!\!\rightarrow} e' \wedge e \rightarrow\!\!\!+ e'$

$\overset{d}{\bullet\!\!\rightarrow} \wedge \rightarrow\!\!\!\!\!\!\%$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	<—		o—	—>	o—	
$t^{1,\delta}_e \in T_e$	<—		<—	—>		
$t^{2,\delta}_e \in T_e$	<—			—>$_{(e',e'')}$	<—$_{(e',e'')}$	
$t^{3,\delta}_e \in T_e$	o—			o—>		o—
$t^{4,\delta}_e \in T_e$	o—			—>$_{(e',e'')}$		<—$_{(e',e'')}$
$t^{5,\delta}_e \in T_e$	o—			<—>		

(e) Arc pattern for $e \overset{d}{\bullet\!\!\rightarrow} e' \wedge e \rightarrow\!\!\!\!\!\!\% e'$

$\bullet\!\!\leftarrow \wedge \overset{d}{\bullet\!\!\rightarrow} \wedge \rightarrow\!\!\!+$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	o—>		o—	
$t^{1,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	<—>			
$t^{2,\delta}_e \in T_e$	—	$\diamond\overset{k}{—}\diamond$	—>$_{(e',e'')}$		<—$_{(e',e'')}$	
$t^{3,\delta}_e \in T_e$	o—>		—>$_{(e',e'')}$			<—$_{(e',e'')}$
$t^{4,\delta}_e \in T_e$	o—>		—>	o—		o—
$t^{5,\delta}_e \in T_e$	o—>		—>	<—		

(f) Arc pattern for $e' \bullet\!\!\leftarrow e \wedge e \overset{d}{\bullet\!\!\rightarrow} e' \wedge e \rightarrow\!\!\!+ e'$

$\bullet\!\!\leftarrow \wedge \overset{d}{\bullet\!\!\rightarrow} \wedge \rightarrow\!\!\!\!\!\!\%$	$p^{In}_{e'}$	$p^{Ex}_{e'}$	$p^{Re}_{(e',e)}$	$p^{Rex}_{(e',e)}$	$p^{Re}_{(e',e'')}$	$p^{Rex}_{(e',e'')}$
$t^{0,\delta}_e \in T_e$	<—	$\diamond\overset{k}{—}\diamond$	o—	—>	o—	
$t^{1,\delta}_e \in T_e$	<—	$\diamond\overset{k}{—}\diamond$	<—	—>		
$t^{2,\delta}_e \in T_e$	<—	$\diamond\overset{k}{—}\diamond$		—>$_{(e',e'')}$	<—$_{(e',e'')}$	
$t^{3,\delta}_e \in T_e$	o—			—>$_{(e',e'')}$		<—$_{(e',e'')}$
$t^{4,\delta}_e \in T_e$	o—			o—>		o—
$t^{5,\delta}_e \in T_e$	o—			<—>		

(g) Arc pattern for $e' \bullet\!\!\leftarrow e \wedge e \overset{d}{\bullet\!\!\rightarrow} e' \wedge e \rightarrow\!\!\!\!\!\!\% e'$

Table 3: Exception relation arc patterns

with formal models to map timed specifications expressed in natural language to timed-arc Petri Nets, that can be formally verified in e.g. the TAPAAL tool. Indeed, the version of the DCR Solutions design tool released December 29, 2023 included a ChatGPT enabled interface, allowing a user to write rules in natural language and get the corresponding timed DCR graph constraint. The mapping algorithm was also tested on a larger example of a railroad controller given in [35], modeling a railroad crossing with two tracks with trains arriving independently of each other. While the timed DCR Graph consists of 16 activities and 66 relations, the corresponding timed-arc Petri Net consists of 34 places, 99 transitions and 1200 arcs.

We believe the work in the present paper provides a plethora of research avenues, which we aim to explore in future work. Firstly, we plan to evaluate the time and space complexity of the mapping. We also plan evaluate the result of mining timed DCR graphs using a prototype timed extension of the DisCoveR [5] miner to mine timed DCR Graphs from well-known, real-life, public event logs and map these to their timed-arc Petri Net counter parts.

As seen by our running example, the mapping nicely captures concurrency between independent events. This could potentially also be combined with a mapping from Petri Nets to BPMN [13], to provide an output following an ISO standard process notation. Finally we aim to integrate the existing mapping from safe Timed Arc Petri Nets to Timed Automata [37] to provide a link from timed DCR Graphs to Timed Automata.

References

[1] Wil M.P van der Aalst and Maja Pesic. "DecSerFlow: Towards a Truly Declarative Service Flow Language". In: *Proceedings of Web Services and Formal Methods (WS-FM 2006)*. Ed. by M. Bravetti, M. Nunez, and Gianluigi Zavattaro. Vol. 4184. 2006, pp. 1–23.

