KU LEUVEN

FACULTY OF ECONOMICS
AND BUSINESS

Studies on Declarative Process Modeling and Its Relation to Procedural Techniques

Dissertation presented to
obtain the degree of Doctor
in Applied Economics

by

Johannes DE SMEDT

Number 547 December 2016

Since the theses in the series published by the Faculty of Economics and Business are the personal work of their authors, only the latter bear full responsibility.

Committee

Chairperson	Prof. dr. Martina Vandebroek	KU Leuven
Promotor	Prof. dr. Jan Vanthienen	KU Leuven
Co-Promotor	Prof. dr. Geert Poels	Ghent University
	Prof. dr. Jochen De Weerdt	KU Leuven
	Prof. dr. Jan Mendling	WU Vienna
	Prof. dr. ir. Hajo A. Reijers	VU Amsterdam
	Prof. dr. Seppe K.L.M. vanden Broucke	KU Leuven

Acknowledgments

Expressing gratitude should not be taken lightly. Therefore it is of no surprise that the minutes spent on any word in this section is higher than elsewhere in this booklet, for as soon as the end of this dissertation was clearly in sight this section started to take shape in my mind. I hope I convey my feelings correctly, and that all of you can recognize yourselves in my words of appreciation.

First of all, I would like to thank my promoter professor Vanthienen, Jan, for granting me the opportunity to commence this journey comprised of many aspects. You have this vast trunk of experience without taking up the role of your typical 'dinosaur', a term which you loathe to be associated with. It renders you able to see the bigger picture, a concept that is often still opaque for a (beginning) PhD student. Thank you for inciting the research flame within me, guiding me towards the declarative side of process modeling, and supporting me along the way. Furthermore you became a friend, one who had many advises on, amongst others, overcoming insomnia with the appropriate means. I hope that there are still many more years to come so you can finally prove that every 30 years Information Systems research repeats itself.

Next, I would like to thank three people that became especially important throughout my PhD track. Allow me to be polite and start with the ladies first. Dr. Serral, Estefanía, your arrival in Leuven coincided with my starting as a researcher and I had the pleasure to share an office with you for more than three years. Meanwhile we have laughed together, performed research on the most beautiful of process modeling languages, Colored Petri nets, got frustrated together (mainly due to CPNs though), and went through different important phases of life. You became mother of the cutest baby, and I hope your future is bright and filled with wonderful moments together with Líam and Mathijs. Professor vanden Broucke, Seppe[1], an office mate

[1] the main inspiration for my using the word 'plethora'[2]

[2] and the only person who would ever consider using a footnote inside a footnote

for one year and a big influence during my PhD, I also cherish fond memories of working and general roaming around together during the past three years. To me, you seem to embody the role model of a researcher: a sharp, fast, and knowledgeable mind, supplemented by a decent portion of humor and self-reflection. I also cannot get past the tremendous input and friendship I received from professor De Weerdt, Jochen, who has been such a great supporter and advisor. You helped me get my head around the business process literature and introduced me to the community, and in the end wound up in an area which was not familiar to you. This, however, did not prevent you from investing a lot of time in my endeavors nonetheless, which has allowed me to perform research of at least decent quality. I am sure that many more students will be able to benefit from this gracious attitude in the future.

Furthermore, I would like to express my gratitude to the other faculty of LIRIS. Professor Baesens, Bart, a true pizza connoisseur (although this applies to anyone at LIRIS), for providing some vital tips on performing research. I enjoyed working with you on the Master's thesis projects and sharing many administrative frustrations, and I cannot thank you enough for tipping me off on the position in Edinburgh which I will take up next year. Professor Snoeck, Monique, for providing a strong feminine spirit to LIRIS, for the numerous intense discussions on the wonderful experience that is teaching BIS, for helping me out with (Petri net) languages, and for being a kind person in general. Professor Put, Ferdi, for always being interested in a chat on the newest developments in hardware and software, and for teaching the most interesting Information Systems Engineering courses.

Finally, I am also grateful for the support of the external committee members. My co-promoter, professor Poels, Geert, you provided invaluable feedback throughout my PhD which often surpassed my general understanding of the topic and helped me position and elaborate my work on a higher level. Professor Mendling, Jan, thank you for providing me with the most insightful and comprehensive comments to my work. Your knowledge of the business process field is remarkable, and shows what a great scholar you are. I also very much enjoyed my time in Vienna, where you lead a very hospitable group of incredible researchers, Claudio, Andreas, Jonas, Cristina, Monika, Johannes, Saimir, Giray, and many others, who all helped to make my stay unforgettable. Professor Reijers, Hajo, who boasts the same business process skills as professor Mendling, I would like to thank for your input during the later stages of my PhD. Your recommendations helped making the scope of this document more profound.

The LIRIS family has seen many changes in lineup and, while I was the only one who started in 2013, knew an enormous surge in the number of PhD researchers as of 2014. Therefore I split my acknowledgments in two parts: pre-, and post-boom. The wonderful people of the pre-boom period comprise of a collection of persons that set me on my way and introduced me to the general practices of performing research. First of all my colleague as a Bachelor and Master student Information Systems Engineering, Véronique, who I have known probably the longest of everyone. You are one of the most complete researchers I have come across and radiate an attitude that makes research seem straightforward though urgent. Secondly, I would like to thank Alex, for being the inventor of the coffee ritual and being an uttermost enjoyable person in general. Next, I have fond memories of hanging around with Aimée during coffee breaks (with and without Bastian), other LIRIS events such as the poker nights, and the fantastic Thanksgiving dinners you organized. I hope you, your family, and your country will enjoy a prosperous future.

We arrive in 2014 and suddenly there was this big group of clever and eager new PhD students joining LIRIS. I hope I may have had the same good influence on you as the pre-boom students had on me. First of all I would like to thank my office mate Jasmien, for being the most kind and caring person I have ever met, and for delighting me by having such a memorable (dis)regard of the students you encounter. Next, I had the pleasure to be hanging out with Tom, sometimes referred to as Tom II, or the person who will end up with a 'DQ4L' tattoo on his left upper arm at some point. You are one of the most sincere researchers/persons I have met, and your thoroughness reflects the level of effort you put into friendship as well. Michael, though internally regarded as the second-smartest member of the Reusens family (which is not so bad given the general level I reckon), I dare to contest that opinion from my experience, and to me you are definitely the most fun to hang out with. María, you are a wonderful person to spend time with for you are a very light-hearted person and a great mother. Though Belgium is not as beautiful as Iceland, I hope that our hospitality makes up for that. Tine, you are the nicest walking dictionary to have roamed the LIRIS hallway, and I appreciated this ability very much. Besides, you also became a great Coca Cola fortune teller, which shows just how vast your range of capabilities is. Klaas, you are a very energetic person, who easily drags anyone along in your enthusiasm for many subjects. Also, if I ever feel like roasting a piglet in my garden, I will reach out to you. Eugen, the most politically incorrect German I ever came across, you often cracked me up with your sound jokes and general good spirit.

2015 brought us even more intellectually capable individuals, who seem to boast a level of experience not always corresponding to their age. Pieter, you make research seem to be incredibly straightforward. What you have accomplished in one year with such ease, probably is unattainable to many. Your passion (for trace clustering and process mining in general) fired up many intricate discussions, sometimes notwithstanding the general belief that process mining is not a good topic for coffee breaks. Vytautas, you exhibit the same features, albeit situated in a somewhat different area of research. Sandra, I would like to thank you for your enduring my incapacity to remember the name of your home country, though I am sure the depth of your knowledge on data mining coincides with your being able to relativize my foolery. Last but not least, I finally got my own declarative modeling successor in the form of Laurent, who excels at disregarding many research topics as ‘trivial’, for to you they are. I am certain you will find your way in between Computer Science and Information Systems and crack the code to the true meaning of the word ‘decision’, while keeping your cool in the process of doing so.

Finally, 2016 brought yet another batch of keen PhD students. While I have yet to get to know you better, I am sure you will all contribute significantly to LIRIS in the coming years. Rongjia, Galina, Daria, Anthony, and Faruk, I wish you all the best. Jenny, I did not know in which era to position you in exactly, but I remember your stays in Leuven with fondness as well.

Other intradepartmental thanks go out to professor Vandebroek, Martina, for trusting me to update the Ilo Vestac applets and chairing my PhD committee, and Peter, for being a great musical companion during the FEB graduation ceremonies.

Last but not least, I would like to thank my family and friends, for coping with my having a wildcard to pretend that I am occupying myself with something intellectually challenging, though in the end researching is (almost) a job like any other. You set up many memorable events throughout the years which took my mind off of declarative process models, which in itself was quite the achievement.

Ynte’s parents, Lut and Bart, thank you for trusting me, who quit a stable job to do some vaguely defined research project for three years, marrying your daughter and for being supportive throughout the years, and Jorne, for being an overall good sport.

I am especially grateful to my parents, who have supported me all my life with the most endearing care and dedication. Without you I would have never been able to get where am I today. You provided me with the best nurturing ground for growing as a human being, and even while I was ‘grown-up’ and doing a PhD, you showed genuine interest and motivated me to achieve the best possible outcome with great enthusiasm.

Finally, I could not forget to thank my wife, Ynte, for her support these last six years, for making me believe in myself, and for starting such a nice family together. Without me being able to come home to you, I would have not had the mental capacity to deliver this PhD. We laugh together, we annoy each other, we play with the cats together, we understand each other like no one else does, and this all makes me incredibly happy. We are now on the verge of new adventures, and I am very much looking forward to starting the next chapter in our lives together. I love you.

Johannes De Smedt
November, 2016

To my wife, Ynte,
to my furballs, Quincy and Kasper,
and to my parents,
for their tremendous support.

Management Summary

The challenging task of managing business processes has become an even more complex endeavor as companies are required to sustain flexible and agile business practices. Process management research has proposed numerous ways of accommodating for this need for flexibility. The declarative process modeling paradigm, with its shifts towards a constraint-based way of approaching process behavior, is a prime example. The downside of numerous languages covered by this paradigm, however, is the complex nature of their constraints, as well as their interactions. This thesis offers numerous solutions to overcome these impediments for improved usability of declarative process models.

In the first part, an overview is given of the current landscape of declarative process modeling, extended with the formal basis for the remainder of the text. A conversion of a common body of process constraints into a procedural variant is provided as well.

In the second part, an approach to reveal (hidden) dependencies among constraints in declarative process models is proposed. This contribution allows users to better grasp the full behavior of such models, for implicit connections are explicited and added as an extra layer of annotation on top of the current representation. The effectiveness and usefulness of the approach is illustrated in a user study and is also used for constructing a complexity measure for constraint-based models.

In the third part, the comparison with the procedural process modeling paradigm is made. Analogies and intricacies to both approaches are leveraged towards modeling in a mixed-paradigm fashion, as well as towards achieving better automated process discovery results. The findings are further extended by an approach for checking mixed-paradigm models for inconsistencies, and a conformance checking approach for assessing mixed-paradigm mining results.

Finally, the last part provides an outlook for future work.

All examples are elaborated in the Declare framework, which provides a widely-supported language and body of tools.

Contents

Part I

The Current Landscape of Declarative Process Modeling

CHAPTER 1

Introduction

"It isn't necessary to imagine the world ending in fire or ice – there are two other possibilities: one is paperwork, and the other is nostalgia."

— Frank Zappa

Companies and governmental institutions nowadays face challenging times that require them to perform in a fast-paced and quickly developing environment which puts a strong emphasis on technology. The market is more volatile than twenty years ago and the agility to cope with this change has become a paramount cornerstone for the modern organization. Meanwhile, many aspects of information systems (ISs) have advanced to accommodate this shift by offering new architectural approaches, better data handling and analysis techniques, and so on. A key part of the modern IS is the business process that drives all the actions, both on a higher level as well as on a more operational level to capture the dynamic aspects that are tied to executing business practices in a structured, controlled, and documented manner. Throughout the years, research has adopted its efforts towards business process management accordingly, which has led to a shift to approaches that put flexibility in the center of attention. This shift has been made in numerous ways, either by extending traditional approaches or by introducing new modeling languages. This thesis performs an in-depth analysis of one of the most prevalent solutions towards introducing flexibility, namely declarative process modeling. It has been recognized as a true paradigm shift with the earlier procedural modeling approach.

In this work, a comparison of both paradigms is made in various ways and the synergies that arise when using a mixed approach are uncovered.

Furthermore, a more in-depth analysis of some of the features of declarative process models are studied. First, however, this chapter provides a review of the most important concepts used in this work.

1.1 Business Process Management

This section introduces the basic definitions and concepts that are prevalent in Business Process Management.

1.1.1 Definitions

The area of Business Process Management (BPM) has seen a vast surge in research attention over the last two decades. For a comprehensive overview of its purposes and definitions, as well as its history, the survey in [155] is highly recommended. A widely used definition of a *business process* is adopted from [179]:

> “A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.”

which is very similar to the one of [44]:

> “An entire chain of events, activities, and decisions that ultimately add value to the organization and its customers.”

Clearly, the key components of a process are the *activities* and their instantiations called *events*, the *organizations* in which the activities are performed, and the *decisions* which are supported.

The definition of a business process clears the road towards defining *business process management* [179]:

> “Business process management includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes.”

BPM encompasses everything and everyone that surround the business process from its conception, all the way through the implementation, execution, refinement, monitoring, and so on.

1.1.2 BPM lifecycle

To enable a structured approach of dealing with a process, the BPM lifecycle is often used to frame the efforts towards achieving a successful process management system [44]. The lifecycle can be depicted as in Figure 1.1. It consists of six stages with each corresponding to a different stage of maturity.

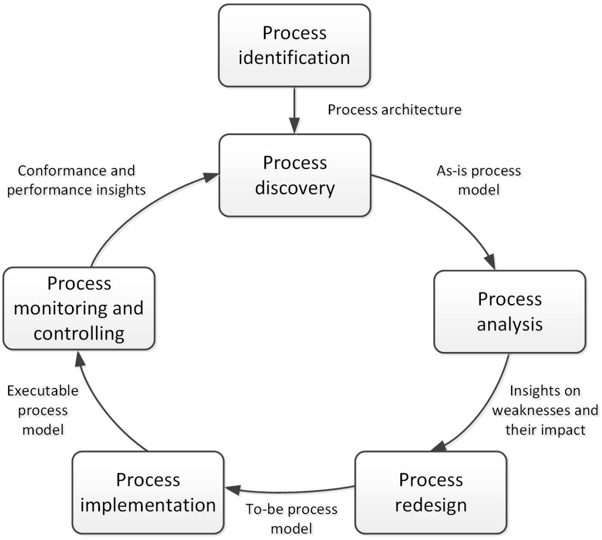


Figure 1.1: The BPM lifecycle as interpreted in [44].

First of all, *process identification* consists of the primordial task of finding the scope of the business process, i.e., where the process fits in the overall information system and how it is connected with other processes, hence focusing on the architectural viewpoint. When the process is situated, the actual specification can take place in the *process discovery* phase. The activities that need to be performed are delineated and a *control flow*, the ordering of activities, is constructed. The stakeholders and process owners are assigned, and relevant data that is part of the process and that is manipulated by the activities are all condensed into one process model, which is often depicted as a flowchart-like artifact. Many languages exist to express process models, which range from the typical procedural ones such as Business Process Model and Notation (BPMN) [114] to declarative languages such as Declare [119]. Ideally, they are easily transferable to a BPM system, or Process-Aware Information system (PAIS) which serves as an execution environment later in the *process implementation* phase. The model is either constructed through observing the current process, the automated discovery of the current process management system, or from scratch. It serves as a tool to discuss and prepare the next steps, which are *process analysis* and *process redesign*. The former takes the AS-IS process and finds ways to optimize it. This can include measuring the efficiency in terms of turnover time, the amount of flexibility, the complexity, and so on. All these dimensions can be improved upon in the redesign phase by reducing waiting times, increasing flexibility, downsizing the process, or redefining roles to perform certain activities. Once the process is deemed fit for use, it can be implemented and executed. This real-life instantiation introduces the possibility to start tracking and monitoring the process, which results in the last stage before closing the loop, namely *process monitoring and controlling*. Its purpose is to track all activities, actors, and process key performance indicators and indicates whether disruptions are present in the system. This delivers actionable data for stakeholders which can be used to traverse the whole loop and adjust the as-is process accordingly, update the analysis, and eventually invoke re-engineering and implementation.

While every phase of the lifecycle is indispensable to obtain process management of a high quality, this thesis will focus on two phases in particular, i.e., process discovery and implementation. First of all, process discovery is a cornerstone of the thesis, as a strong emphasis is put on the modeling and understanding of declarative and mixed-paradigm processes, as well as the discovery of models from historic event data. The latter can also be considered as a part of process analysis and redesign, however, in the context of this work it resides close to the languages and constructs of models used for representing a process. Second of all, process implementation in the form of executable process models is present, especially in the context of executable declarative and mixed-paradigm models. Hence, the majority of the lifecycle phases are covered or at least touched upon in some way, except for the identification phase. Since the discovery phase is situated on a higher, architectural level, there is often no real need for models that adhere to a certain paradigm. There is less focus on the expliciting of processes, but rather on the holistic view. In some respect, this thesis will also reflect upon the ideas of combining processes that are modeled according to different paradigms, however, this is considered to be rather a challenge for the process discovery phase.

1.2 Procedural and Declarative Process Discovery

In this section, a brief overview of procedural and declarative process discovery is introduced. First, the act of manually constructing a model is reviewed. Next, the task of the automated discovery of a process model with mining techniques is introduced. Both form an important building block for positioning the subjects dealt with in this thesis.

1.2.1 Process Modeling

To guide the reader in understanding the basic concepts of business process modeling, an example of a simple process model is given in the BPMN notation. Consider Figure 1.2. A simple paper writing process is depicted. It contains some of the major elements of a business process, i.e., activities (depicted as rounded rectangles), actors and organizations (depicted as rectangular pools and lanes), and routing elements (depicted as arcs and diamond-shaped connectors).

In this process, a researcher writes a paper and sends the end product to a conference. A *message flow* is used to represent the fact that a paper is sent to the program committee of the conference. After reviewing the paper, the committee sends a notification regarding the successfulness of the submission. This is processed by the researcher, after which she/he acts accordingly. Either the paper is rejected and she/he asks the promoting professor for another project, or the paper is accepted, which leaves the researcher preparing the final version and (hopefully) spending a lot of money on flight tickets, conference registration fees, and hotels. After these actions, the process ends. Notice that there exist many more aspects and accompanying symbols that are not used here. For example, exceptions and timers can be introduced to capture that the researcher employs a backup plan in case the notification is not received timely.

Notice that there is a strong emphasis on the control flow aspect of the process, i.e., the arcs and connectors take up a central role in modeling and understanding the process. The declarative way of modeling a process is different in this regard. In analogy with declarative (logic) programming [80], the declarative process modeling paradigm aims at specifying the relationships between activities without envisioning an underlying control flow. According to [119], the focus is shifted from *how* a process should be performed, to *what* should be performed. In Chapter 2, a more in-depth overview of what a typical declarative process model constitutes is given. For now, a slimmed-down variation of the paper writing process is given to establish an intuition. Consider the declarative process model in Figure 1.3. It is modeled in Declare [118], one of the most widely used declarative process modeling languages, at least in terms of citations. Rather than fixing the order in which the activities have to be performed, it applies a constraint-based approach. There are five activities, which are linked through constraints that each have a different impact on when and how they can be executed, but not every constraint is aimed at defining a sequence relationship. The intersec-

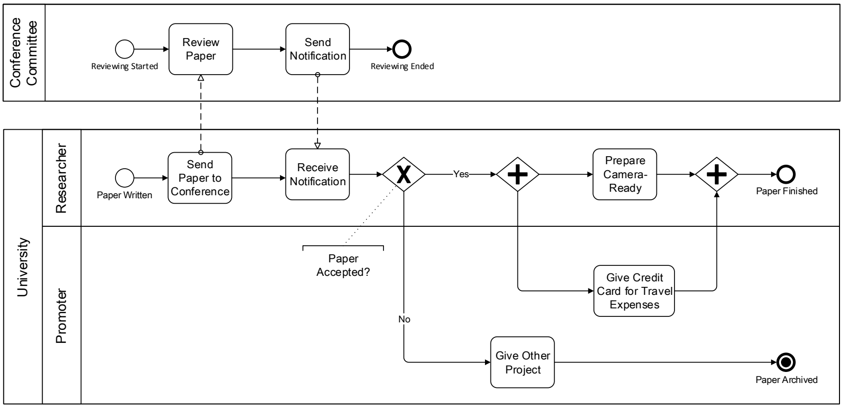


Figure 1.2: A simple business process representing the writing of a paper in standard BPMN notation.

tion of all the implications that the rules impose on the activities then forms the behavior that is allowed for by the model. First, sending a paper to a

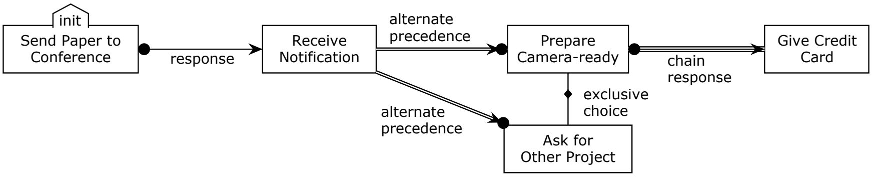


Figure 1.3: A simple business process representing the writing of a paper in standard Declare notation.

conference is required. Afterwards, the notification can be received, but also a new paper can be sent to the conference. The *response* constraint implies that a paper can be sent all the time, and notifications can be received, but in the end, there needs to be at least one notification before the process ends and after the last occurrence of the sending of a paper. This means that there is very little structure in the way these two activities can occur, hence it is very hard to capture this in a procedural workflow. The same holds for the activity *Give Credit Card.* The credit card can only be given right after the preparation of a camera-ready paper. While this sounds rather procedural, consider that in the case of concurrency and active parallel branches in a procedural model, it is hard to clearly indicate this without some extra symbolism or even duplication of the pair of activities throughout the control flow. The rest of the description and semantics of the constraints can be found in Chapter 3. The exact behavior is not further explained, however, as this model also serves as an interesting example to be tested with the execution environment introduced in Chapter 4. The constraint-based approach also inherently offers the possibility to monitor the constraints during execution and check for violations. In general, more flexibility in terms of execution order is available, which shifts the decision of what the next step to execute is towards the user. This is particularly interesting in environments where it is necessary to leave many options open for execution, or where the user actually better knows how the process needs to be executed in, for example, very knowledge-intensive processes such as medical examinations.

Nevertheless, every approach has its merits. It would perhaps not be very interesting to model a simple, straightforward model such as the writing of a paper with Declare. The difference between when and how each paradigm can excel is one of the main topics in this thesis.

1.2.2 Automated Process Discovery

When processes are implemented in an information system, they leave a trail of executions, called *traces*. They contain the enactment of activities in the form of *events*. In Table 1.1, an example of a possible implementation of the paper writing process is shown. The log contains three different straightforward executions of activities bundled by the case ids and pertaining to the writing of one or more papers by multiple researchers. The log also contains other information, depending on which activity is executed, such as the number of pages, the outcome of the notification, and the amount of money received. The field of process mining [152] deals with the automated retrieval of process models from such event logs. There exist many types of algorithms for this purpose, which can also be tailored towards discovering

case id	**time**	**event**	**resource**	**attributes**
1	05/07/’15 15:20	Send Paper to Conference	Johannes	5pages
1	05/07/’15 15:38	Send Paper to Conference	Johannes	5pages
1	15/10/’15 15:21	Receive Notification	Johannes	accept
1	13/12/’15 16:45	Prepare Camera-Ready	Johannes	14pages
2	18/09/’15 14:01	Send Paper to Conference	Estefanía	13pages
2	31/12/’15 14:58	Receive Notification	Estefanía	accept
2	21/02/’16 15:02	Ask for Other Project	Estefanía	2years
3	23/03/’16 09:43	Send Paper to Conference	Jasmien	29pages
3	05/04/’16 23:19	Give Credit Card	Jan	1.300euro

Table 1.1: An overview of a simple event log, which might have resulted from the paper writing process.

procedural or declarative models. Procedural approaches typically retrieve a holistic process model that describes the full behavior of the log, although fragmented approaches exist as well. On the other hand, declarative approaches often test all the entries in the set of constraints over all possible combinations of activities to verify whether they hold over the behavior observed in the event log. In the example log, it is not possible to retrieve the model from Figure 1.2, as multiple papers are sent in case 1. The declarative model, however, will aptly rediscover the behavior, but might also include a lot of constraints that are not informative as no counter evidence is found. Each paradigm has its drawbacks in this respect, which will be investigated in the thesis.

Next to discovering models from data, process mining also investigates the act of reconstructing the execution log over a process model. This is called conformance checking [133, 169]. Many techniques exist for this purpose and are available for both paradigms. They are useful in order to check whether there were violations in the implemented system that were, e.g., not anticipated by the reference as-is model. Correspondingly, companies can act upon these insights in order to re-engineer the process according to unsupported behavior, or underperforming parts in a model. Consider for example case 3 over the model in Figure 1.3. Clearly, after executing the trace, the *response* constraint is not fulfilled, as no notification was received. This can be due to bad monitoring, exceptions, or manual overriding of the process.

1.3 Methodology and Research Questions

This thesis situates itself on the verge between Information Systems and Computer Science research. On the one hand, formal definitions and techniques for modeling checking, machine learning, etc. are used towards business purposes such as process modeling. Research in general requires the researcher to perform some elementary activities, including generating new ideas and new practices which should also support enterprises in their generating new products, or new business [41].

This thesis makes use of the design science research paradigm applied to an information systems context research [70]. Design science differs from the typical natural sciences in the fact that there is an emphasis on *creating* things, typically called *artefacts*, rather than on describing (reality) [99]. There exist 4 such artifacts, i.e., constructs, models, methods, and instantiations. Constructs provide vocabulary and symbols, e.g., the set of symbols used in BPMN, models provide abstractions and representations, e.g., a business process model uses these constructs to give a holistic overview of tasks performed in an organization, methods provide algorithms and abstractions, e.g., the BPM lifecycle offers a structured way to deal with processes in enterprises, and instantiations provide implementations and prototypes, e.g., a BPMN model is implemented in an information system.

This thesis adheres to this framework by offering new methods to combine business process modeling constructs, an approach to analyze declarative process models, and new algorithms for automated discovery and verification. Design science was later presented as a three cycle approach [69] that connects the research to its environment and the current knowledge base, as depicted in Figure 1.4. The purpose of this is to ensure that research is indeed performed in answer to actual problems that exist in the application domains which is reflected by the relevance cycle, though grounded in the current body of existing work. The research performed in this thesis addresses the need that exists in organizations for solid approaches in Process-Aware Information Systems (PAIS) that support flexibility and agility. It was observed from the body of literature that the declarative process modeling paradigm has been one of the primordial approaches to address this need, however, there are some impediments regarding understandability and scope. Hence, it was first studied how they relate to currently existing and successful procedural techniques, and in a second and third iteration the issues with understandability were studied and later reevaluated with users. The different iterations are reflected in the different parts that exist in the thesis and have all been published in scientific outlets to add to the current body of knowledge.

Throughout the different design science cycles, the following research questions were answered:

I How can declarative process models be improved in order to raise their usefulness and applicability?

  i How can declarative process models be made more understandable?

  ii How can we provide better analysis and learning techniques for declarative process models?

II How do declarative and procedural process models relate to each other, and how can they be leveraged towards better process modeling and mining techniques?

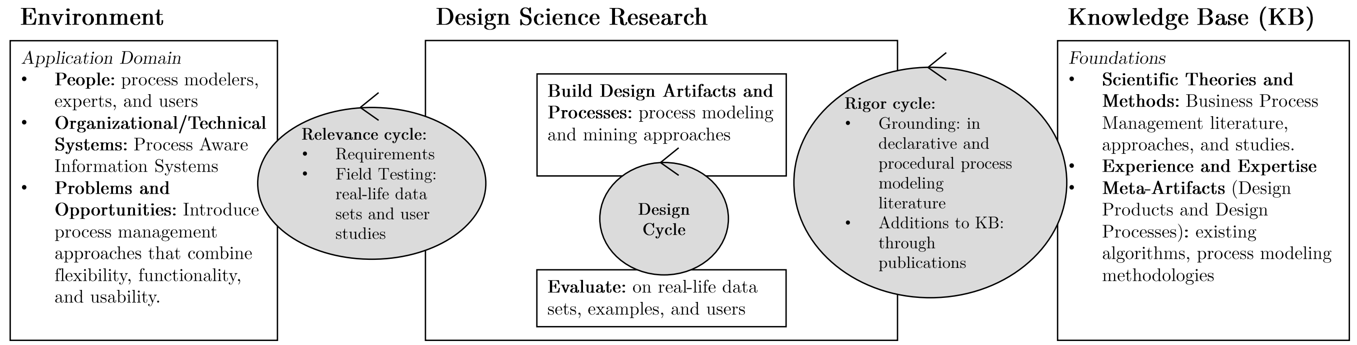

Environment
Application Domain
• People: process modelers, experts, and users
• Organizational/Technical Systems: Process Aware Information Systems
• Problems and Opportunities: Introduce process management approaches that combine flexibility, functionality, and usability.
Relevance cycle:
• Requirements
• Field Testing: real-life data sets and user studies
Design Science Research
Build Design Artifacts and Processes: process modeling and mining approaches
Design Cycle
Evaluate: on real-life data sets, examples, and users
Rigor cycle:
• Grounding: in declarative and procedural process modeling literature
• Additions to KB: through publications
Knowledge Base (KB)
Foundations
• Scientific Theories and Methods: Business Process Management literature, approaches, and studies.
• Experience and Expertise
• Meta-Artifacts (Design Products and Design Processes): existing algorithms, process modeling methodologies

Figure 1.4: The three-phased design science cycle as proposed by Hevner [69], applied to the research performed in this thesis.

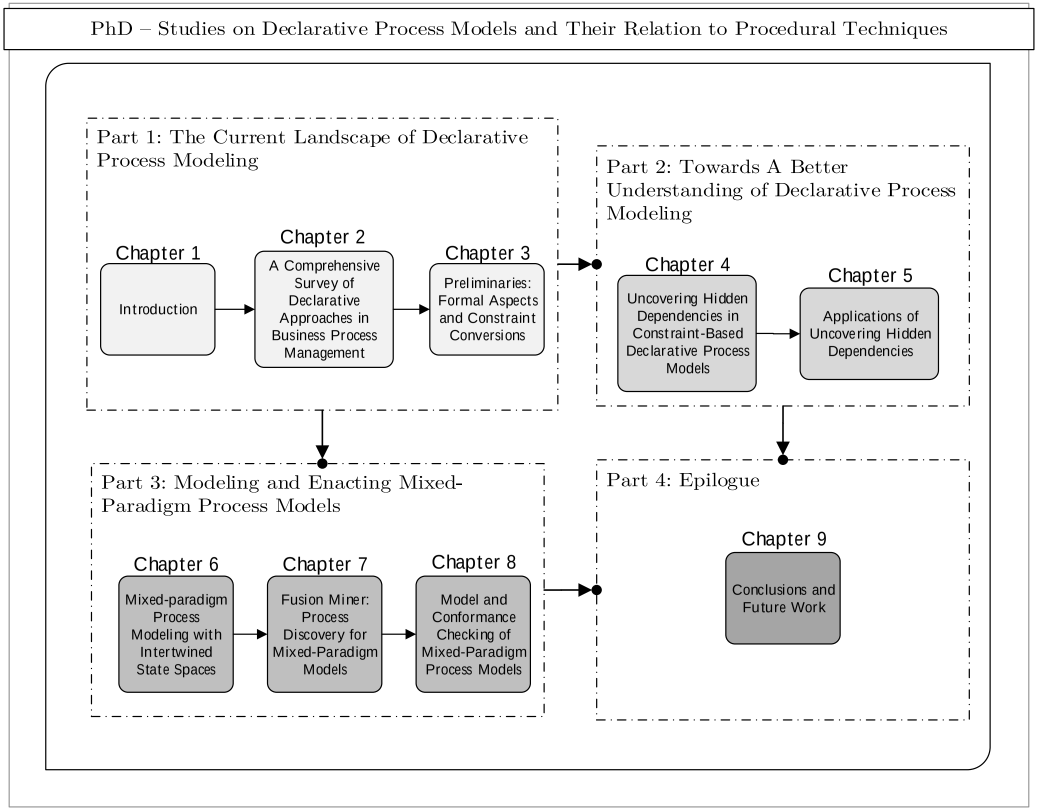

PhD – Studies on Declarative Process Models and Their Relation to Procedural Techniques
Part 1: The Current Landscape of Declarative Process Modeling
Chapter 1
Introduction
Chapter 2
A Comprehensive Survey of Declarative Approaches in Business Process Management
Chapter 3
Preliminaries: Formal Aspects and Constraint Conversions
Part 2: Towards A Better Understanding of Declarative Process Modeling
Chapter 4
Uncovering Hidden Dependencies in Constraint-Based Declarative Process Models
Chapter 5
Applications of Uncovering Hidden Dependencies
Part 3: Modeling and Enacting Mixed-Paradigm Process Models
Chapter 6
Mixed-paradigm Process Modeling with Intertwined State Spaces
Chapter 7
Fusion Miner: Process Discovery for Mixed-Paradigm Models
Chapter 8
Model and Conformance Checking of Mixed-Paradigm Process Models
Part 4: Epilogue
Chapter 9
Conclusions and Future Work

Figure 1.5: An overview of the structure of this dissertation.

1.4 Outline and Contributions

An overview of the structure of this dissertation can be found in Figure 1.5. The rest of this thesis is structured as follows. In the rest of Part I, “The Current Landscape of Declarative Process Modeling”, two chapters will introduce extra insights into declarative process modeling. First of all, the existing declarative process modeling paradigm literature is positioned along the different phases of the BPM lifecycle in Chapter 2. This exercise gives an idea of how this approach has matured over the years, and which challenges exist. Finally, it also frames the problems that are solved in this thesis.

To avoid notational issues and to ensure consistency, the main concepts regarding process models and their semantics are elaborated in Chapter 3. Also, it offers a full conversion of Declare into R/I-net constructs [174]. This has been published as a technical report in:

– Johannes De Smedt, Seppe K. L. M. vanden Broucke, Jochen De Weerdt, and Jan Vanthienen. **A full R/I-net construct lexicon for declare constraints.** Technical report, KU Leuven, 2015.

Part II, “Towards A Better Understanding of Declarative Process Modeling”, deals with the work on understanding the interplay of constraint in constraint-based declarative process models. In Chapter 4, the approach to retrieve the connections that exist between constraints is elaborated on.

In Chapter 5, the applications that are made possible by uncovering dependencies between constraints are presented. They include a user-study that reports on the increased understandability of the models when annotated with extra information regarding the constraints, the construction of a suitable complexity measure for constraint-based process models, the gamification of declarative models, the refactoring of such models, and the implications for semantics. The work has been published in:

– Johannes De Smedt, Jochen De Weerdt, Estefanía Serral, and Jan Vanthienen. **Gamification of declarative process models for learning and model verification**. In Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers, pages 432–443, 2015.

– Johannes De Smedt, Jochen De Weerdt, Estefanía Serral, and Jan Vanthienen. **Improving understandability of declarative process models by revealing hidden dependencies**. In Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings, pages 83–98, 2016.

- Johannes De Smedt, Jochen De Weerdt, Estefanía Serral, and Jan Vanthienen. **Discovering hidden dependencies in constraint-based declarative process models for improving understandability**. Inf. Syst. In submission.

In Part III, "Modeling and Enacting Mixed-Paradigm Process Models", the connection with the procedural process paradigm is established in three chapters. First, in Chapter 6 the comparison and construction of mixed-paradigm modeling is investigated. This yields insights in the similarity of modeling constructs, as well as insights into which characteristics are the most desirable in each paradigm. Next, an automated discovery approach is presented in Chapter 7. It leverages the best of both paradigms towards a better process mining technique. Finally, Chapter 8 illustrates how mixed-paradigm models can be checked and used towards a smart discovery approach. The findings have been published in the following works:

- Johannes De Smedt, Jochen De Weerdt, Jan Vanthienen, and Geert Poels. **Mixed-paradigm process modeling with intertwined state spaces**. Business & Information Systems Engineering, 58(1):19–29, 2016.

- Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. **Multi-paradigm process mining: Retrieving better models by combining rules and sequences**. In OTM Conferences, volume 8841 of Lecture Notes in Computer Science, pages 446–453. Springer, 2014.

- Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. **Fusion miner: Process discovery for mixed-paradigm models**. Decision Support Systems, 77:123–136, 2015.

- Johannes De Smedt, Claudio Di Ciccio, Jan Vanthienen, and Jan Mendling. **Model checking of mixed-paradigm process models in a discovery context**. In Business Process Management Workshops, Lecture Notes in Business Information Processing. Springer, 2016.

In the last part, the Epilogue, Chapter 9 summarizes the main conclusions that can be drawn from this work and provides a quick glance at possible future work.

Besides these works, the PhD project has yielded the following works as well:

On Context-Aware Petri nets:

- Estefanía Serral, Johannes De Smedt, and Jan Vanthienen. **Making business environments smarter: A context-adaptive Petri net**

approach. In UIC/ATC/ScalCom, pages 343–348. IEEE Computer Society, 2014.

– Estefanía Serral, Johannes De Smedt, and Jan Vanthienen. **Extending CPN tools with ontologies to support the management of context-adaptive business processes**. In Business Process Management Workshops, volume 202 of Lecture Notes in Business Information Processing, pages 198–209. Springer, 2014.

– Estefanía Serral, Johannes De Smedt, Monique Snoeck, and Jan Vanthienen. **Context-adaptive Petri nets: Supporting adaptation for the execution context**. Expert Syst. Appl., 42(23):9307–9317, 2015.

On decision modeling and mining:

– Johannes De Smedt, Seppe K. L. M. vanden Broucke, Josue Obregon, Aekyung Kim, Jae-Yoon Jung, and Jan Vanthienen. **Decision mining in a broader context: an overview of the current landscape and future directions**. In Business Process Management Workshops, Lecture Notes in Business Information Processing. Springer, 2016.

– Laurent Janssens, Johannes De Smedt, and Jan Vanthienen. **Modeling and enacting enterprise decisions**. In CAiSE Workshops, volume 249 of Lecture Notes in Business Information Processing, pages 169–180. Springer, 2016.

– Laurent Janssens, Ekaterina Bazhenova, Johannes De Smedt, Jan Vanthienen, and Marc Denecker. **Consistent integration of decision (DMN) and process (BPMN) models**. In CAiSE Forum, volume 1612 of CEUR Workshop Proceedings, pages 121–128. CEUR-WS.org, 2016.

– Jan Vanthienen, Caron, Filip, and Johannes De Smedt. **Business rules, decisions and processes: five reflections upon living apart together**. In Proceedings SIGBPS Workshop on Business Processes and Services (BPS'13), pages 76–81, 2013.

On learning analytics:

– Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Johannes De Smedt. **Student journey mapping: Learning analytics in action**. In Data Analytics Applications in Education, 2015.

– Johannes De Smedt, Seppe K. L. M. vanden Broucke, and Jan Vanthienen. Data Analytics Applications in Education, chapter **Improved Student Feedback with Process and Data Analytics**. Taylor and Francis. In submission.

CHAPTER 2

A Comprehensive Survey of Declarative Approaches in Business Process Management

"An original idea. That can't be too hard. The library must be full of them."
— Stephen Fry

In recent years, the introduction of the Declarative Process Modeling (DPM) paradigm has brought a new perspective to the Business Process Management community for topics such as process modeling, execution, monitoring, and so on. It marks a departure from modeling fixed process sequences but rather poses different restrictions on how to execute the activities within a workflow, typically in the form of constraints. Many aspects are hence revised, from supporting flexibility and maintenance over understandability, to expressiveness, and even integration with procedural models. This chapter summarizes the current situation of declarative process models (DPMs) in the light of the business process management lifecycle to address how mature the field has become in the last decade. The main findings include that, indeed, the declarative process management field has adopted a wide range of concepts from the lifecycle while providing alternative interpretations that are tailored towards its inherent properties such as the support of flexibility.

This chapter provides an extensive overview of the relevant works and problems that exist in DPM, and provides a backdrop for the related work of the rest of the thesis. It connects with the basic introduction of Chapter 1 in the sense that it elaborates the DPM paradigm along the BPM lifecycle to fill in the specifics of this paradigm and elaborates on the challenges that still exist in a more detailed manner. The published work of the other chapters, if relevant, is already included in the study, which was performed in March 2016. Consequently, it also constructs a link of the work in this thesis with the research area and explains which particular problems were addressed.

2.1 Introduction

Recently, major contributors to the BPM community have analyzed the state of the research area by reviewing the works published in the proceedings of the influential BPM conference [125]. One of the results that followed from the assessment, was the insight into how skewed the different contributions were in terms of the BPM lifecycle [44]. The conclusion devised a better coverage of the underlit phases of that lifecycle, which might greatly improve the maturity of the field.