[2] Lars Ackermann et al. "Execution of Multi-perspective Declarative Process Models". In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. Ed. by Hervé Panetto et al. Cham: Springer International Publishing, 2018, pp. 154–172. ISBN: 978-3-030-02671-4.

[3] T Agerwala. "A complete model for representing the coordination of asynchronous processes". In: (1974). Hopkins Computer Research Report 32.

[4] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90010-8. URL: https://www.sciencedirect.com/science/article/pii/0304397594900108.

[5] C.O. Back et al. "DisCoveR: accurate and efficient discovery of declarative process models". In: *Int Journal of Software Tools Technology Transfer* 24 (2022), pp. 563–587.

[6] Paolo Baldan et al. "Functional concurrent semantics for petri nets with read and inhibitor arcs". In: *CONCUR 2000—Concurrency Theory: 11th International Conference University Park, PA, USA, August 22–25, 2000 Proceedings 11*. Springer. 2000, pp. 442–457.

[7] David Basin et al. "Monitoring metric first-order temporal properties". In: *Journal of the ACM (JACM)* 62.2 (2015), pp. 1–45.

[8] Bernard Berthomieu and Miguel Menasche. "A state enumeration approach for analyzing time Petri nets". In: *3rd European Workshop on Applications and Theory of Petri Nets*. 1982.

[9] Andrea Burattin, Fabrizio M. Maggi, and Alessandro Sperduti. "Conformance checking based on multi-perspective declarative process models". In: *Expert Systems with Applications* 65 (2016), pp. 194–211. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.08.040.

[10] Joakim Byg, Kenneth Yrke Jørgensen, and Jiří Srba. "An Efficient Translation of Timed-Arc Petri Nets to Networks of Timed Automata". In: *Formal Methods and Software Engineering*. Ed. by Karin Breitman and Ana Cavalcanti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 698–716. ISBN: 978-3-642-10373-5.

[11] Vlad Paul Cosma, Thomas T. Hildebrandt, and Tijs Slaats. "Transforming Dynamic Condition Response Graphs to Safe Petri Nets". In: *Application and Theory of Petri Nets and Concurrency: 44th International Conference, PETRI NETS 2023, Lisbon, Portugal, June 25–30, 2023, Proceedings*. Lisbon, Portugal: Springer-Verlag, 2023, pp. 417–439. ISBN: 978-3-031-33619-5. DOI: 10.1007/978-3-031-33620-1_22. URL: https://doi.org/10.1007/978-3-031-33620-1_22.

[12] Johannes De Smedt et al. "A full R/I-net construct lexicon for declare constraints". In: *Available at SSRN 2572869* (2015).

[13] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. "Formal semantics and analysis of BPMN process models using Petri nets". In: *Queensland University of Technology, Tech. Rep* (2007), pp. 1–30.

[14] Rik Eshuis et al. "Deriving Consistent GSM Schemas from DCR Graphs". In: *Service-Oriented Computing*. Ed. by Quan Z. Sheng et al. Cham: Springer International Publishing, 2016, pp. 467–482. ISBN: 978-3-319-46295-0.

[15] Olivier Finkel. "On the High Complexity of Petri Nets ω-Languages". In: *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer. 2020, pp. 69–88.

[16] Thomas Hildebrandt et al. "Contracts for cross-organizational workflows as timed Dynamic Condition Response Graphs". In: *The Journal of Logic and Algebraic Programming* 82.5 (2013). Formal Languages and Analysis of Contract-Oriented Software (FLACOS'11), pp. 164–185. ISSN: 1567-8326. DOI: https://doi.org/10.1016/j.jlap.2013.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1567832613000283.

[17] Thomas T. Hildebrandt and Raghava Rao Mukkamala. "Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs". In: *PLACES*. 2010, pp. 59–73.

[18] Thomas T. Hildebrandt et al. "Contracts for cross-organizational workflows as timed Dynamic Condition Response Graphs". In: *Journal of Logic and Algebraic Programming* 82.5-7 (2013), pp. 164–185.

[19] Thomas T. Hildebrandt et al. "Decision Modelling in Timed Dynamic Condition Response Graphs with Data". In: *Business Process Management Workshops - BPM 2021 International Workshops, Rome, Italy, September 6-10, 2021, Revised Selected Papers*. Ed. by Andrea Marrella and Barbara Weber. Vol. 436. Lecture Notes in Business Information Processing. Springer, 2021, pp. 362–374. DOI: 10.1007/978-3-030-94343-1_28. URL: https://doi.org/10.1007/978-3-030-94343-1%5C_28.