Over the last few years, the declarative process modeling paradigm has seen a vast surge in terms of publications and research interest, with major contributions only 10 years old. In this chapter, the exercise stated above is redone with a focus on declarative process models, including all approaches in which a declarative process model is used for the purposes listed in the BPM lifecycle in a comprehensive literature review. Given the young age and the rather niche topic, the magnitude of the amount of papers is still manageable and can offer insights into whether the declarative process paradigm is maturing in the same way as its parent field is.

In order to establish a full literature overview of the thesis, a digression on mixed-paradigm models is included as well. This supports the understanding of Part III of the dissertation.

The contribution of this chapter is to provide a comprehensive overview and a set of actionable insights to further mature the field of declarative process modeling, as the research area might greatly benefit from a status report to reflect upon and to steer the direction of future contributions.

This chapter is structured as follows. First, a comprehensive state-of-the-art of the research area is provided in Section 2.2. Next, in Section 2.3 the works are framed within the BPM lifecycle phases to get an overview of the maturity of the field. Finally, a roadmap is discussed based on the findings of Sections 2.2 and 2.3 in Section 2.4, followed by a conclusion and future work in Section 2.5.

2.2 A Comprehensive State-of-the-art of the Declarative Process Paradigm Literature

One of the first works that explicitly mentions to pursue a declarative approach for workflows, was Freeflow [43], a framework that incorporated constraint-based process modeling and declarative dependency relationships between activities. The first works, however, that mention the term 'declarative process model' are [118] and [55]. From then on, they became the standards to which nearly all other papers referred to, next to [156]. Besides, in many works Declare often became synonymous with declarative process models altogether. These early works, however, based themselves on ideas and works on flexible workflows such as the approach of Sadiq et al. [135], ADEPT [126], and the case handling paradigm [159], where a traditional workflow is revised to incorporate more flexibility in the sense that less of the process is defined in the model, but rather remains underspecified to support different execution scenarios at runtime in order to achieve adaptability.

The aim of this section is to construct a comprehensive overview of works that relate directly to declarative process models and as such explicitly claim to offer a framework which includes a complete idea for modeling, or a language that is tailored specifically towards executing in an underspecified fashion and for which this approach is either proposed, altered, extended, applied, and so on. All of this should be defined in a context of business processes and BPM, hence, works on the declarative scheduling of abstract sequences of events or non-process-oriented goals (such as the scheduling of, e.g., processor or computation steps, a common problem in declarative modeling) were not included.

Besides [59], the authors are not aware of any other comprehensive study of the declarative process modeling field. This work also has become somewhat obsolete, as the latest works included date back to 2012. Furthermore, it focuses not strictly on capturing all works that use declarative process models, but rather gives an overview of the approaches that are capable of incorporating flexibility into a process' workflow.

2.2.1 Methodology

The literature review is started from searching in three popular academic reference database systems, i.e., Scopus[1], Google Scholar[2], and DBLP[3]. Each of these search engines has its advantages and disadvantages and churn out

[1] `http://www.scopus.com/`

[2] `http://scholar.google.com/`

[3] `http://dblp.uni-trier.de/`

different results depending on indexation and hence outlets [51]. By combining the results of these three sources and a further search within the documents itself, a very comprehensive overview of the field of declarative process modeling and languages is believed to have been established. The search terms used were 'Declarative process model', and 'Declarative process', in combination with the mentioning of 'Business', 'Declarative process mining', and 'Declarative process' in combination with 'Business Process Management'. A manual test was performed that left out the search terms to ensure that no works from, e.g., declarative modeling, an area within computer science dedicated to modeling in logic, and other non-related works, e.g., dealing with declarative programming, would be included in the result. A final subset of 188 papers was produced, which was further downsized to 169. The final list is not included in appendix, as most of the papers can be found in the reference section of this thesis. However, a digital version can be found at `http://perswww.kuleuven.be/~u0092789/BPM2016/`, which includes the encoding and the benefit of offering easy filter capabilities.

The choice was made not to include technical reports, master, and PhD theses, for it was assumed that the most influential parts are published in an academic outlet. In case there were multiple entries due to a conference proceeding that was later extended into a journal paper, the latter was included. Finally, if there were important insights in these works, they were included anyway to obtain a complete coverage. For the sake of brevity, in this chapter only the most relevant works are referred to.

2.2.2 Common Definitions

In order to be able to capture the scope of the research area, it is important to delineate the actual idea that is inherent to declarative process models. For this purpose, the definitions from the most important works are stated below. Many works have proposed a definition or notion for declarative process models. The most prevalent one is the underspecification of the control flow of activities, which can be altered either through time ordering- or data-dependencies.

In [118] declarative process models are defined as models that specify *what* should be done without specifying *how* it should be done, referring to the nature of the control flow which is not fixed before run-time, in contrast with procedural approaches that rather overspecify the control flow up front. This is said to be an 'outside-in' approach, as depicted in the well-known figure in [119].

In [71], declarative process languages are defined as languages in which the control flow is defined implicitly rather than explicitly.

In [77] and other artifact-centric approaches, the notion of a declarative process model reflects the unspecified execution of lifecycles of entities, which can stem from not only temporal execution dependencies, but also data-dependencies.

In [49, 50, 127] the comparison with procedural and declarative programming languages is made in order to frame the concept, suggesting that the difference is the shift in predefining the control flow of a process.

Finally, in [56], a framework is proposed for marking the difference between procedural and declarative approaches, which is repeated in Table 2.1. It does not only summarize the more control flow-oriented aspects, but the general concept of any declarative system, either activity or artifact-based.

	Procedural models	**Declarative models**
Business concerns	Implicit	Explicit
Rule enforcement	What, when, and how	What
Communication	What, how	What
Execution scenario	Design-time	Run-time
Execution mechanism	State-driven	Goal-driven
Model granularity	Process-centric	Activity-centric
Modality	What *must*	What *must, ought, and can*
Assumption bias	Overspecified	Underspecified
Alteration	Design time	Design and run-time
Coordinator/Worker	Human-machine	Agent
Coordinator/Activity	Coordination $\neq$ activity	Coordination $=$ activity
Activity lifecycle	Single event	Multiple lifecycle events

Table 2.1: Main difference of characteristics between procedural and declarative process models [56].

Clearly, there is no real disagreement on the nature of what a declarative process model is. Generally, there has not been any evolution in the way researchers have approached the topic either. This also backs the choice of not including any procedural works that incorporate declarative concepts only partially, unless the goal is to find a fit between both paradigms. The subject of mixed-nature models [184, 144] is gaining traction in literature and hence also forms one of the subjects in the declarative process modeling domain as explained below.

2.2.3 Bibliometrics

In this section, some key figures regarding the contributors, the number of publications over time, and the most prevalent case studies are given.

Top Contributors

The main authors in the field are listed in Table 2.2. Comparable with the parent domain BPM, many contributors are specialized in business process mining and verification, rather than process modeling. The larger half of all papers is provided with a very formal component to express modeling constructs and their semantics.

Author	**Number of works**
Maggi F.M.	34
van der Aalst W.M.P.	21
Di Ciccio C.	25
Weber B.	19
Zugal S.	14
Montali M.	14
Slaats T.	13
Mendling J.	12
Pinggera J.	11
Vanthienen J.	10
HildeBrandt T.T.	10

Table 2.2: Top contributors to the research area.

Timeline

In Figure 2.1, an overview is given of the number of works published in the area over the past decade. Clearly, the field has gained a lot of traction since the appearance of [156, 55, 118], but now the interest level seems to have leveled off. It even shows somewhat of a decline in the last two years. Since many of the works originate from conference proceedings and workshops, the scope of these outlets might have influenced this result, e.g., BPM 2013, an important outlet for BPM research, saw a great number of works being published on the topic of declarative process modeling.

Case Studies and Empirical Evaluation

Since many works offer a research artifact in the form of a new modeling construct, procedure, or analysis approach, they provide synthetic examples or real-life cases to support the claim of being useful and effective. In total, 54 works include a real-life example, 32 provide a synthetic example, and 15 works provide a user study.

For process mining, many commonly-used event logs are included for benchmarks and evaluation, including the logs provided by the BPI workshop series, e.g., [165]. Process modeling approaches often resort to synthetic examples, and many of them stick to the same example to offer a clear view on the evolution or extension that is provided.

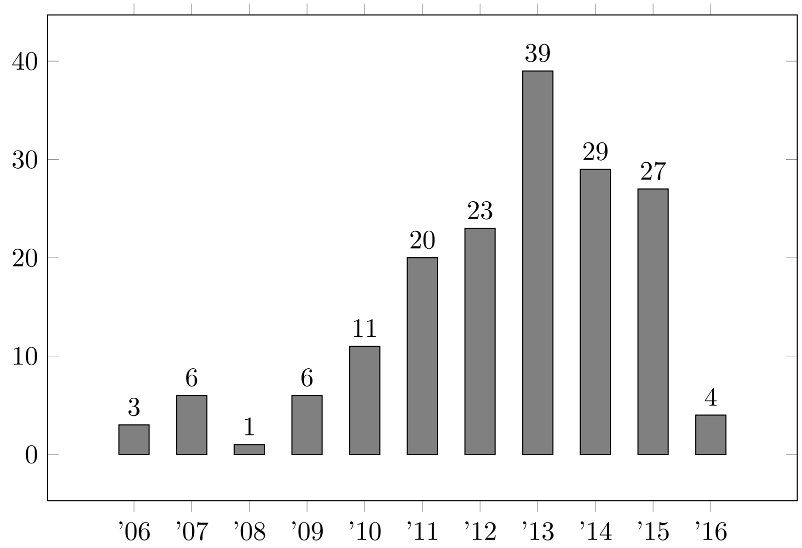


Figure 2.1: Overview of the number of papers published between 2006 and 2016.

Finally, there are many user studies to evaluate the user-friendliness and understandability of, e.g., Declare [187], as well as comparisons with other paradigms [120]. These empirical insights offer useful information on how to improve the approaches, however, the sizes of the studies remain relatively small and are performed in an academic context [143], rather than a professional context with business users, with the exception of [127].

2.2.4 Languages, Approaches, and Frameworks

In this section, an overview of the most widely accepted declarative process modeling frameworks, languages, approaches, etc. is given. It starts off with the most widely-used framework in terms of citations, and ends with the less commonly-cited ones.

Declare

Declare [119] is the most prevalent framework for modeling, executing, analyzing, and controlling declarative process models. It was initially proposed as ConDec and DecSerFlow [118, 156], which were comprised of a set of Linear Temporal Logic (LTL)-based constraints. The framework was tailored towards these purposes, however, it was eventually also used for process mining [93], scheduling and resource management [9], and conformance checking

[34, 35], and has been used in an extended or supplemented form as indicated in Table 2.3. These forms range from extensions to incorporate data, resource awareness, and extended support for model checking.

Declare aims at offering a generic framework for declarative process models, which does not limit itself to a type of semantics. Next to LTL, also finite state machine representations [22, 23, 185] were used. The template base used for the constraints is also not fixed, although it was never extended beyond the initial set. Very few papers support the exact same subset of the template base either, which conflicts with the target of being a general framework. Improvements and considerations on LTL for declarative process models were made in [180, 32]. Nevertheless, the rest of this thesis will use Declare for it has seen a wide-spread acceptance throughout the research community for various topics and numerous applications. This makes assessing, comparing and applying the findings in this thesis more straightforward. A more detailed discussion on the semantics and setup of Declare can be found in Chapter 3.

DCR graphs

DCR graphs [71, 72, 73] has a similar setup to Declare, however, the contributors claim that by offering a smaller subset of only four constraints rather than a big constraint set as provided in Declare, the execution is based on the direct translation of the semantics over transitions between markings of the graph, rather than on a conversion to LTL. Hence, it resembles marking-based models such as Petri nets [113] and offers a faster translation to executable automata. The framework has been extended to include other aspects, such as timing information [75], hierarchy [38], concurrency [40], and data in the form of context variables [74].

Guard-Stage-Milestone

The Guard-Stage-Milestone (GSM) approach [77, 78, 148, 30] originated from the artifact-centric Business Entities with Lifecycles approach. It supports the lifecycle of entities by providing them execution semantics based on Event-Condition-Action (ECA) rules [85]. While it does not aim to offer as rich an approach as Declare or DCR graphs in terms of expressing sequence constraints, it does provide a more powerful way to incorporate data into the process model as it stems from an enterprise modeling perspective. This way, it was actually used for the foundation of the Case Management Model and Notation (CMMN) standards' semantic core [100].

Author(s)	**Name**	**Purpose**
Barba et al. [10]	(T)ConDec-R	Extension of ConDec to include resources and time estimates for scheduling activities.
Maggi et al. [97]	RV-FLTL	Run-time Verification extension for LTL for finite execution lengths.
Sun et al. [145]	FOL	Combination of first-order logic and Declare LTL constraints to incorporate data from artifacts.
Montali et al. [111]	(Mobucon) Event Calculus	Used to express Declare constraints for time constraints, and non-atomic, durative activities.
Maggi et al. [97]	LTL-FO	Extension of Declare with first-order LTL rules to capture relations between tasks, conditions, making it possible to include data conditions in Declare.
Silva et al. [139]	PROV-DM	Provenance Data Model used next to Declare to represent dependencies between entities and their instantiations such as activities, agents, and time.
Laurent et al. [90]	Alloy	Implements Declare in first-order logic and enacts the model using Alloy and SAT solvers.
Lam et al. [88]	LWB	Redefines and extends Declare constraints in the Logic Workbench for enabling the use of natural language descriptions and formal verification.
Russo et al. [112]	CIGDec	Variant of ConDec for modeling and enacting clinical guidelines.
Montali et al. [109, 107, 108]	SCIFF/ CLIMB	CLIMB is a framework for modeling, enacting, verifying, and analyzing interaction models, with SCIFF as the reasoning and verification language based on abductive logic. They can incorporate Declare constraints for reasoning and verification purposes.
Kucukoguz et al. [86]	ArtiNets	Artifact-centric language which uses lifecycles similar to Declare constraints.
De Masellis et al. [36]	FOLTL(f)	First-order logic extension of LTL for finite traces to deal with complex data structures.
Mertens et al. [105]	Declare-R-DMN	Extension of Declare with Decision Model and Notation (DMN) [115] for expressing decision knowledge in workflows.
De Carvalho et al. [31, 140]	ReFlex	Provides a rule engine based on Declare constraints, which does not calculate the full state space, but rather calculates the possibility of reaching an end state in which all constraints are fulfilled to obfuscate illegal states from the user.

Table 2.3: All extensions or complementary approaches of Declare.

Other Approaches and Languages

As can be seen in Table 2.2, the works of the authors of the three previous approaches already cover more than half of all the work in the domain. Also Figure 2.2 indicates that the majority of works stem from Declare. Other languages that appear in literature, including the different variants of Declare as listed in Table 2.3, are not widespread, or do not seem to have the same impact.

Most notably, one of the first approaches that was proposed, is PENELOPE [55], a language that defines temporal deontic assignments over the places in a Petri net to model the control flow declaratively. Later, the authors also proposed EMBRACE, a framework for expressing business vocabulary, based on Semantics of Business Vocabulary and Rules (SBVR) [116, 57], with Colored Petri nets (CPNs) [83] providing the execution engine.

Also the Declarative Process Intermediate Language (DPIL) framework [138] has become an important approach for modeling and executing agile, resource-aware declarative process models.

Finally, declarative specifications can also be achieved by using behavioral profiles [177, 121, 122].

Mixed-paradigm Approaches

Besides efforts focusing solely on DPM, there exist many techniques on the verge between DPM and the traditional procedural process modeling paradigm. While the initial efforts on Declare already provide a basis for constructing mixed models with atomic subprocesses [119] containing Declare and Yet Another Workflow Language (YAWL) [157], recent efforts extend upon this concept to obtain a more general approach towards hybrid process modeling and mining in [141, 98, 39]. The benefit of such approaches is the possibility to use the most appropriate paradigm in a hierarchical way. Hence, flexibility can be achieved where necessary.

Next to atomic subprocesses, the works on mixed-paradigm modeling and mining with intertwined state spaces by using Petri nets and Declare [184, 142, 144], and BPMN and Declare [33] rather provide an approach for using languages of both paradigms over all activities at once to obtain a global semantics consisting of multiple languages. The difference is that activities move in both models at the same time, hence influencing and requiring an update of the semantics of either paradigm throughout the length of the execution.

2.2.5 Limitations

Evidently, it is possible that this study is still not comprehensive, and as stated above, tried to be as representative, rather than complete. The reader is also encouraged to provide any works that he/she feels are missing and could provide better or different insights into the domain of study.

2.3 Declarative Process Modeling Along the Business Process Management Lifecycle

In this section, the maturity of the declarative process modeling area is determined by positioning the works along the different phases of the BPM lifecycle. Often, many of the topics are intertwined. The strength of many DPM languages is that they combine modeling, verification, and execution often in one language or framework because of their constraint- or rule-based nature and formal background. Petri nets also exhibit this property and it forms one of the main reasons for their widespread use in the realm of concurrent systems [151]. In Figure 2.2, the number of works for every phase and every approach is given. Clearly, there is a big bias towards process discovery and implementation, while the other phases are relatively untouched. This is a trend that was also observed in [125]. Every phase is going to be reviewed for the current state, the opportunities, and future work that might enrich the lifecycle.

2.3.1 Process Identification

Current State

The phase of process identification encompasses finding relevant processes and their relations and churns out an appropriate architecture. This step forms the foundation on which a process management system builds and requires an in-depth study of the company's situation. For declarative process modeling, very few works actually try to support this phase in an elaborate way, with the works on CMMN being the closest to providing a methodology to perform process identification. Given the high-level and holistic nature of this phase, the works that deal with process identification also do not discuss paradigms as such, but rather delineate which parts of the process architecture requires flexibility and adaptability, hence paving the way for declarative modeling techniques to be used in the further lifecycle. In this respect, delineating paradigms might be one of the most prevalent process identification techniques available [119, 144].

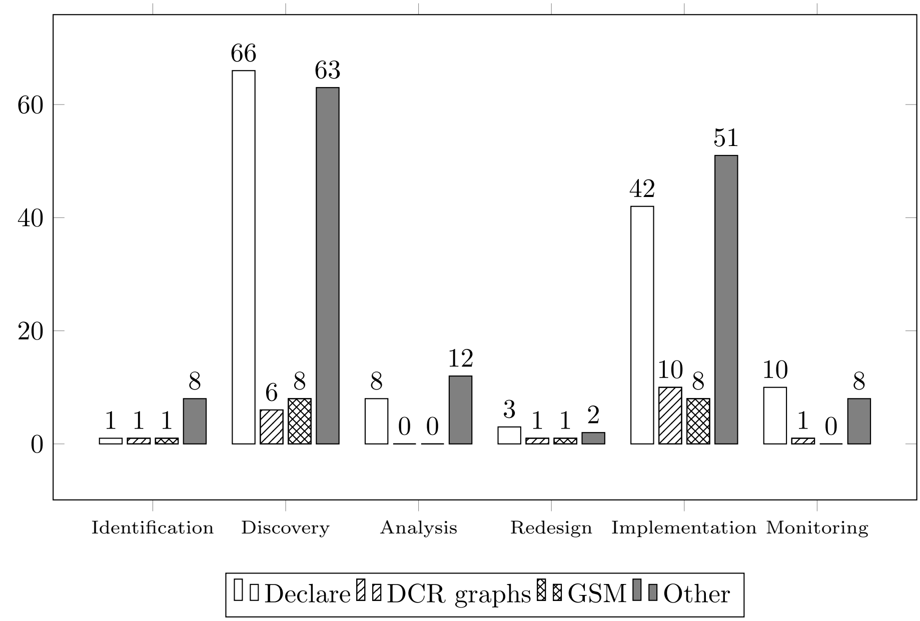


Figure 2.2: Overview of the number of papers published for the different languages and phases of the BPM lifecycle.

Opportunities and Future Work

Clearly, this phase of the lifecycle is not well-supported and can use extra attention in future research. The big question remains how declarative process models would influence the act of setting up the architecture and overall setup of an information system. For example, artifact-centric approaches have a wider impact than the process, while control-flow centric approaches typically have an execution-oriented impact.

For improving the incorporation of DPM in this phase, future process identification exercises should clearly state how paradigms can provide extra possibilities into constructing a process architecture and a global process outline.

2.3.2 Process Discovery/Modeling

Current State

In line with the findings of [125], most of the research focuses on the discovery of process models, including modeling and mining. This holds for all languages and approaches. Many works focus on either new extensions to existing approaches, as indicated in Section 2.2.4, or tackling more advanced problems in discovering process models from event logs, such as noise and errors [24, 21] or data-aware logs [97]. Process mining, however, is almost solely performed by Declare-related approaches [19, 93], while this component is absent for DCR graphs and was only recently introduced for GSM [123].

Opportunities and Future Work

Many frameworks are available for declarative process modeling, however, the question remains in which situation each of them excel. On the one hand, Declare and DCR graphs are focused on abstracting the control flow, occasionally enriched with data extensions, while GSM is artifact-centric with less attention dedicated to expressing control flow. Meanwhile, each of the languages can theoretically be used for mining models.

A full comparison framework for declarative discovery techniques should be established, focusing on understandability and expressiveness to assess which techniques are better suited towards different applications and different users.

2.3.3 Process Analysis

Current State

Almost no works provide in-depth insights into process analysis, being the systematic reviewing of a process according to well-defined performance measures. A notable exception is [100], providing metrics for CMMN models. Also, many works focusing on optimizing the execution of a workflow, such as [11] provide insights into which quantitative metrics can be used to analyze processes. However, efforts on qualitative analysis such as valued added and root-cause analysis are absent.

Opportunities and future work

Since the approach for analyzing processes is not dependent of the process model itself, there is no real need for adapting process analyses to DPM, especially for qualitative approaches. However, it remains to be seen whether DPM can support analyses such as calculating throughput time, due to the flexible nature of the process which entails many different outcomes. It remains to been seen whether these metrics still remain useful in such a context. Furthermore, it still needs to be seen whether there are other metrics that can be introduced, such as the number of scenarios still possible in respect to the remaining execution steps, to provide tailored metrics that make use of the properties of DPMs. Metrics tailored towards flexibility and adaptability could improve the analysis and the impact that declarative approaches have.

2.3.4 Process Redesign

Current State

Few works address the concept of declarative process redesign and solely focus on the construction or discovery of new models. Much like process analysis and identification, it may require to change and adapt the whole process on a higher level of abstraction. For DPM, model-driven or behavioral approaches that introduce hierarchy and restructure process models [190, 38, 47], are a common way to deal with redesign, as well as extending models by analyzing past behavior [167].

Opportunities and Future Work

The phase of redesigning addresses many aspects of the process, many of which are less focused on behavioral aspects, e.g., customer interactions or the resource perspective [44]. However, declarative process models could vastly improve processes in terms of flexibility, maintainability, and hence cost. In this respect, this phase is intertwined with the analysis phase. The more tailored metrics exist to evaluate DPMs, the more interesting it becomes to elaborate on redesigning for DPMs. Next to this, an interesting future study could entail researching whether shifting design from procedural to declarative process models would actually improve the process system via, e.g., a business case.

2.3.5 Process Implementation and Execution

Current State

Next to the process discovery phase, the implementation and execution phases receive most of the attention in literature. Many approaches incorporate both the modeling as well as the execution part in the works by default. Since a strong emphasis is put on the formal background of most approaches, these concepts typically are considered the focusing point of many works.

Besides providing execution semantics, two subjects that are prevalent for this phase of the BPM lifecycle, are model verification and conformance/compliance checking, which are often intertwined with the execution semantics due to the nature of constraint-based process models.

Opportunities and Future Work

The implementation and execution phase of BPM is well-understood and elaborated in DPM. Still, comparable to discovery, there exists no framework to compare, e.g., scalability, usefulness, and user-friendliness of the different approaches both between different languages (Declare vs. DCR graphs vs. GSM,...) as well as between different applications (e.g., SCIFF vs. a-posteriori checking [14] vs. conformance checking [35]).

2.3.6 Process Monitoring and Control

Current State

The benefit of declarative languages is that they are often rule-based, making them inherently tailored towards verification and monitoring. This phase of the lifecycle therefore is adequately represented by works such as [92, 94, 111], while DCR graphs and GSM are also tailored towards monitoring but have less works explicitly addressing the topic. Therefore, only the works that narrowed their scope to the monitoring applications itself were included in Figure 2.2. A case study of the application of CMMN in a hospital context

[68] shows the capabilities of case management and a declarative approach in a real-life setting for monitoring.

Opportunities and Future Work

The field of monitoring and complex event processing requires a substantial process system to be in place in order to validate the usefulness and power of the approaches used. For now, little real-life empirical evidence of monitoring with declarative process models in a large-scale environment has been offered. Performing such studies would greatly benefit the application set that different approaches offer. Again, a comprehensive framework for which approaches offer a more in-depth or more scalable solution is desirable in the future.

2.4 A Roadmap for Declarative Process Modeling

In this section the insights condensed from Sections 2.2 and 2.3 are used to form a body of future directions which are opportune for the DPM area.

2.4.1 Comparison Framework

As illustrated in the previous section, there exist a plethora of techniques for modeling, mining, and executing declarative process models, however, there is no framework to compare them, nor have their been any attempts at a full comparison between different aspects. Therefore the following dimensions are of interest for a comparison framework:

1. **User-acceptance:** Are DPMs tailored towards the needs of the users? Are they understandable or overly-convoluted and too academic to be used in a practical setting? Are they solving common problems better or rather providing new solutions?

2. **Scalability:** Which techniques are particularly fast in doing tasks such as verification and conformance checking? Are they suitable for monitoring and discovery in large-scale operations?

3. **Functionality:** What makes the unique characteristics of one DPM approach better over the others? Which situations benefit from either the extensive expressiveness for workflows of Declare or DCR graphs, compared to artifact centric situations where ECA rules are used in GSM? Is the elaborate execution component necessary in every situation or does CMMN provide a good alternative for sketching processes?

2.4.2 Declarative Process Modeling as a Mature Approach to Support the BPM Lifecycle

The state-of-the-art shows that there still remains a huge difference in terms of coverage of the BPM lifecycle. While this is due to the fact that the lifecycle is an all-encompassing framework not solely focusing on the modeling and behavioral aspects of a process model, DPM still misses out on work in substantial parts of the lifecycle that should be addressed in order to become mature along the full spectrum. Especially the way in which DPM can be incorporated as a genuine part of the identification phase is still missing in current literature.

Furthermore, few works have addressed how DPM can be used for process analysis and redesign. This is somewhat surprising, especially since these concepts typically rely on behavioral properties which are very well elaborated for declarative process models and hence can straightforwardly borrow insights of existing approaches that are used for procedural techniques. Next to this, tailored approaches making use of the intricate properties of declarative process modeling languages should be found for process analysis and redesign. It would also be interesting to indicate how it is easier or exclusively possible to express flexibility and agility metrics for analysis purposes compared to procedural models.

2.4.3 Empirical Evaluations and Advanced Case Studies

Few works show advanced empirical evaluation based on a holistic study throughout the whole BPM lifecycle. While there is plenty of evidence of discovery and execution approaches, few examples center around a case study and elaborate on the specifics that were solved and why declarative process models were, e.g., superior to each other or to other paradigms. Especially successful case studies where the full lifecycle is traversed with DPM in mind would vastly mature the usefulness of the approaches. While approaches such as GSM and DCR Graphs are particularly tailored towards the support of business needs, still, it would be beneficial to see how they can support a large-scale process environment. With the introduction of CMMN an industry standard is now available which is gaining traction among practitioners, as observed from a simple Internet lookup for the term. This could vastly increase the significance and importance of the formal research background of DPM, as a growing interest might bring along a greater need for analysis, model discovery and checking, and so on.

2.4.4 Towards Modeling Guidelines for Declarative Process Models

There exist numerous languages to model DPM and although there is a vast amount of papers that put the emphasis on constructing models, none of them has any specific guidelines for constructing them. This clashes with the fact that the understandability of the constraints is often low. For now, the works that offer the best reconciliation are [189, 65, 143]. These works provide descriptive studies that could be the starting point for condensing a full body of declarative process modeling guidelines that tailor existing guidelines [103] to the flexible and rule-based nature of the models, while taking into account previous efforts on procedural models [102, 175, 191]. This can bridge the gap that exists between comprehensibility and expressiveness, e.g., CMMN provides easy to understand models with simple semantics, while Declare has no limits in terms of expressiveness, while having very intricate semantics and possibly unclear visualizations [63].

2.4.5 Integration with Decision Modeling

The recent surge of work dedicated to decision modeling in BPM [12, 82, 13], especially in the context of the new Decision Model and Notation standard [146], has introduced a new way of shifting attention away from the control flow towards the decision layer of a process model. Often, decisions are hard-coded into a process model [173, 149], while it would be more appropriate to model them as a chain of decisions.

By combining the decision and declarative process modeling paradigm, a multi-perspective declarative process model can be constructed. Preliminary work of [105] provides a proof-of-concept for Declare and DMN, however, strong semantics that deal with the considerations of, e.g., [110], can provide a better and more holistic view that joins artifact- and control flow-based declarative approaches in a general concept.

2.5 Conclusion and Future Work

In this chapter, the state-of-the-art of the declarative process paradigm was established by performing a comprehensive literature study. The BPM lifecycle was used as a reference to frame the contributions in terms of the topics they address. It was shown that, comparable to the parent domain BPM, the same tendency to support modeling and execution exists, and the span of the field is still providing a rather narrow scope on the research. This calls for a more mature approach in research, as stated in the research agenda provided. Although many principles of BPM apply to DPM, there are still mismatches or gaps for the main efforts in BPM research were previously focusing on procedural languages for constructing, analyzing, and redesigning process models.

For future work, a roadmap was constructed which can be used to steer future research. Five actionable considerations were made that address different gaps in literature. The main focus should be on finding mature assessment frameworks for different approaches, a wider support of the BPM lifecycle, more empirical research, the introduction of tailored guidelines and analysis criteria, and the integration with decision modeling.

Finally, it was illustrated how the work in this thesis addresses current issues in DPM. Chapters 4 and 5 introduce approaches to overcome the convoluted and hard-to-understand semantics of the Declare language, while Chapters 6 through 8 make the comparison with the procedural process mining paradigm to infuse both methods with concepts of one another for both modeling and mining.

CHAPTER 3

Preliminaries: Formal Aspects and Constraint Conversions

"Is that everything?
It seemed like he said quite a bit more than that."
— Bill Murray, Lost in Translation

This chapter provides the thesis with the formal semantics for the process models throughout the work. More specifically, the semantics of the Declare language and Petri nets with its extensions for reset and inhibitor arcs are given to illustrate how the models can be executed, and provide the background to the formal aspects of the later chapters. The choice for these two languages is founded on the widespread use of both, each being a forefront for their respective paradigm. Petri nets provide a wide range of analysis techniques that are used to analyze workflows, and even other languages such as BPMN. Declare on the other hand, as illustrated in Chapter 2, is also adopted in many works as the prime declarative process language. Finally, previous work on combining these two languages for mixed-paradigm semantics [184] exists, confirming that they are an appropriate choice for this work. The introduction of this chapter also serves as a quick overview of the state-of-the-art regarding the conversion and mixed-paradigm approaches that exist for both languages.

Next to the separate semantics, a template-based conversion is proposed for every Declare constraint into a single Petri net fragment with weighted reset and inhibitor arcs, i.e., a weighted R/I-net. As such, a formalization of the execution semantics of Declare is obtained, similar to linear temporal logic or regular expressions, but now expressed in the form of Petri nets.

Equivalence of Declare constraints and the respective Petri net templates are analyzed at the theoretical level and by means of a simulation experiment.

3.1 Introduction and Related Work

Petri nets are a widely used language to express concurrent systems [113]. They are highly expressive and have clear execution semantics due to sound mathematical underpinnings, which results in a wide availability of robust analysis techniques. Therefore, in the context of business processes, Petri nets are extensively used for model design and verification [151], so that even more user-centric languages such as Business Process Modeling and Notation (BPMN) [20] and Event-driven Process Chains (EPCs) [150] are frequently converted to some form of Petri net for checking model properties such as soundness [44, 151].

As explained in the previous chapters, the need for flexibility in business processes, however, has led to an increased popularity of declarative models. An overview of the existing declarative approaches can be found in Chapter 2. Most notably, Declare has become the de facto standard for modeling flexible workflows by using an event-driven, constraint-based approach. Mixed forms of both paradigms are also gaining ground. YAWL [157], a language which extends Petri nets with more flexible constructs such as resetting capabilities, can include Declare constraints in subworkflows [160], a more advanced version of ad-hoc subprocesses in BPMN. A real mix of state-spaces of Petri nets and Declare is proposed and implemented in [184]. More recently, the incorporation of Declare constraints into BPMN was also investigated in [33].

In this chapter, the formal aspects of both Petri nets and Declare are presented, as well as a template-based approach that converts every Declare constraint into a weighted R/I-net, i.e., a Petri net with weighted reset and inhibitor arcs. This conversion results in a formalization of Declare, similar to Linear Temporal Logic (LTL) [29] or regular expressions [185], but one expressed with Petri nets. As such, a Petri net template lexicon is obtained that expresses the same behavior as the collection of Declare constraints originally described in the seminal work of Pesic [119].

The conversion of Declare models to Petri nets has been touched upon in [48], proposing R/I-net constructs for a subset of DecSerFlow constraints [156], the predecessor of Declare. Also, the synthesis to a Petri net without R/I-constructs is proposed in an example. The approach in this chapter differs from this by putting forward a full lexicon of conversions and a thorough analysis of their applications. A full synthesis approach is proposed in [124], where Declare constraints are redefined as regular expressions, converted to finite state machines and finally synthesized into Petri nets with the theory of regions [28, 27]. The conversion strategy of [124] differs from the one proposed in this chapter in several ways. First, it relies on a full executable Declare model, being a finite state machine. The upside is that there are no conflicting states in the model, which is superior over having the separate constraints that still might contain inconsistencies. The downside, however, is that it is computationally very expensive to synthesize such a model. Even for a few constraints and activities, this might already be computationally intractable. Second, synthesis will provoke a duplication of tasks (i.e. transitions with the same label) which results in (i) nets that are arguably even harder to read than automata, (ii) increased complexity for analysis tasks such as conformance checking, and (iii) impediments for the straightforward plugging in of Petri net fragments that could be obtained by synthesizing every constraint separately with the technique presented in [124]. Third, the use of inhibitor arcs to withhold an ending activity from firing when a constraint is violated as done in this work makes the specification and keeping track of violation easier. In addition, every violation can be traced back to a certain place and activity.

The remainder of this chapter is structured as follows. First, Section 3.2 contains a description of the semantics of both Declare (in LTL and regular expressions) and weighted R/I-nets. Section 3.3 describes in full the conversion lexicon of all Declare constraints into Petri net constraints. Section 3.4 contains an analysis of equivalence and empirical validation, followed by the conclusion in Section 3.5.

3.2 Preliminaries

This section provides a brief overview of formal concepts and definitions used throughout the thesis. First, a general notion of constraint-based declarative process models is given, after which the specific case of Declare is dealt with. Next, Petri nets and their extensions are introduced.

3.2.1 Constraint-Based Declarative Process Models

Many declarative approaches, such as Declare, DCR Graphs, and DPIL, each apply a constraint-based approach, i.e., the models consist of activities for which the behavior is applied by adding constraints over sets of them. A constraint-based declarative process model $DM = (A, \Pi)$ can be defined as follows.

- A is a set of activities or atomic propositions from the alphabet Σ,

- $\Pi(A)$ is a set of constraints defined over the activities,

- $\S(\pi)$, $\pi \in \Pi$ is a function assigning semantics to a constraint, and
- $\Phi = \bigwedge_{\pi \in \Pi} \S(\pi)$ is the model comprised of the conjunction of constraints, given that the language used to express the constraints is closed for common properties such as concatenation, intersection, Kleene star, and so on, as is the case for automata-based languages such as regular expressions and LTL.

3.2.2 Declare Constraints and Their Characteristics

Declare models are typically constructed by using the set of constraints of Table 3.1 of which the descriptions can be found in Table 3.2. They range from unary constraints, indicating the position and cardinality of an activity, to n-ary constraints, which capture typical sequence behavior such as precedence and succession relationships. A Declare model is the conjunction of its constraints, $\phi_{DM} = \bigwedge_{\pi \in \Pi} \S_{Declare}(\pi)$, where $\S_{Declare}$ corresponds to the semantics available in Table 3.1. The constraints can be represented in more detail as follows:

- $\Pi_1(a, P)$ is the set of unary constraints with P its properties indicating cardinalities or position, and
- $\Pi_2(a, b)$ where $a, b \in A$ is the set of binary constraints that follow an activation/resolution strategy.

In literature, a and b are, depending on the constraint, typically referred to as antecedent and consequent [14, 26, 95]. In general a serves as the antecedent, except for *precedence* constraints, where this relation is reversed. For some constraints, both a and b serve as antecedent and consequent at the same time, i.e., for *(not) responded/co-existence*, *(exclusive) choice*, and *succession.* There also exist constraints that use a set for a or b called branched constraints [117], most notably the target/consequent-branched constraints [26]. In this case, multiple consequents can resolve the status of a constraint. For, e.g., *response(a,B)*, the LTL-formula becomes $\Box(a \implies \Diamond(\bigvee_{b \in B} b))$ and any $b \in B$ can resolve the temporary violation caused by an occurrence of a. The automata of the separate constraints expressed in Finite State Machines/Automata (FSM/A) can be found in Section 3.4.

Finally, for an activity $a \in A$ it is defined:

- $\bullet a = \{\pi \,|\, \pi(b, a) \in \Pi_2 \vee \pi(a, P) \in \Pi_1\}$ all prefix constraints (and unaries) of an activity, and
- $a\bullet = \{\pi \,|\, \pi(a, b) \in \Pi_2\}$ all postfix constraints of an activity.

Declare constraints exhibit a hierarchy, which is well-explained in [117, 22]. For unary constraints, *existence(a,n)* and *absence(a,n)* together form *exactly(a,n)*. Binary constraints are divided in different classes, for which every class is an extension of its predecessors. *Unordered* constraints include *responded existence(a,b)* and *co-existence(a,b)*, of which the latter is the two-way version of the former. The *simple ordered* constraints include *precedence (p), response (r), and succession (s)(a,b)*, of which *precedence(a,b)* and *response(a,b)* are both extensions of *responded existence(a,b)*. *Alternating ordered* constraints include *alternate p,r,s(a,b)*, and *chain ordered* constraints include *chain p,r,s(a,b)*. Next to these constraints, there exist negative versions for three of them, i.e., *not co-existence(a,b), not succession(a,b)*, and *not chain succession(a,b)*. Finally, the *choice(a,b)* constraint exists, which is comparable with a branched unary constraint *existence({a,b},1)*. For binary constraints, *(alternate/chain) response(a,b)* and *(alternate/ chain) precedence(a,b)* form *(alternate/chain) succession* respectively. When a property is discussed for, e.g., *precedence/response*, this hence also includes *(chain/alternate) succession*. In the remainder of the text, the set of, e.g., *precedence* constraints is denoted as $\Pi_{prec} \subseteq \Pi$ with $\Pi_{chaiprec} \subseteq \Pi_{altprec} \subseteq \Pi_{prec}$.

A constraint can have multiple states. First of all, a constraint can be in an *accepting* state, i.e., the constraint is satisfied. In the automaton representation, accepting states are indicated with a double circle. Secondly, a constraint can be violated, i.e., it cannot get satisfied anymore. In the automaton representation, this is not included, as this state should not be reachable. Consider for example *precedence(a,b)*. Firing *b* before *a* would render that constraint violated. Finally, a constraint can be temporarily violated, i.e., it is not in an accepting state, but it can still reach that state. In the automaton representation, this is represented by a regular circle for which there still exists a path to an accepting state. Consider for example *response(a,b)*. Firing *a* brings this constraint to a temporarily violated status, however, firing *b* afterwards brings the constraint to an accepting state.

There exists a graphical notation for each of the constraints, for which we refer to [183] and [117]. All graphical instantiations of Declare models in this thesis will be accompanied by their corresponding name. In Figure 3.1, a small Declare model is introduced of which the description is as follows. Two binary and three unary constraints are present. First of all, you can only pay a bill after receiving one, and receive one only after you have paid the previous one. Eventually, you always have to pay the bills you receive. You have to receive at least one bill, and you can only end the subscription right after you paid a bill. Unsubscribing is only possible once, at the end of the execution. The corresponding FSM of the whole model is displayed next to the standard graphical notation. In essence, the behavior is quite straight-

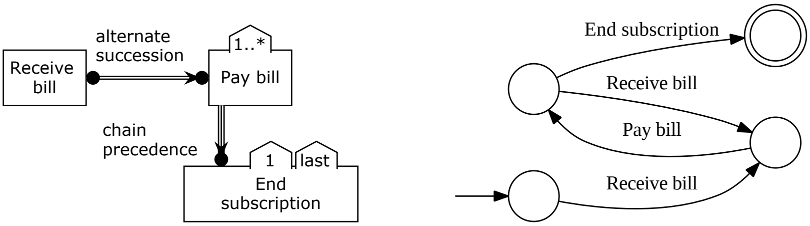


Figure 3.1: Example of a small Declare model describing a payment process in Declare standard notation on the left and its corresponding finite state machine to the right.

forward, with only four states. The initial state of an FSM is indicated with an incoming arrow not connected to a previous node, and accepting states are indicated with a double circle.