[20] Zhaoxia Hu and Sol M Shatz. "Mapping UML Diagrams to a Petri Net Notation for System Simulation." In: *SEKE*. Citeseer. 2004, pp. 213–219.

[21] Richard Hull et al. "Introducing the guard-stage-milestone approach for specifying business entity lifecycles". In: *Proc. of WS-FM'10*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 1–24. ISBN: 978-3-642-19588-4.

[22] Lasse Jacobsen et al. "Verification of Timed-Arc Petri Nets". In: *SOFSEM 2011: Theory and Practice of Computer Science*. Ed. by Ivana Černá et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 46–72. ISBN: 978-3-642-18381-2.

[23] Fabrizio Maria Maggi et al. "Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata". In: *Business Process Management*. Ed. by Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 132–147. ISBN: 978-3-642-23059-2.

[24] Marco Montali. *Specification and verification of declarative open interaction models: a logic-based approach*. Vol. 56. Springer Science & Business Media, 2010.

[25] Marco Montali et al. "Declarative Specification and Verification of Service Choreographiess". In: *ACM Trans. Web* 4.1 (Jan. 2010). ISSN: 1559-1131. DOI: 10.1145/1658373.1658376. URL: https://doi.org/10.1145/1658373.1658376.

[26] Ugo Montanari and Francesca Rossi. "Contextual nets". In: *Acta Informatica* 32 (1995), pp. 545–596.

[27] Raghava Rao Mukkamala. "A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs". PhD thesis. IT University of Copenhagen, June 2012.

[28] Raghava Rao Mukkamala and Thomas T. Hildebrandt. "From Dynamic Condition Response Structures to Büchi Automata". In: *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*. 2010, pp. 187–190. DOI: 10.1109/TASE.2010.22.

[29] Håkon Norman et al. "Zoom and Enhance: Action Refinement via Subprocesses in Timed Declarative Processes". In: *Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings*. Ed. by Artem Polyvyanyy et al. Vol. 12875. Lecture Notes in Computer Science. Springer, 2021, pp. 161–178. DOI: 10.1007/

978-3-030-85469-0_12. URL: https://doi.org/10.1007/978-3-030-85469-0%5C_12.

[30] Amir Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[31] Viara Popova and Marlon Dumas. "From Petri Nets to Guard-Stage-Milestone Models". In: *Business Process Management Workshops*. Ed. by Marcello La Rosa and Pnina Soffer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 340–351. ISBN: 978-3-642-36285-9.

[32] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. "From Declarative Processes to Imperative Models." In: *SIMPDA* 1293 (2014), pp. 162–173.

[33] Ivo Raedts et al. "Transformation of BPMN Models for Behaviour Analysis." In: *MSVVEIS* 2007 (2007), pp. 126–137.

[34] V Valero Ruiz, David de Frutos Escrig, and F Cuartero Gomez. "On non-decidability of reachability for timed-arc Petri nets". In: *Proceedings 8th International Workshop on Petri Nets and Performance Models (Cat. No. PR00331)*. IEEE. 1999, pp. 188–196.

[35] Krzysztof Sacha. "Verification and Implementation of Dependable Controllers". In: *2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*. 2008, pp. 143–151. DOI: 10.1109/DepCoS-RELCOMEX.2008.30.

[36] Tijs Slaats. "Flexible Process Notations for Cross-organizational Case Management Systems". PhD thesis. IT University of Copenhagen, Jan. 2015.

[37] Jiří Srba. "Timed-Arc Petri Nets vs. Networks of Timed Automata". In: *Applications and Theory of Petri Nets 2005*. Ed. by Gianfranco Ciardo and Philippe Darondeau. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 385–402. ISBN: 978-3-540-31559-9.

[38] Tony Spiteri Staines. "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets". In: *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*. 2008, pp. 191–200. DOI: 10.1109/ECBS.2008.12.

[39] D. Thapa, S. Dangol, and Gi-Nam Wang. "Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique". In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. Vol. 2. 2005, pp. 228–233. DOI: 10.1109/CIMCA.2005.1631473.

[40] Rüdiger Valk. "Infinite behaviour of Petri nets". In: *Theoretical computer science* 25.3 (1983), pp. 311–341.

[41] Nianhua Yang et al. "Modeling UML Sequence Diagrams Using Extended Petri Nets". In: *2010 International Conference on Information Science and Applications*. 2010, pp. 1–8. DOI: 10.1109/ICISA.2010.5480384.

[42] Dmitry A. Zaitsev. "Toward the Minimal Universal Petri Net". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.1 (2014), pp. 47–58. DOI: 10.1109/TSMC.2012.2237549.