3.2.3 Weighted R/I-nets

In this chapter, a formalization to express Declare constraints in the form of weighted Petri nets with reset and inhibitor arcs is proposed and in Chapter 6, the same formalization is used to compare Declare with Petri nets. Finally, in Chapter 8, Petri nets are also used to verify the behavior of mixed-paradigm models. Therefore, the basic principles of Petri nets are introduced.

Petri nets [113] are a mathematical modeling language used to describe distributed, concurrent systems. A weighted Petri net with reset and inhibitor arcs is a directed graph, expressed as a tuple, $PN = (P, T, F, R, I, W)$, with P a finite set of places (visually represented as circles), T a finite set of transitions (visually represented as boxes) with $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the set of normal arcs (shown as arcs with a single arrow). Let $W : F \to \mathbb{N}$ determine a weighting function which associates a weight to each arc. Let $R : T \to \mathbb{P}(P)$ define the reset places (with $\mathbb{P}(P)$ the powerset of P) and $I : T \to \mathbb{P}(P)$ the inhibitor places for each transition, which also implicitly define the reset arcs (shown as an arc ending with double arrows) and inhibitor arcs (shown as an arc ending with a circle) respectively. The set of input nodes of a node $x \in P \cup T$ is denoted as $\bullet x = \{(y \in P \cup T \,|\, (y, x) \in F) \vee (x \in T \wedge y \in R(x) \cup I(x))\}$, and the output nodes similarly as $x\bullet$.

The state of a Petri net is called marking $M \in P \to \mathbb{N}$, indicating the number of tokens contained in each place. A transition t is said to be enabled,

Template	**LTL Formula [118]**	**Regular Expression [185]**	**R/I-Net Constructs**
Existence(A,n)	$\Diamond(A \land \bigcirc(existence(n-1,A)))$	.*(A.*){n}	$F = \{(A,p_1),(p_1,t_{sink})\}, W(p_1,t_{sink}) = n$
Absence(A,n)	$\neg existence(n,A)$	[^A]*(A?[^A]*){n-1}	$F = \{(t_{source},p_1),(p_1,t_A)\}, W(t_{source},p_1) = n-1$
Exactly(A,n)	$existence(n,A) \land absence(n+1,A)$	[^A]*(A[^A]*){n}	$F = \{(t_{source},p_1),(p_1,t_A)\}, W(t_{source},p_1) = n, I(t_{sink}) = p_1$
Init(A)	A	(A.*)?	$F = \{(t_{source},p_1)\}, \forall t \in T \setminus t_A, I(t) = p_1$
Last(A)	$\Box(A \implies \neg X \neg A)$	.*A	$F = \{(t_A,p_1),(p_1,t_{sink})\}, \forall t \in T \setminus t_A, R(t) = p_1$
Responded existence(A,B)	$\Diamond A \implies \Diamond B$	[^A]*((A.*B.*) |(B.*A.*))?	$F = \{(t_A,p_1),(t_B,p_2),(p_2,t_\lambda),(t_\lambda,p_3)\}, I(t_A) = p_3, I(t_B) = p_3, I(t_{sink}) = p_1, R(t_\lambda) = \{p_1,p_2\}$
Co-existence(A,B)	$\Diamond A \impliedby \Diamond B$	[^AB]*((A.*B.*) |(B.*A.*))?	$F = \{(t_A,p_1),(t_A,p_4),(t_B,p_2),(t_B,p_4),(p_1,t_\lambda),(p_2,t_\lambda),(t_\lambda,p_3)\}, I(t_A) = p_3, I(t_B) = p_3, I(t_{sink}) = p_4, R(t_\lambda) = \{p_1,p_2,p_4\}$
Response(A,B)	$\Box(A \implies \Diamond B)$	[^A]*(A.*B)*[^A]*	$F = \{(t_A,p_1)\}, I(t_{sink}) = p_1, R(t_B) = p_1$
Precedence(A,B)	$(\neg B\, U A) \lor \Box(\neg B)$	[^B]*(A.*B)*[^B]*	$F = \{(t_A,p_1),(t_B,p_1),(p_1,t_B)\}$
Succession(A,B)	$response(A,B) \land precedence(A,B)$	[^AB]*(A.*B)*[^AB]*	$F = \{(t_A,p_1),(t_B,p_1),(p_1,t_B),(t_A,p_2)\}, I(t_{sink}) = p_2, R(t_B) = p_2$
Alternate response(A,B)	$\Box(A \implies \bigcirc(\neg A\, U B))$	[^A]*(A[^A]*B[^A]*)*	$F = \{(t_{source},p_1),(p_1,t_A),(t_A,p_2),(t_B,p_1)\}, I(t_{sink}) = p_2, R(t_A) = p_1, R(t_B) = p_2$
Alternate precedence(A,B)	$precedence(A,B) \land \Box(B \implies \bigcirc(precedence(A,B))$	[^B]*(A[^B]*B[^B]*)*	$F = \{(t_A,p_1),(p_1,t_B)\}, R(t_B) = p_1$
Alternate succession(A,B)	$altresponse(A,B) \land precedence(A,B)$	[^AB]*(A[^AB]*B[^AB]*)*	$F = \{(t_{source},p_1),(p_1,t_A),(t_A,p_2),(p_2,t_B),(t_B,p_1)\}, I(t_{sink}) = p_2$
Chain response(A,B)	$\Box(A \implies \bigcirc B)$	[^A]*(AB[^A]*)*	$F = \{(t_A,p_1)\}, I(t_A) = p_1, I(t_C) = p_1, R(t_B) = p_1$
Chain precedence(A,B)	$\Box(\bigcirc B \implies A)$	[^B]*(AB[^B]*)*	$F = \{(t_{source},p_1),(t_B,p_1),(t_C,p_1)\}, I(t_B) = p_1, R(t_A) = p_1$
Chain succession(A,B)	$\Box(A \iff \bigcirc B)$	[^AB]*(AB[^AB]*)*	$F = \{(t_{source},p_2),(t_A,p_1),(t_B,p_2)\}, I(t_A) = p_1, I(t_B) = p_2, I(t_C) = p_1, R(t_A) = p_2, R(t_B) = p_1$
Not co-existence(A,B)	$\neg(\Diamond A \land \Diamond B)$	[^AB]*((A[^B]*) |(B[^A]*))?	$F = \{(t_A,p_1),(t_B,p_2)\}, I(t_A) = p_2, I(t_B) = p_1$
Not succession(A,B)	$\Box(A \implies \neg(\Diamond B))$	[^A]*(A[^B]*)*	$F = \{(t_A,p_1)\}, I(t_B) = p_1$
Not chain succession(A,B)	$\Box(A \implies \neg(\bigcirc B))$	[^A]*(A+[^AB][^A]*)*A*	$F = \{(t_A,p_1)\}, I(t_B) = p_1, R(t_C) = p_1$
Choice(A,B)	$\Diamond A \lor \Diamond B$	.*[AB].*	$F = \{(t_{source},p_1)\}, I(t_{sink}) = p_1, R(t_A) = p_1, R(t_B) = p_1$
Exclusive choice(A,B)	$(\Diamond A \lor \Diamond B) \land \neg(\Diamond A \land \Diamond B)$	([^B]*A[^B]*) |.*[AB].*([^A]*B[^A]*)	$F = \{(t_{source},p_2),(t_A,p_1),(t_B,p_3)\}, I(t_A) = p_3, I(t_B) = p_1, I(t_{sink}) = p_2, R(t_A) = p_2, R(t_B) = p_2$

Table 3.1: An overview of Declare constraint templates with their corresponding LTL formula, regular expression, and R/I-net constructs.

Template	**Description**
Existence(A,n)	Activity A happens at least n times.
Absence(A,n)	Activity A happens at most n times.
Exactly(A,n)	Activity A happens exactly n times.
Init(A)	Each instance has to start with activity A.
Last(A)	Each instance has to end with activity A.
Responded existence(A,B)	If A happens at least once then B has to happen or happened before A.
Co-existence(A,B)	If A happens then B has to happen or happened after A, and vice versa.
Response(A,B)	Whenever activity A happens, activity B has to happen eventually afterwards.
Precedence(A,B)	Whenever activity B happens, activity A has to have happened before it.
Succession(A,B)	Both Response(A,B) and Precedence(A,B) hold.
Alternate response(A,B)	After each activity A, at least one activity B is executed. A following activity A can be executed again only after the first occurrence of activity B.
Alternate precedence(A,B)	Before each activity B, at least one activity A is executed. A following activity B can be executed again only after the first next occurrence of activity A.
Alternate succession(A,B)	Both alternative response(A,B) and alternate precedence(A,B) hold.
Chain response(A,B)	Every time activity A happens, it must be directly followed by activity B (activity B can also follow other activities).
Chain precedence(A,B)	Every time activity B happens, it must be directly preceded by activity A (activity A can also precede other activities).
Chain succession(A,B)	Activities A and B can only happen directly following each other.
Not co-existence(A,B)	Either activity A or B can happen, but not both.
Not succession(A,B)	Activity A cannot be followed by activity B, and activity B cannot be preceded by activity A.
Not chain succession(A,B)	Activities A and B can never directly follow each other.
Choice(A,B)	Activity A or activity B has to happen at least once, possibly both.
Exclusive choice(A,B)	Activity A or activity B has to happen at least once, but not both.

Table 3.2: An overview of Declare constraint templates with their corresponding textual descriptions.

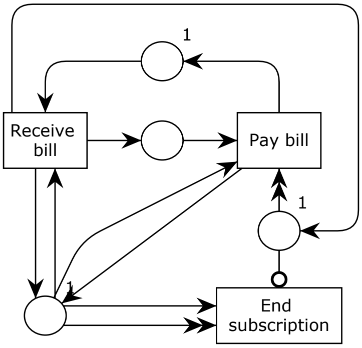


Figure 3.2: Example of a small R/I-net describing the same payment process. Note that the model is both representation and execution at the same time.

denoted as $M[t\rangle \iff M(p) > 0, \forall p \in \bullet t : [(p,t) \in F \lor p \in R(t)] \land M(p) = 0, \forall p \in I(t)$. Firing an enabled transition results in a new marking M' so that $M'(p) = M(p) - (M(p)$ iff $p \in R(t), W(p,t) \iff (p,t) \in F$, 0 otherwise) + $(W(t,p) \iff (t,p) \in F$, 0 otherwise). That is, tokens are removed from input places according to arc weights. Places which act as reset places for a fired transition are emptied completely. Subsequently, the token count of output places is incremented according to arc weights to obtain the new marking. The initial state of the model is called the initial marking, denoted M_0. For more details, the interesting reader is referred to [113].

An over-convoluted approximation of the example in Section 3.2.2 is given in Figure 3.2 to illustrate the constructs of R/I-nets. The figures next to the places indicate the initial marking, or the number of tokens present in the place in the initial state of the net.

3.3 Conversion of Declare Constraints to Weighted R/I-nets

This section describes the conversion of each Declare constraint into their dedicated weighted R/I-nets. First, the general setup is explained, next each constraint is reviewed independently, and finally it is illustrated how multiple constraints can be used in a model. Despite the fact that elements from a set are indicated in lower case and sets in upper case throughout the thesis, activities A, B, and C used in the examples in the following sections are also instantiations.

3.3.1 Conversion templates

The purpose of the conversions is to capture constraints which are usually expressed in Büchi automata that yield ω-regular languages, or finite state automata, which yield regular languages. This means that the Petri net templates, if behaviorally equivalent, produce the same (type of) languages. However, the benefit of high-level Petri nets such as R/I-nets, are that they are Turing complete [147]. Still, the Petri nets that are constructed here produce regular languages. This is the case when the net is in clean standard form [62]. Hence, the following guiding principles are followed.

For every letter in the Declare template/model alphabet, one is defined in the Petri net alphabet as well $\Sigma_{PN} = \Sigma_{dec} \cup \{\lambda_{Invisible}, \lambda_{Start}, \lambda_{End}\}$, with labeling function $\delta : T \rightarrow \Sigma_{PN}$. As such, PN has a non-lamda free language L^{λ}. Invisible transitions (labeled $\lambda_{Invisible}$) are used for two constraints and are referred to λ_{Start} and λ_{End} as *Start* and *End* respectively. Their corresponding nodes are referred to as t_{source} and t_{sink} with $\delta(t_{source}) = \lambda_{Start}$ and $\delta(t_{sink}) = \lambda_{End}$. t_{source} is used to start the net by filling helper places used for the constraints, which need tokens to inhibit constraints which are (temporarily) violated by default. For example, an *existence(a,n)* constraint puts the model in a temporarily violated state when initialized, only to reach an accepting state after activity a has fired at least n times. t_{sink} empties the net completely and functions as an indicator for the state of the net as well. If t_{sink} is enabled, this means the net is in an accepting state, otherwise, there are places tied to constraints inhibiting it from firing. The end transition empties all places upon firing by using reset arcs, leaving the net in its final marking, comprising of only one token in the final place p_{sink}. Hence PN has an *L-type* ending. According to [62], as all template nets are in clean standard form, they yield a regular language. Indeed, unless the net is in a violated state (marking) which cannot reach an accepting state anymore (permanent violation), there exists a firing sequence reachable from that marking which ends with the sink transition. t_{sink} empties the net and is the sole terminal on the right hand side of that marking. Also, since none of the binary constraints contain any cardinality-based execution patterns, none of the Declare Petri nets need context-free grammar constructs such as $a^n b^n$. The R/I-net constructs for each template can be found in Table 3.1. In the construct templates, it is assumed that $T = \{t_{sink}, t_A(, t_B, t_C), t_{sink}\}, t_C = T \setminus \{t_A, t_B, t_{source}, t_{sink}\}$, $P = \{p_{source}, p_{sink}, p_1(, p_2, p_3, p_4)\}$, and $F = \{(p_{source}, t_{source}), (t_{sink}, p_{sink})\}$.

Note that the activities constraint by *init* and *last* cannot be substituted by t_{source} and t_{sink}, unless the activities these constraints refer to also have to be executed exactly once. Otherwise, they would recreate the initial marking over and over again (*Init*, as t_{source} initializes the net), or be inhibited from firing multiple times (*last*, as t_{sink} empties the net). As such, *init(a)* is instead modeled as a fragment containing a single place which is filled by t_{source} and inhibits every transition but t_A. The *last(a)* constraint is enforced by a single place which gets filled by t_A and is reset by all the other transitions but t_A.

Many helper places are introduced which are used to enforce and indicate the state of the constraints in a one-to-one fashion. This means that one can trace back every constraint in the net to a certain Declare template. This is not possible for example, in the approach of [124], but yields benefits for, e.g., model and conformance checking, where in an execution the violation can be traced back to a certain constraint and the activities involved.

Figure 3.3 provides an example of converting a full Declare model in standard notation to its converted Petri net counterpart in Figure 3.4. For this purpose, a Declare model is included with three unary and three binary constraints. *Deposit money* is executed first, and at least once. *Request credit card* can only occur exactly after *Deposit money* and exactly once. *Deposit money* precedes *Withdraw money.* Every new occurrence of *Print statement* must be preceded by at least one new occurrence of *Withdraw money.* Note that opportunities exist for simplifying the resulting Petri net model, which was not done here for the sake of understandability of the separate converted constraints; in the following paragraphs, we discuss the conversion of each of the Declare constraint types to their corresponding Petri net templates and how to create a model out of a full Declare model.

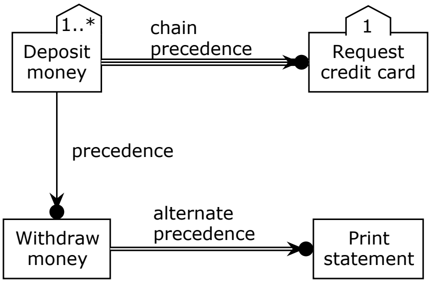


Figure 3.3: Simple Declare model containing 3 unary and 3 binary constraints.

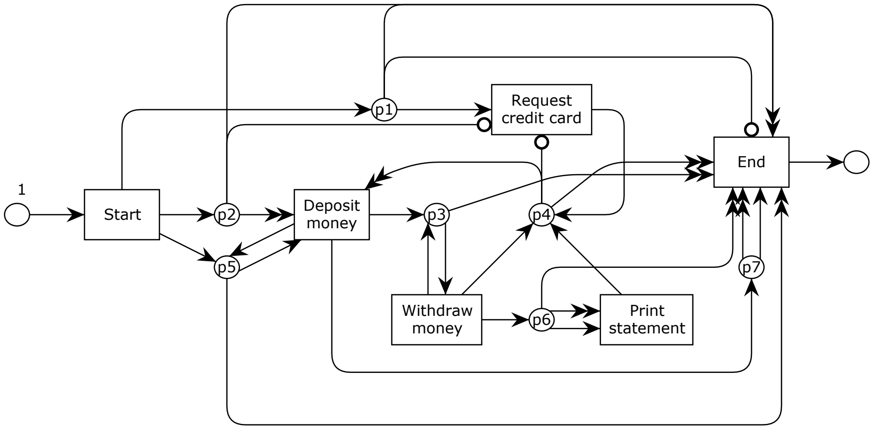


Figure 3.4: The same model as in Figure 3.3 converted into an R/I-net. The net contains 7 places (besides p_{source} and p_{sink}) of which 6 are used for the constraints and one, p_5, which adds a self-loop for *Deposit Money*. p_1 is used for *exactly(Request credit card,1)*, p_2 for *init(Deposit money)*, p_3 for *precedence(Deposit money, Withdraw money)*, p_4 for *chain precedence(Deposit money, Request credit card)*, p_6 for *alternate precedence(Withdraw money, Print statement)*, and p_7 for *existence(Deposit money,1)*. Many reset arcs are connected from these places to t_{sink} to empty the net upon firing this transition.

Unary Constraints

Unary constraints (shown in Figure 3.5) focus mainly on the enforcement of cardinalities of the activity involved. In these scenarios, the source and sink transitions prove useful already. *Existence(A,n)* requires an activity A to be executed at least n times. Therefore, a helper place p_1 is installed, that can only fire after n tokens are collected in p_1, by introducing an arc with weight n connected to t_{sink}.

Absence(A,n) translates into putting $n-1$ tokens in p_1, which will prevent the activity from firing n or more times. The *exactly(A,n)* constraint is similar, but p_1 inhibits t_{sink} from firing until the activity has happened at least n times.

Binary Existence Constraints

These constraints (shown in Figure 3.6) are the hardest to capture in a Petri net template, as they require some sort of memory to keep track of the fact whether the antecedent and consequent have occurred before. For *responded existence*, the net cannot end before a first occurrence of the consequent when the antecedent fires. The net thus has to keep track whether the

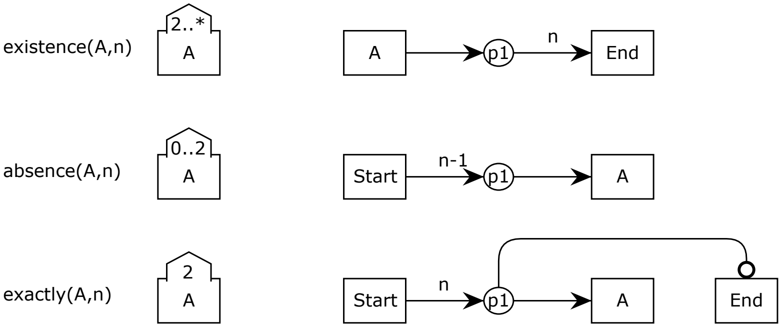


Figure 3.5: The mapping of unary constraints *existence(A,n), absence(A,n)*, and *exactly(A,n)*. Note that the t_{source} and t_{sink} transitions (labeled *Start* and *End*) are only shown when necessary in this and following figures.

consequent has fired already, in case a termination is sought after the firing of the antecedent. For this purpose, an invisible activity is used which, when fired, leaves no transition enabled but the last one, acting as a placeholder for t_{sink}. The same principle can be applied for *co-existence*, which is the two-way version of *responded Existence.*

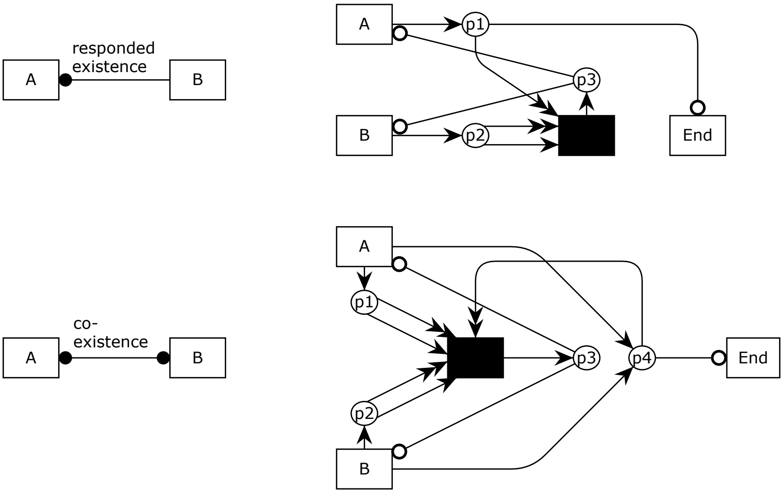


Figure 3.6: The mapping of the binary existence constraints *responded*, and *co-existence.*

Simple Ordered Constraints

Response requires a helper place that inhibits t_{sink} as long as the consequent of the constraint is not fired after an occurrence of the antecedent. *Precedence* also needs an additional input place for the antecedent, which serves to enable it after the firing of the consequent. Afterwards, the consequent keeps itself enabled indefinitely. *Succession* is, similar to the LTL formula, the combination of both constraints. The constraints are shown in Figure 3.7.

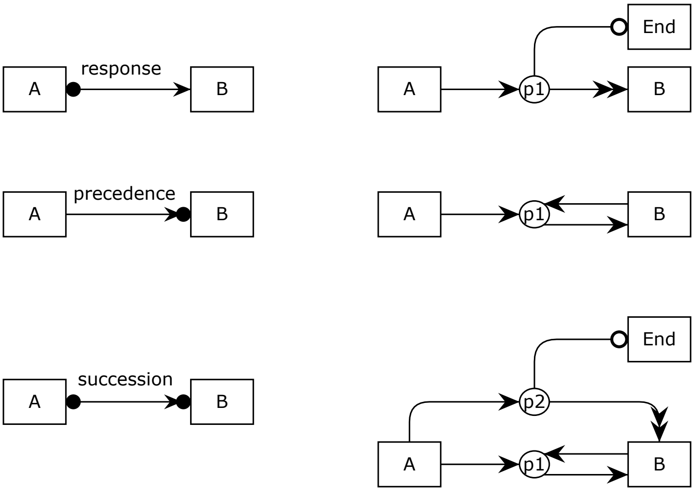


Figure 3.7: The mapping of the simple ordered constraints *Response, Precedence*, and *Succession.*

Alternating Ordered Constraints

Alternating constraints (shown in Figure 3.8) are a variant of the simple ordered constraints to express loops. Modeling *alternate response* is somewhat tedious. First of all, two helper places need to be introduced, p_1 and p_2. p_1 is an input place of t_{source}, resulting in an initial marking of 1. Next, it serves both as an input for the antecedent, and also a reset place. The consequent can fire any time, but whenever the antecedent is fired, t_{sink} is inhibited until the occurrence of the consequent, which frees p_2 with a reset operation. Also, the consequent delivers a new token to the input place of the antecedent. *Alternate precedence* is much more straightforward. Whenever the antecedent is fired, it resets its input place, which models the fact that it can only occur after any new occurrence of the consequent. Again, *alternate succession* is a combination of the two, but can be reduced to a smaller mapping. Two places are required, one as an input place for the antecedent, enabled by default (marking of 1), and another one serving as an input place for the consequent, which will also inhibit t_{sink}, avoiding a violation of the rule.

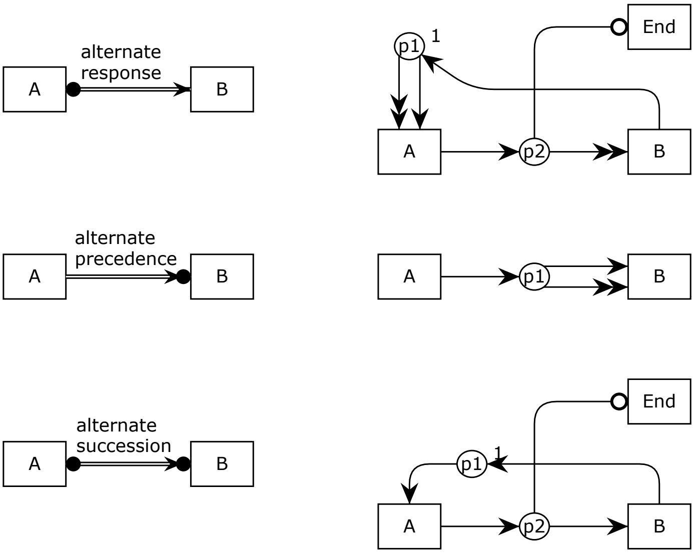


Figure 3.8: The mapping of the alternating ordered constraints *alternate response, precedence*, and *succession. Start* is omitted, but provides the initial marking of the helper places (e.g. one token in p_1 for *alternate response*).

Chain Ordered Constraints

The chain constraints (shown in Figure 3.9) extensively make use of the *next* LTL operator. To incorporate the setup in which whenever the antecedent is fired, the consequent needs to be next, a helper place is introduced which inhibits every other activity in the net but the consequent. The other activities (besides the antecedent) are indicated in Figure 3.9 as *C*. The chain precedence constraint in its turn requires a helper place that inhibits the execution of the antecedent, which can only fire right after an execution of the consequent, which frees the inhibiting place with a reset arc and gets filled by any other activity. The combination of both constraints allows only the execution of the antecedent, and afterwards inhibits every other activity but the consequent, resulting in a strict *ABAB...* pattern.

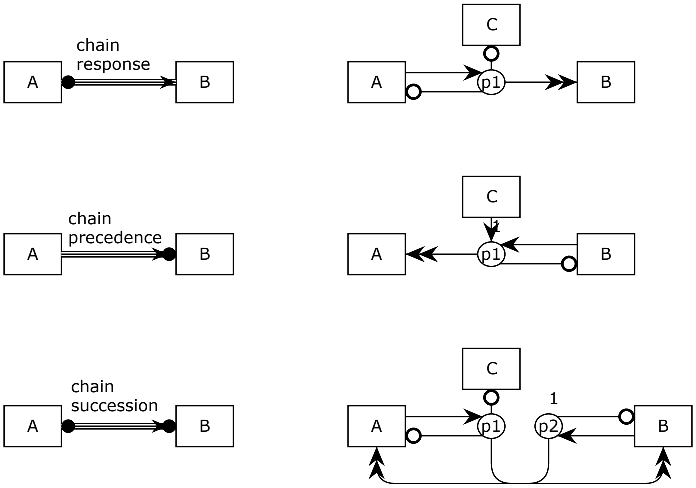


Figure 3.9: The mapping of the chain ordered constraints *chain response, precedence*, and *succession*.

Negative Constraints

The negative constraints (shown in Figure 3.10) also use inhibitor places to model violation. *Not co-existence* uses two helper places, one for each activity, which, when containing a marking, block the execution of the other activity. *Not succession* uses one place to inhibit the consequent from executing after the antecedent is ever fired. *Not chain succession* inhibits the consequent from happening exactly after the execution of the antecedent. Again, all the other activities need to be connected to the inhibiting place with a reset arc, freeing the consequent from the marking imposed by the antecedent.

Choice Constraints

The simple *choice* template inhibits t_{sink} as long as not one of the two involved activities have fired, modeled with two reset arcs, linking both activities to the place. The *exclusive choice*, is similar to the *not co-existence* constraint, but inhibits t_{sink} from firing as long as not one of the two (which is the only possibility) has fired. The helper places in both constraints are marked with a token by default, and are linked to t_{source}. The constraints are shown in Figure 3.11.

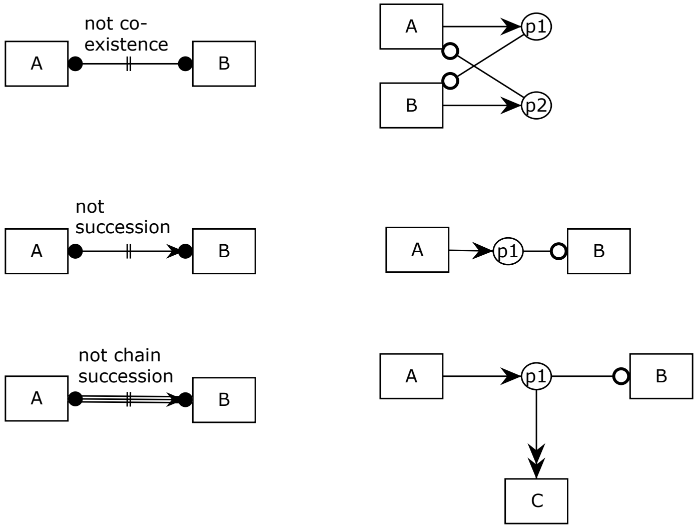


Figure 3.10: The mapping of the negative constraints *not co-existence, succession*, and *chain succession*.

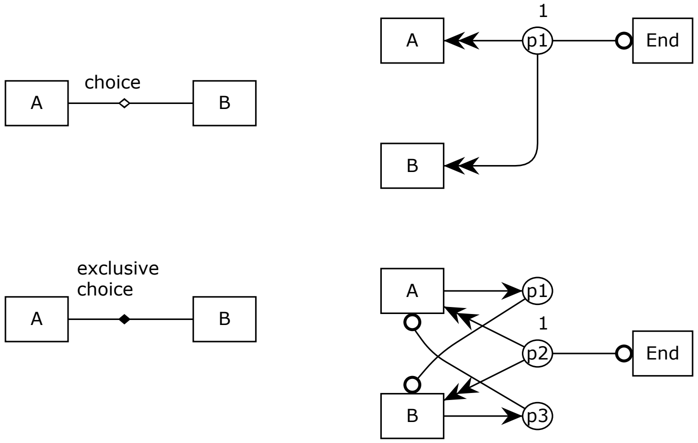


Figure 3.11: The mapping of the choice constraints *choice*, and *exclusive choice*.

Merging Constraints into a Model

As indicated previously and as shown in the example model (Figure 3.4), it is possible to merge a set of constraints, having converted these to separate Petri net templates, into one single model as follows. First, note that

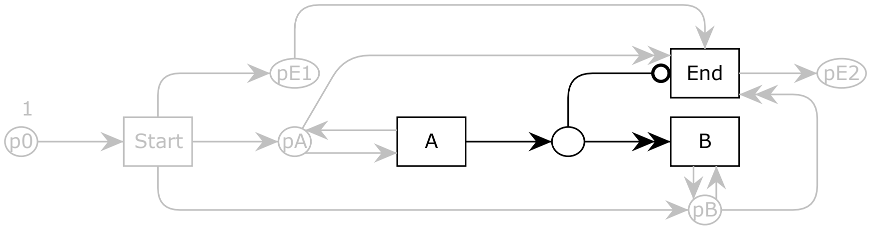


Figure 3.12: Example of the *response* constraint in a model. The constructs specific to the constraint are in black, the ones used for the model are in gray.

since the constraints are modeled as separate regions, the overlap of places is void. Second, transitions are merged so that the input and output arcs for each transition form the union of all input and output arcs in the separate templates respectively. Third, activity transitions that do not have an input place with corresponding input arcs (not a reset or inhibitor arc) after performing the steps above, are connected with t_{source} with an extra place, which also keeps these transitions enabled after firing by means of a self-loop (i.e. an arc from the transition to this extra place is also added). When t_{sink} does not have an input place, the same rule applies (i.e. also connected with t_{source}, but without a self-loop. Fourth, a unique source and sink place, p_{source} and p_{sink}, are added, and every place in the model except p_{source} and p_{sink} are reset by firing t_{sink} (by adding reset arcs). Note that these steps *also need to be performed when only dealing with the conversion of one single Declare constraint*. For the sake of clarity, however, the adding of these extra constructs is not shown in the separate constraints listed above, but is done in the example model in Figure 3.4.

An additional example of how a sole constraint fits into a model is given in Figure 3.12. The constructs used for the *response(A,B)* constraint are depicted in black and are supplemented by p_0, t_{source}, p_A and p_B which act as the self-loops places of A and B, and the input place p_{E_1} and output place p_{E_2} of *End*. These constructs are indicated in gray and are only used once, even when merging multiple constraints between A and B, while the constructs indicated in black are specific to *response(A,B)*. Two reset arcs are connected to the self-loop places to empty the net after firing t_{sink}.

n-Ary and Target-Branched Declare Constraints.

Some Declare constraints, such as *choice*, have multiple variants such as *choice1of3(A,B,C)*, *choice2of3(A,B,C)*, etc. These variants can be easily expressed in R/I-nets as well. One has to simply add multiple transitions to p_1 in Figure 3.11, one for every activity. *Choice2of3* can be modeled with two places, etc.

Target-branched constraints, as explained in Section 3.2.2, can also be modeled with R/I-nets. We do not elaborate on this for every constraint but rather give an example in the form of *response(A,{B,C})*. Both B and C can resolve the *response* constraint, hence the addition of C to *response(A,B)* would simply require an extra reset arc from p_1 to t_C.

3.4 Equivalence Analysis

In this section, it is shown how the separate Declare constraint conversions to R/I-nets are behaviorally equivalent to the FSAs. First, the notion of state spaces is explained. Next, each constraint is analyzed by simulating it for a vast number of executions and subsequently mining the generated event log.

3.4.1 State Spaces and Automata

A Petri net's reachability graph is an exhaustive enumeration of all states in the net that can be reached from the initial marking [128]. The reachability graph of a bounded Petri net is a transition system constructed as follows. The initial marking is the initial state. Every reachable marking from M_0 is a state. Transitions between pairs of states represent the transitions that lead from a marking to another by means of a firing. A state in which no transitions are enabled anymore is called a final state. The reachability graph can be represented as a Kripke structure. We define such structures as follows. A Kripke structure is a tuple $KS = (S, I, R, L)$ with

- S a finite set of states
- $I \subseteq S$ a set of initial states
- $R \subseteq S \times S$ the transition relations which are left-total ($\forall s \in S, \exists\, s'$ s.t. $(s, s') \in R$)
- L a labeling function $L : S \to \mathbb{P}(\Sigma_{AP})$ for alphabet Σ_{AP}, with $\mathbb{P}$ denoting the powerset.

Translated to a Petri net, every state $s \in S$ is a marking in the net, $S = M$, with an outgoing arc for every enabled transition in that marking. Hence $\Sigma_{AP} = \Sigma_{PN}$, $L : M \to \mathbb{P}(\Sigma_{PN})$. Since a dedicated source place and transition is used, $I = M_0$ with $M_0(p_{source}) = 1$. We use C to represent all other activities $\Sigma_{AP} \setminus \{A(,B), \lambda, \lambda_{Start}, \lambda_{End}\}$, of which the latter are indicated as *Begin* and *End*.

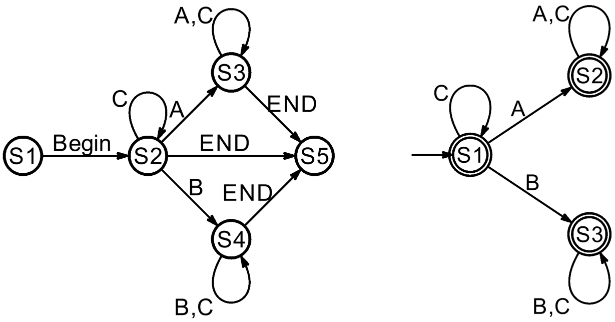


Figure 3.13: Example of the conversion of a Petri net fragments' state space to an equivalent FSA for the *not co-existence* constraint. To the left, the state space for the *not co-existence* Petri net template is depicted with $S = \{S1, S2, S3, S4, S5\}$, $AP = \{Begin, A, B, C, END\}$, $R = \{(S1, S2), (S2, S3), (S2, S4), (S2, S5), (S3, S5), (S4, S5)\}$, $L = \{(S1, \{BEGIN\}), (S2, \{A, B, C, END\}), (S3, \{A, C, END\}), (S4, \{B, C, END\}), \}$, and $I = \{S1\}$. To the right, the equivalent FSA is given.

In order to match this with Declare automata, the reachability graph of every Petri net is converted into a finite state automaton. Since the constraints yield regular languages, this conversion also yields a regular language model. The automaton is defined as a tuple, $\Phi_{PN} = (Q, \Sigma_c, Q_0, \Delta, A_c)$ with:

- Q the finite set of states,
- Σ_c the finite alphabet,
- $Q_0 \subseteq Q$ the initial states,
- $\Delta \subseteq Q \times \sigma \times Q$ the transition relations,
- $A_c \subseteq Q$ a set of accepting states.

Then:

- $Q = M \cup I$,
- $Q_0 = I$,
- $\Sigma_c = \Sigma_{PN}$,
- $\forall s, s' \in Q, a \in \Sigma_c : (s, a, s') \in \Delta \iff L(s') = a \wedge (((s, s') \in R) \vee (s = s^i \wedge s' = s^0))$,
- $\forall a \in A_c, \lambda_{End} \in a$.

This can be done for every Petri net template included in the previously introduced conversions. The same principle can also be applied for Büchi automata, as used in [117]. An example is included for the *not co-existence* template in Figure 3.13, which displays the state space to the left, and the corresponding finite state automaton to the right. Every automaton includes all the accepting and temporarily violated states in the Declare automata, as for example included in [185]. Hence, the Petri nets' languages are equal to the Declare automata's languages. The other templates are included below in Figures 3.14 to 3.20.

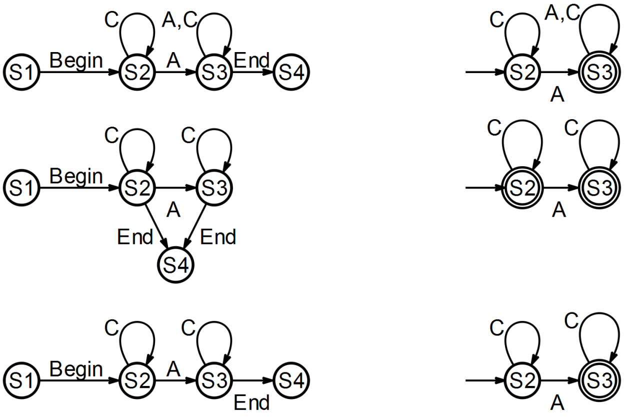


Figure 3.14: Conversion of *existence1, absence*, and *exactly1* constraints.

3.4.2 Empirical Validation by Simulation

The validity of the conversions was discussed in the previous section. Now it is shown how the conversions can be used in a practical setting by means of simulation as follows. First, traces of all the different Petri net templates are simulated separately and mined with Declare Maps Miner [95], a Declare process mining algorithm introduced in Chapter 2. Hence traces that are generated from the R/I-net are mined to a Declare model. Next, traces from different Declare constraints are simulated and replayed over the corresponding Petri net conversions, addressing the other direction by going from Declare model traces to replaying over a Petri net. It is assumed that the finite state and Büchi automata yield the same language over finite traces.

The basic setup for the first direction (traces generated from R/I-net mined to a Declare model) is performed by constructing a regular Petri net with a simple sequential process placed in a loop, which is then supplemented by one or two activities (unary vs. binary) that are constrained by a chosen Petri net template following a Declare conversion. The same considerations are used to create the model, i.e., t_{source}, t_{sink} and self-loops are added.

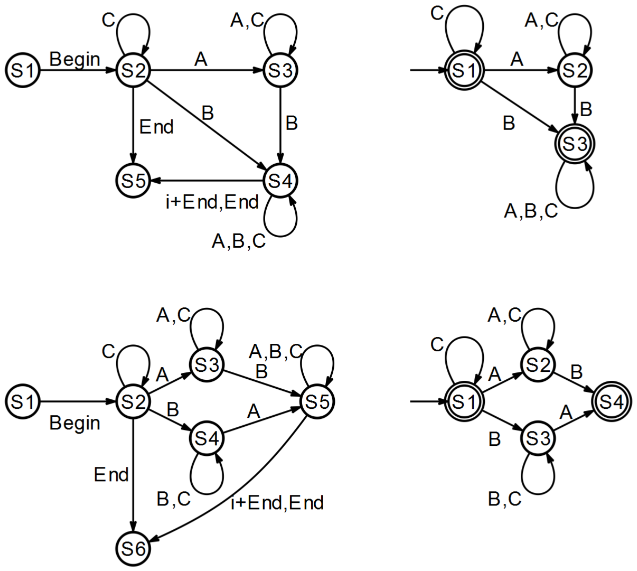


Figure 3.15: Conversion of *responded*, and *co-existence* constraints.

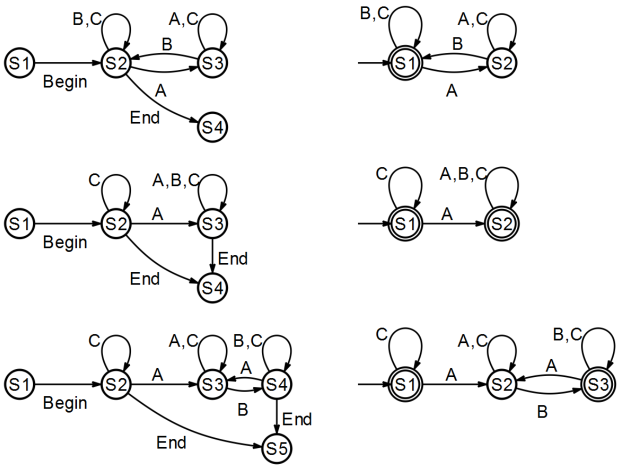


Figure 3.16: Conversion of *response, precedence*, and *succession* constraints.

CPN Tools [182, 181] is used to produce models, simulate them, and create an event log, in a fashion similar to [37]. An example for simulating *alternate response(B,Z)* is given in Figure 3.21, in which a straightforward loop

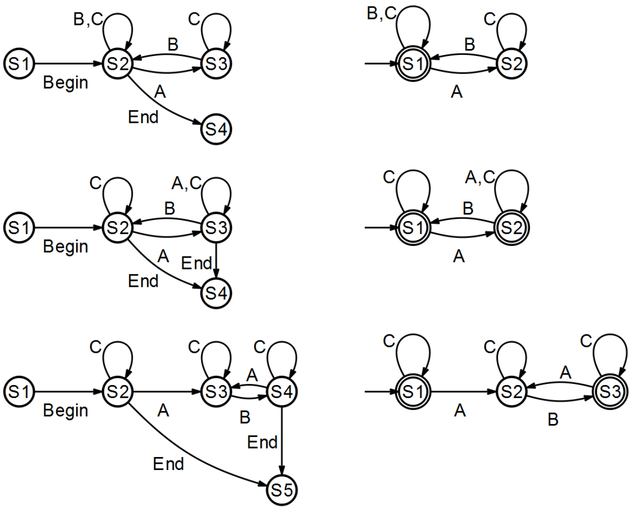


Figure 3.17: Conversion of *alternate response, precedence*, and *succession* constraints.

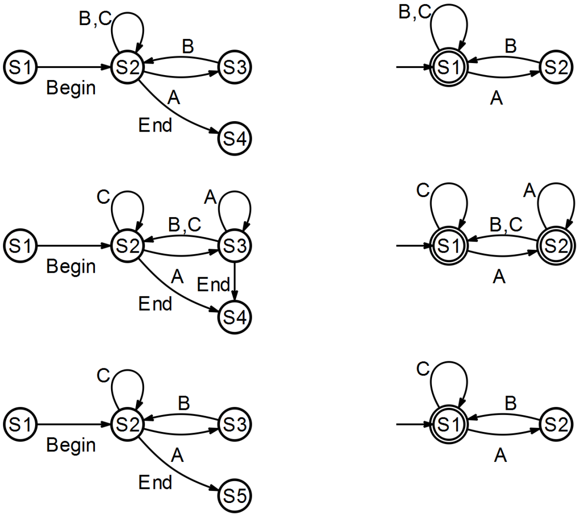


Figure 3.18: Conversion of *chain response, precedence*, and *succession* constraints.

is supplemented with the matching Petri net template. The places used for the self-loops insert the trace ID (indicated as the i variable in the Colored Petri net) with a colored token and the trace identifier is incremented every

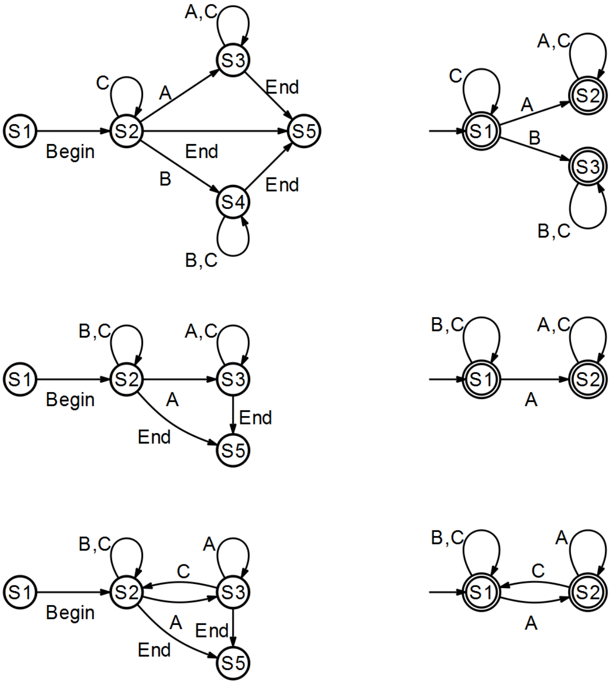


Figure 3.19: Conversion of *not co-existence, not succession*, and *not chain succession* constraints.

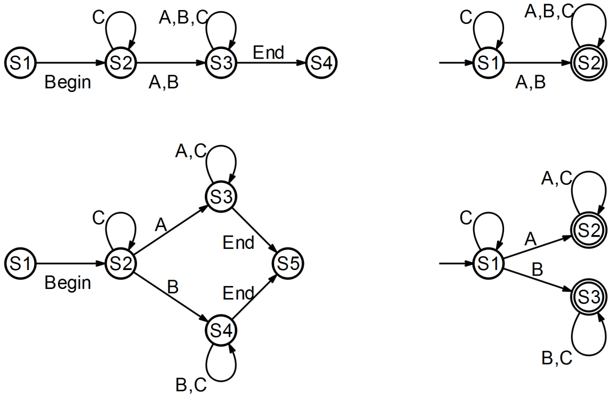


Figure 3.20: Conversion of *choice*, and *exclusive choice* constraints.

time t_{sink} is fired. Every simulated event log contains 100,000 traces. The resulting log is then mined with Declare Miner[1] afterwards. It is expected that, for each of the Petri net templates, the corresponding Declare constraint was found by the miner to be either present and supported 100%, or exhibiting a confidence of 100% [93], thus confirming the validity of the technique for this direction. Next, the simulation is performed the other way around (Declare model traces replayed over Petri net). A regular expression simulator is used, which uses the expressions in Table 3.1, in which the .-wildcard is substituted by C. The simulated traces are then replayed over the Petri net conversions in ProM using the conformance checking plugin used in [2]. When no tokens need to be inserted, the net is not too restrictive for discovering a certain Declare constraint.

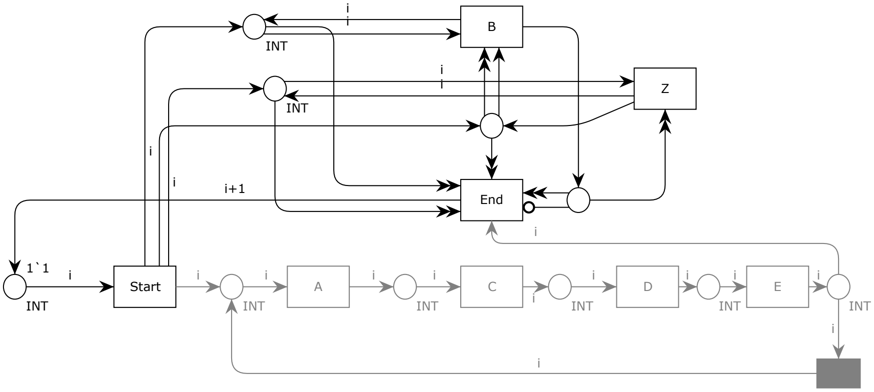


Figure 3.21: Example of the verification simulation for *alternate Response(B,Z)*. The transition actions for writing the log are omitted for clarity. Note that, again, *Start* and *End* serve the purpose of t_{source} and t_{sink} respectively, while also supporting the simulation by initiating and ending a new trace.

Both simulation strategies yield positive results. Declare Miner indeed reports a support and confidence of 100% for every process log generated from the conversions, making sure that they are at least as strict as their Declare counterparts. The replaying algorithm fittingly executed every single trace generated for the Declare constraints over the Petri net conversions, proving that the conversions are not looser (resulting in more behavior) than their Declare counterparts. The simulated Declare constraints could be replayed over the Petri net conversions without violation and token inserts, which means the conversions are not stricter than their regular expression

[1] Declare Miner is available in the ProM process mining framework, see: http://www.processmining.org/

counterparts. All the data generated and models used for the simulation have been made available online[2]. They are pluggable and can be used in different settings such as constructing Declare or mixed-paradigm models.

3.4.3 Bisimilarity of Converted Models

Although the separate constraints are bisimilar to their regular and ω-regular counterparts, the R/I-net conversions do not enjoy the same closure properties as finite and Büchi automata. Consider for example the model in Figure 3.22. The model contains just two constraints, *response(A,B)* and *existence(B,1)*. When executing B once, the sole time it can appear, the *End* transition is enabled, indicating an accepting state. Firing A after firing B for the last time renders the *response* constraint to a temporarily violated state. However, B cannot get fired anymore to resolve the temporary violation. This behavior is not enforced by the model, although this would be the case in the global automaton of the conjoined LTL formulas or regular expressions. The interrelations between the constraints are not taken into account by the conversions, as they are template-based, however, in Chapter 4, they will be studied in detail.

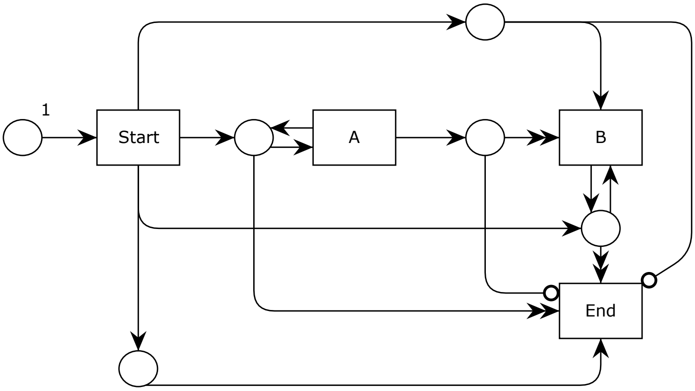


Figure 3.22: Example of a Petri net model that is not bisimilar to its FSA equivalent.

[2] http://j.processmining.be/dec2pet

3.5 Conclusion

This chapter presented the necessary formal introduction to the languages and their concepts used throughout the thesis. First, Declare was introduced with its various semantics. Next, Petri nets and its extensions were reviewed to give an overview of its capabilities as a modeling and execution language. Finally, the conversion of separate Declare constraints into R/I-nets was achieved and explained by means of examples and the intuition for bisimilarity on constraint level. The main use case of converting the models is the easy interchangeability of Declare and procedural languages. This is interesting for, e.g., mixed-paradigm models, as will be illustrated in Chapter 6, or the template-based use of R/I-nets, which embeds them in a more declarative approach towards process modeling, i.e., a constraint-based approach. Finally, it is also interesting to use R/I-nets for expressing Declare constraints in order to achieve greater expressiveness. As R/I-nets are Turing complete, they can enact more interesting constraints of the type $a^n b^n$, as counting in (weighted) Petri nets in general is better possible compared to regular languages. The downside, however, is that the R/I-net conversions cannot be used for recreating the behavior of models of multiple constraints if no inconsistencies are allowed, unless no constraint interdependencies exist as will be explained in Chapter 5.

Part II

Towards A Better Understanding of Declarative Process Modeling

CHAPTER 4

Uncovering Hidden Dependencies in Constraint-Based Declarative Process Models

"The meteorite is the source of the light
And the meteor's just what we see
And the meteoroid is a stone that's devoid of
the fire that propelled it to thee."
— Joanna Newsom, Emily

In the next part of this thesis, the properties of constraint-based declarative process models are studied in-depth in order to achieve new applications. More specifically, the underlit subject of the interplay between different constraints is studied. First, in this chapter, a detailed methodology is explained to retrieve the interaction of constraints in Declare models. Next, the applications of this approach are examined in Chapter 5.

The declarative process modeling approach has become a mature alternative to the procedural one, as proven in Chapter 2. The constructed models are well-capable of representing flexible behavior, as everything that is not allowed by the constraints in the model is possible during execution. The behavior that results from this setup, however, is more difficult to comprehend and requires a higher level of mental effort of both the modeler and the reader. Like procedural process models, the constraints still impose a certain sequence on the activities, but the current choice might have an impact on later execution possibilities, especially when multiple constraints are combined around one activity. Since constraints can be added to the model freely, it is often overseen what impact the combination of them has on the global behavior, i.e., the behavior supporting all the constraints at once.

This is often referred to in literature as the problem of hidden dependencies, i.e., dependencies that are not explicitly modeled or visualized. Especially in the context of the understandability of declarative process models it is interesting to make these dependencies explicit.

This chapter addresses the interplay of constraints used in order to get a grasp on the way different combinations convolute a model and inflict these hidden dependencies. By considering a Declare model as a graph and relying on the constraints' characteristics, it is delineated when and how such dependencies establish themselves. Then, it becomes possible to show users which constraints are related and hence the cognitive effort needed for comprehending the models can be lowered. This will be illustrated in Chapter 5, besides other applications including measuring the complexity, and re-engineering of a declarative process model.

4.1 Introduction

Instead of modeling predetermined paths of activities, declarative process models typically use constraints or rules to express what can, cannot, and must happen. Every execution sequence that is not strictly forbidden by the constraints can be enacted by the model, which makes it very flexible. Many outcomes are possible due to the interaction of the constraints over the activities. However, each type of constraint can have distinct effects on the enabledness of an activity, creating dependencies that are not explicit or visible in the graphical model and even in the execution semantics. These dependencies between constraints and activities that are not explicit or visible in the model [60, 64, 188] are so called hidden dependencies and make declarative models difficult to comprehend [64, 49].

This chapter introduces a technique capable of revealing hidden dependencies in constraint-based declarative process models by propagating the constraints' properties through the activities of a model. Every constraint can be assessed for these properties and used to build dependency structures for the whole model. In this way, it is aimed to address three out of four suggestions for improvement that were found in the user study performed in [64], i.e., 'Simplify combination on constraints', 'Make hidden dependencies explicit', and 'Use modularization'.

The structure of the rest of the chapter is as follows. First, Section 4.2 formalizes the idea of activities' upper and lower bounds and the concept of hidden dependencies in constraint-based declarative process models. Next, in Section 4.3, the construction of constraint dependency structures is elaborated on. Finally, the conclusion is presented in Section 4.4.

4.2 Hidden Dependencies

As explained in Section 3.2.2, the execution of a constraint-based declarative process model can be realized by constructing an automaton (either a Büchi [117] or finite state automaton [22, 185] for Declare) by conjoining the different constraints' automata to obtain the behavior that is allowed for by all of them. This conjunction actually abolishes the notion of the separate constraints and thus throws away the information of how the separate constraints interact. Consider for example the model in Figure 4.3 and its corresponding FSA displayed in Figure 4.4. One technique that exists to mitigate this is the coloring of the global automaton with violation information of the separate constraints [92], which is made possible by updating both the global and separate automata during execution. Nevertheless, even when using this technique, the interactions are untraceable. Similarly, the ReFlex framework [31] also offers a rule engine that only provides users with states that do not end up in a deadlock, however, a motivation for why certain activities are disabled in order to avoid a deadlock is not given.

A hidden dependency is defined as an interaction between constraints and their activities that are not made explicit as such in the model and its executable counterpart, in line with the definitions of [60] and [65]. Each constraint has specific characteristics that cause these dependencies over activities:

- Some constraints have an impact on the temporary violation aspect of the model as they can be in a non-accepting state and require an activity to resolve this temporary violation, i.e., for Declare this includes *existence*, *response*, and *choice* constraints.
- Some constraints can disable activities for the remainder of the execution, i.e., the *absence*, *not succession*, *exclusive choice*, and *not co-existence* constraints.
- Some constraints can temporarily block all other activities, i.e., all *chain* constraints.

When these characteristics are applied simultaneously over an activity, the enabledness of the activity gets influenced in many ways which requires foresight regarding the resolution of a constraint in the future. E.g., consider the model in Figure 4.1 consisting of $A = \{a, b, c\}$ and $\Pi = \{\pi_{resp}(a, b), \pi_{resp}(b, c), \pi_{exa}(c, 2)\}$. It contains a hidden dependency between *exactly(c,2)* and *response(a,b)*. When c is fired once (and hence can only fire one more time), and a has fired without b firing already, c should not fire before b resolves the temporary violation of $response(a, b)$. Since after firing c, c cannot resolve $response(b, c)$ anymore (as it can only fire two times) and b should not fire to avoid another temporary violation of $\textit{response}(b,c)$. A hidden dependency between *exactly(c,2)* and the two *response* constraints is therefore present, caused by *exactly(c,2)* over all the activities in the model.

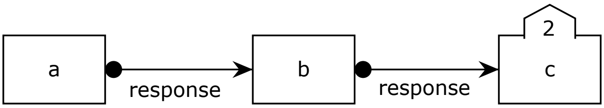


Figure 4.1: An example of a small Declare model with hidden dependencies.

In order to be able to better reflect how the constraints interact, an extra annotation layer for the model should be constructed that brings all these characteristics together. In the next section, it will be shown how *dependency structures* built on top of the model can clarify the interplay of constraints.

4.3 Dependency Structures

This section discusses how dependency structures retrieved from constraint-based models can be constructed (Section 4.3.2) and how they can aid the interpretation of the model regarding the way in which constraints interact (Section 4.3.3). The formalization heavily relies on the formal definitions of Section 3.2.2.

4.3.1 Activity Bounds

In order to establish the notion of hidden dependencies formally, the upper and lower bound on the number of occurrence of an activity during the model execution are expressed as follows. For every activity $a \in A$ and timestamp $t \in \mathbb{N}$, the following functions are defined:

- $L : (a,t) \rightarrow \mathbb{N}$ the lower bound of the amount of occurrences of an activity at time t,

- $U : (a,t) \rightarrow \mathbb{N}$ the upper bound of the amount of occurrences of an activity at time t,

- $E : (a,t) \rightarrow \{0,1\}$ a function keeping track of the enabledness of an activity at time t,

- $O : (a,t) \rightarrow \{0,1\}$ a function indicating whether an activity fired at time t, and

- $\#_A(a,t) = \sum_{i=0}^{t} O(a,i)$ the number of times an activity has occurred up until time t.

For every constraint $\pi \in \Pi$ the following function can also be defined

- $C : (\pi, t) \rightarrow \{0,1\}$, a function keeping track of the satisfaction status of the constraint at time t.

4.3.2 Construction

Hidden dependencies are caused by the implications that constraints pose on activities and the propagation of these implications through other constraints connected to that activity. To indicate these properties, the functions L and U are used: its constraints can require the activity to either fire at least a number of times still (L), or to stop firing after a limited amount of times (U). An activity a is not enabled anymore when $U(a,t) = 0$, or $U(a,t) = 0 \implies E(a,t_l) = 0, \forall t_l > t$. In Table 4.1, an overview is given for all constraints and how they propagate these bounds, as well as their impact on the enabledness on a certain time $E(a,t)$. Again, all statements formulated for, e.g., *precedence(a,b)* also hold for *(chain/alternate) precedence/succession*. In case a declarative process model is not in a permanently violated state, it holds that $U(a,t) \geq L(a,t)$, $\forall a, t$, as the lower bound is always calculated as the maximum of the old and newly assigned value, and the upperbound is the minimum of the old and newly assigned value. E and C are determined by the outcome of the operationalization of the separate constraints, e.g., an FSA, and the outcome of applying the rules in the table iteratively over all activities. $E(A, t+1|\pi(a,b))$ is calculated as in Algorithm 1 where $\pi(a,b)$ is the constraint for which the rule is applied, i.e., *chain response* or *not chain succession*. The procedure checks whether the activity is enabled and whether it will become enabled through firing the antecedent of the constraint in question which may serve as a consequent in a *precedence* relationship.

For a branched *response(a,B)* constraint, the implications on the activity bounds become:

- $\sum_{b \in B} U(b,t) = 0 \implies U(a,t) = 0$,
- $\exists b \in B$, $U(b,t) = 1 \wedge \sum_{b_o \in B \setminus b} U(b_o,t) = 0 \wedge L(a,t) > 0 \implies E(b,t+1) = 0$, and
- $(L(a,t) > 0 \vee C_A(t) = 1) \wedge \sum_{b_o \in B \setminus b} U(b_o,t) = 0 \implies L(b,t) = \max(L(b,t),1)$.

The same exercise can be repeated for other branchable constraints as well.

Constraint	**Implications for Activity Bounds**	**Motivation**
Existence(a,n)	$L(a,t) = \max(n - \#_A(a,t), L(a,t))$	The lower bound of a is the maximum of either the amount a had to fire (n) minus the amount fired, and the lower bound as imposed through other constraints.
Absence(a,n)	$U(a,t) = \min(n - 1 - \#_A(a,t), U(a,t))$	The upper bound of a is the minimum of the amount of times a can still fire, and the upper bound as imposed through other constraints.
Responded existence(a,b)/ Precedence(b,a)	$L(a,t) > 0 \wedge C(t) = 0 \implies L(b,t) = \max(L(b,t), 1)$	If a still has to fire, b has to be able to resolve the constraint, unless this has already happened.
	$U(b,t) = 0 \wedge C(t) = 0 \implies U(a,t) = 0$	If b cannot fire anymore and the constraint is not activated, a cannot inflict a constraint and becomes disabled.
Response(a,b)	$U(b,t) = 0 \implies U(a,t) = 0$	If b cannot fire anymore, a cannot fire anymore as the violation cannot be resolved anymore.
	$U(b,t) = 1 \wedge L(a,t) > 0 \implies E(b,t+1) = 0$	If b can fire only once anymore, it cannot fire until a does not have to fire anymore.
	$L(a,t) > 0 \vee C(t) = 1 \implies L(b,t) = \max(L(b,t), 1)$	If the constraint is not satisfied or a still has to fire, b has to fire at least once still.
Alt. response(a,b)	$U(a,t) = \min(U(b,t) - C(t), U(a,t))$	a can only fire as many times as all $b \in B$ can still fire.
	$L(b,t) = \max(L(a,t) + C(t), L(b,t))$	The sum of the minimum occurrence of all $b \in B$ has to be able to cover the minimum number of times a still has to occur.
	$U(b,t) < L(a,t) \implies E(b,t+1) = 0$	If a still has to fire $L(a,t)$ times, b has to be able to accommodate that number. Once its upperbound falls below that number, it becomes disabled (temporarily).
Alt. precedence(b,a)	$U(a,t) = \min(U(b,t) + C(t), U(a,t))$	a can only fire as many times as $b \in B$ can still fire.
	$L(b,t) = \max(L(a,t) - C(t), L(b,t))$	The sum of the minimum occurrence of b has to be able to cover the minimum number of times a still has to occur.
	$U(a,t) < L(b,t) \implies E(a,t+1) = 0$	If b still has to fire $L(b,t)$ times, a has to be able to accommodate that number. Once its upperbound falls below that number, it becomes disabled (temporarily).
Chain response(a,b)	$E(b,t+1 \mid \pi(a,b)) = 0 \implies E(a,t) = 0$	If b cannot fire in the next position, a cannot fire to avoid irresolvable violation.
Chain precedence(a,b)	$LB(a) = \max(LB(a), LB(b))$	The lower bound of a should always be at least the lower bound of b, because the lower bound can only be lowered when b fires immediately after a.
Not succession(a,b)	$L(b,t) > 0 \implies E(a,t+1) = 0$	If a still has to fire, it cannot fire until b does not have to fire anymore.
Not co-existence(a,b)/ Exclusive choice(a,b)	$\#_A(a,t) > 0 \vee L(a,t) > 0 \implies E(b,t+1) = 0$	If a fired or still has to fire, b cannot fire anymore.
	$\#_A(b,t) > 0 \vee L(b,t) > 0 \implies E(a,t+1) = 0$	If b fired or still has to fire, a cannot fire anymore.
Not chain succession(a,b)	$\nexists c \in A \setminus \{a\}, E(c,t+1 \mid \pi(a,b)) = 1 \wedge \exists d \in A \setminus \{a\}, L(d,t) > 0 \implies E(a,t) = 0$	If there is no activity other than a that is enabled in the next position and there exists an activity that still has to fire, a cannot be enabled.
Choice(a,b)	$C(t) = 0 \wedge U(b,t) = 0 \implies L(a,t) = \max(L(a,t), 1)$	If b cannot fire anymore, a still has to happen at least once.
	$C(t) = 0 \wedge U(a,t) = 0 \implies L(b,t) = \max(L(b,t), 1)$	If a cannot fire anymore, b still has to happen at least once.

Table 4.1: Table explaining the relation of every constraint towards the bounds of its activities.

Algorithm 1 Calculating $E(c, t+1|\pi(a,b))$.

```
Input: $c$                                                      ▷ An activity $c \in A$
Input: $\pi(a,b)$                                               ▷ A constraint $\pi \in \Pi$
Output: $\{0,1\}$
procedure $E(c,t+1|\pi(a,b))(c,\ \pi(a,b))$
    if $E(c,t) = 1$ then                                        ▷ $c$ is enabled
        for $\pi_i(a_i,b_i) \in \bullet c$ do
            if $\pi_i \in \Pi_{PD} \cup \Pi_{notchaisuc} \wedge a = a_i)$ then    ▷ Check whether $c$
                return 0                                        ▷ will not be disabled by $a$
            end if
        end for
    else
        for $\pi_i(a_i,b_i) \in \bullet c$ do
            if $\pi_i \in \Pi_{prec}$ then                      ▷ Only a precedence constraint can enable $c$
                if $C_A(\pi_i) = 0 \wedge a = a_i$ then
                    return 0                                    ▷ $c$ will not be enabled by $a$
                end if
            end if
        end for
    end if
    return 1
end procedure
```

By propagating all these dependencies whenever a change is made to an activity until all bounds are set, the dependencies can be made explicit throughout the model. To explain how the constraints relate exactly, dependency structures are constructed to link parts of the model into constraint groups. Each constraint has different implications, and from Table 4.1, the following types are derived:

– **Backward-propagating constraints:** all *response*-type constraints require that the consequent resolves the temporary violation that might be triggered by the antecedent. Hence, all constraints that have the antecedent working as a consequent, being 'on the left hand side' directly influence the consequent, for it needs to resolve any outstanding temporary violations and hence has its lower bound raised:

  $\Pi_{BW} = \Pi_{respex} \cup \Pi_{coex} \cup \Pi_{resp/suc}$

– **Forward-propagating constraints:** all *precedence*-type constraints require that the consequent fires to activate the antecedent. Hence, all constraints that require the antecedent to act as a consequent to resolve a violation rely on this type of constraints:

  $\Pi_{FW} = \Pi_{coex} \cup \Pi_{prec/suc}$

– **Permanently-disabling constraints :** *not succession(a,b)*, *not chain succession(a,b)*, and *exclusive choice(a,b)* all permanently disable either both a and b, or only b, as the cumulative function $\#_A$ is used to set the bounds. Once the activities are disabled, they cannot become enabled again. The same holds for the *absence* and hence *exactly* constraint:

$\Pi_{PD} = \Pi_{notsuc} \cup \Pi_{notcoex} \cup \Pi_{exclchoi}$

Now the set of dependency structures DP is constructed for DM with a dependency structure being a tuple $DS = (\pi^{DS}, \Pi_{dep}^{DS}, DS_{dep}^{DS})$, $DS \in DP$ with

- π^{DS} the constraint triggering the structure,
- Π_{dep}^{DS} the set of dependent constraints, and
- DS_{dep}^{DS} the set of nested dependency structures dependent of π^{DS}.

To fill Π_{dep}^{DS} and DS_{dep}^{DS}, Algorithm 2 creates a dependency structure for every activity that is involved in at least one of the five constraints that can permanently disable it. Hence, a structure is created for a in *absence/exactly(a,n)*, a and b in *exclusive choice/not co-existence(a,b)*, and for b in *not succession(a,b)* as can be seen in Algorithm 2, lines 7-25.

First, all backward-propagating constraints are considered ($\Pi_{BW} \subseteq \Pi$) and used for recursive search, as well as stored in Π_{dep} (Algorithm 3, lines 1-22) as they have a direct impact on π^{DS}. During this procedure, all incoming *existence* and *choice* constraints are stored as well (Algorithm 3, lines 16-18). They also need to be fulfilled, but do not propagate due to their unary nature. When *responded existence* is encountered, a new dependency structure $DL \in DS_{dep}^{DS}$ is constructed because when the constraint becomes satisfied (by firing its consequent), it is satisfied indefinitely (unlike, e.g., *response* which can become temporarily violated again) and its propagation is also abolished (Algorithm 3, lines 6-10).

For every activity that is encountered by the algorithm, a forward-dependency search is performed for all forward-propagating constraints $\Pi_{FW} \subseteq \Pi$, which includes all (*alternate/chain*) *precedence* constraints and *co-existence*. These constraints need to be activated (the antecedent has to be fired, in the case of alternating versions even multiple times) to resolve dependencies from backward-propagating constraints. The constraints dependent on them are linked to them through a separate, nested dependency structure $DL \in DS_{dep}^{DS}$ (Algorithm 3, lines 23-36).

Algorithm 2 Retrieving dependency structures

```
Input: $DM = (A, \Pi)$
Input: $\Pi_{BW} \leftarrow \Pi_{resp/coex} \cup \Pi_{resp}$          ▷ Backward-propagating constraints
Input: $\Pi_{FW} \leftarrow \Pi_{coex} \cup \Pi_{prec}$          ▷ Forward-propagating constraints
Output: $DP$          ▷ The set of dependency structures for $DM$
procedure ReturnDepTrans($DM, \Pi_{BW}, \Pi_{FW}$)
    $DP \leftarrow \emptyset$          ▷ The set of all dependency structures of the model
    for $\pi \in \Pi$ do
        $DS \leftarrow \emptyset$          ▷ The dependent structure for $\pi$
        $V^l \leftarrow \emptyset$          ▷ Set of visited activities for left search
        $V^r \leftarrow \emptyset$          ▷ Set of visited activities for right search
        if $\pi \in \Pi_{abs} \vee \pi \in \Pi_{exa}$ then
            $\pi^{DS} \leftarrow \pi$
            $DS \leftarrow SeaLe(\pi_a, V^l, DS) \cup SeaRi(\pi_a, V^r, DS)$
            $DP \leftarrow DS$
        end if
        if $\pi \in \Pi_{notsuc}$ then
            $\pi^{DS} \leftarrow \pi$
            $DS \leftarrow SeaLe(\pi_b, V^l, DS) \cup SeaRi(\pi_b, V^r, DS)$
            $DP \leftarrow DS$
        end if
        if $\pi \in \Pi_{exclchoi} \vee \pi \in \Pi_{notcoex}$ then
            $\pi^{DS} \leftarrow \pi$
            $DS \leftarrow SeaLe(\pi_a, V^l, DS) \cup SeaRi(\pi_a, V^r, DS)$
            $DP \leftarrow DS$
            $DS_2 \leftarrow \emptyset$
            $\pi^{DS_2} \leftarrow \pi$
            $DS_2 \leftarrow SeaLe(\pi_b, V^l, DS) \cup SeaRi(\pi_b, V^r, DS_2)$
            $DP \leftarrow DS_2$
        end if
    end for
    return $DP$
end procedure
```

4.3.3 Interpretation

First, the definition of a hidden dependency is reinterpreted in light of the previous formalization:

> *All the updates that are performed throughout the model which change the upper and lower bounds of an activity because of unary propagation, and which are caused by other activities and constraints not directly connected to that activity, are externalizations of hidden dependencies.*

By discovering and calculating all these updates, the approach of using the propagation and the dependency structures give a *complete* overview of the hidden dependencies.

While the unary propagations provide the rationale for explaining hidden dependencies, dependency structures are used to visualize them. Although constructing dependency structures can already give extra information by displaying them in a graph showing which constraints interact with the main constraint (π^{DS}) in the structure, they can also be expressed in extra de-

Algorithm 3 Searching for dependent constraints

```
procedure SeaLe($a, V, DS$)                          ▷ Search the left hand side of the activity
    if $\neg(a \in V)$ then                           ▷ Do if $a$ is not visited yet, avoids infinite loops
        $V \leftarrow a$
        for $\pi \in \bullet a$ do                     ▷ Scan all incoming Declare constraints of activity $a$
            if $\pi \in \Pi_{BW}$ then
                if $\pi \in \Pi_{respex}$ then
                    $DL \leftarrow \emptyset$          ▷ Create new nested dependency structure
                    $\pi^{DL} \leftarrow \pi$
                    $DL \leftarrow SeaLe(\pi_a, V, DL) \cup SeaRi(\pi_a, V, DL)$
                    $DS_{dep}^{DS} \leftarrow DL$      ▷ Add nested structure to main structure $DS$
                else
                    $\Pi_{dep}^{DS} \leftarrow \pi$
                    $DS \leftarrow SeaLe(\pi_a, V, DS) \cup SeaRi(\pi_a, V, DS)$
                end if
            end if
            if $\pi \in \Pi_{exis} \vee \pi \in \Pi_{exa} \vee \pi \in \Pi_{choi}$ then
                $\Pi_{dep}^{DS} \leftarrow \pi$
            end if
        end for
    end if
    return $DS$
end procedure

procedure SeaRi($a, V, DS$)                          ▷ Search the right hand side of the activity
    if $\neg(a \in V)$ then
        $V \leftarrow a$
        for $\pi \in a\bullet$ do                      ▷ Scan all outgoing Declare constraints of activity $a$
            if $\pi \in \Pi_{FW}$ then
                $DL \leftarrow \emptyset$
                $\pi^{DL} \leftarrow \pi$
                $DL \leftarrow SeaLe(\pi_b, V, DL) \cup SeaRi(\pi_b, V, DL)$
                $DS_{dep}^{DS} \leftarrow DL$
            end if
        end for
    end if
    return $DS$
end procedure
```

scriptions to annotate the model in order to help understand why constraints are related and what combined impact they have. These descriptions can be provided next to the model and are based on the following principles.

First of all, for *exclusive choice(a,b)* and *not co-existence(a,b)*, the structures reflect that whenever an activity from either structure is fired (either one in the structure containing a or b), the activities in the other structure become disabled permanently ($UB(a)/UB(b) = 0$). Indeed, firing any activity in the dependency structure of a or b requires a or b to fire, hence activating *exclusive choice* or *not co-existence*. If the structures of a and b share activities, this means the net is not deadlock-free.

Secondly, for *not succession(a,b)*, a becomes disabled whenever a constraint $\pi \in \Pi_{dep}^{DS}$ is temporarily violated and needs b to resolve it (i.e. $LB(b) > 0$). Also, dependent structures in $d \in DS_{dep}^{DS}$ cannot contain any violations in their Π^d_{dep} unless the antecedent of the main constraint $\pi^d \in DS^d_{dep}$ is activated and can execute a minimum number of times required. For unary constraints, *absence(A,n)* and *exactly(A,n)*, this applies as well, with the exception that a becomes disabled when a constraint relies upon it to become satisfied again ($UB(a) = 1$ but there exist activities in Π^{DS}_{dep} for which the lower bound is higher than 0).

Finally, every execution of activities in *chain* constraints should be checked. For each of them, it is checked whether the consequent is available to fire for *chain response*, or is the only one available for *not chain succession* in order to avoid deadlock.

4.3.4 Example

Consider the model in Figure 4.3 and its dependency structure in Figure 4.2. The lower and upper bounds and the enabledness of the activities are added for clarification. *Not succession(f,d)*, so any occurrence of f cannot be followed eventually by d, causes the algorithm to construct a dependency structure for d. Backward searching will yield no constraints, however, a forward search adds a new dependency structure for *alternate precedence(d,e)*. If d cannot fire anymore ($UB(d) = 0$ through *not succession(f,d)*), any activity that still relies on d to enable it will become permanently disabled as well, or have an upper bound of 1 when the *alternate precedence* is activated. A backward search for e adds *alternate response(b,e)* to the set of dependent constraints, then the operation continues until the following structure is constructed: $DS = \{\pi^{DS} = \pi_{notsuc}(c,b), \Pi^{DS}_{dep} = \emptyset, DS_{dep} = \{\pi^{DS} = \pi_{altprec}(d,e), \Pi^{DS}_{dep} = \{\pi_{altresp}(b,e), \pi_{altresp}(a,b)\}$, $DS_{dep} = \{\pi_{DS} = \pi_{altprec}(b,c), \Pi^{DS}_{dep} = \pi_{exis}(c,2), DS_{dep} = \emptyset\}\}\}$. The dependency structure can also be visualized as in Figure 4.2.

In this model, situations with very tricky implications can occur because of the interplay of constraints. As long as the lower bound of e does not reach 1 and *alternate precedence(d,e)* is not activated, f cannot fire. When c has fired once and a has fired, b needs to fire only once more to enable c for a second time. Hence e has to fire only once more afterwards to resolve *alternate response(b,e)*, and f can fire. If e would fire before reaching a lower bound of 1, d would have to fire in order to make e enabled again, hence disabling f. In a case where d would fire for the last time and e can fire only once more, e becomes disabled in case a fires, as this would require b to resolve *alternate response(a,b)* first and hence e to resolve *alternate response(b,e)* afterwards. Since e can only fire once more, it cannot use its last execution until all previous constraint violations have been resolved before it can resolve the violations directly connected to it.

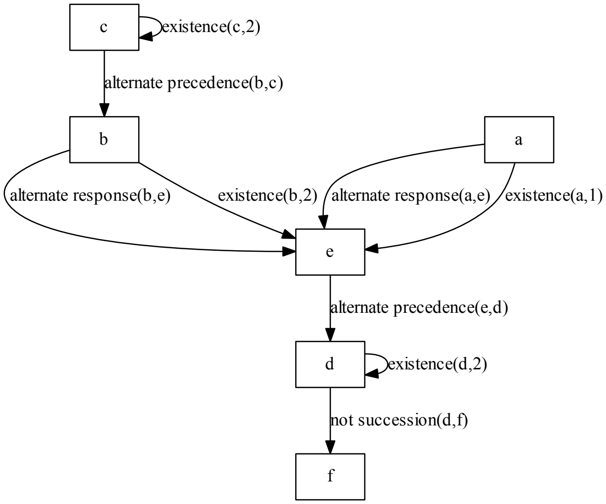


Figure 4.2: The corresponding dependency graph of the model in Figure 4.3.

4.4 Conclusion and Future Work

This chapter reported on the in-depth analysis of declarative process models which yields an approach that renders users and modelers able to discover the dependencies that exist between constraints defined over the activities in a model. The approach is based on the propagation of characteristics regarding the upper and lower bound on the occurrence of activities and their effect on the enabledness of activities. The results include an overview of all propagation properties of the constraints on a template-level, as well as algorithms to construct dependency structures, which are a tree-like visualization of all the connections that exist between the constraints.

The uncovering aims at improving the understanding of how constraints are related and to relieve users from the impediment of not being able to grasp the full behavior that the constraints impose on the activities. Therefore, in the next chapter, an overview of a user study in which the dependency structures are calculated and used for extra textual annotations is reported. Finally, the approach is also a valid tool that can be used to understand the impact of conjoining certain types of constraints for executable models such as FSAs and R/I-nets.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(a)	1	1	1	1	1	1	1	1	1	1	1	0	0	0
U(a)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	0	0	0
E(a)	1	1	1	1	1	1	1	1	1	1	1	0	0	0

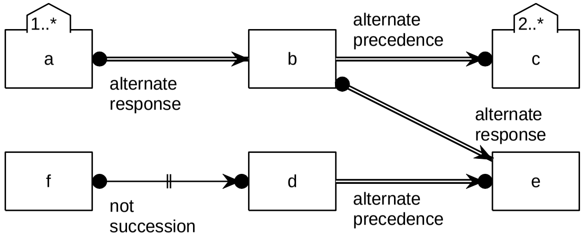
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(b)	2	1	1	1	1	1	1	1	1	1	1	1	0	0
U(b)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	1	0	0
E(b)	1	0	0	1	1	0	0	1	1	1	0	1	0	0

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(c)	2	2	2	2	1	1	1	1	0	0	0	0	0	0
U(c)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	1	1	1
E(c)	0	1	1	1	0	1	1	1	0	0	0	0	1	1

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(d)	2	2	1	1	1	1	1	1	1	0	0	0	0	0
U(d)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	0	0
E(d)	1	1	1	1	1	1	1	1	1	1	0	0	0	0

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(e)	2	2	2	1	1	1	1	1	1	1	1	1	1	0
U(e)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1	1	1	0
E(e)	0	0	1	0	0	0	1	0	0	1	0	0	1	0

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L(f)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U(f)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
E(f)	0	0	0	0	0	0	0	0	0	1	1	1	1	1



t	1	2	3	4	5	6	7	8	9	10	11	12	13
Execution	b	d	e	c	b	d	e	c	d	f	a	b	e

Figure 4.3: An example of a small Declare model with hidden dependencies with an example execution. The upper and lower bounds, as well as the enabledness are visualized per activity.

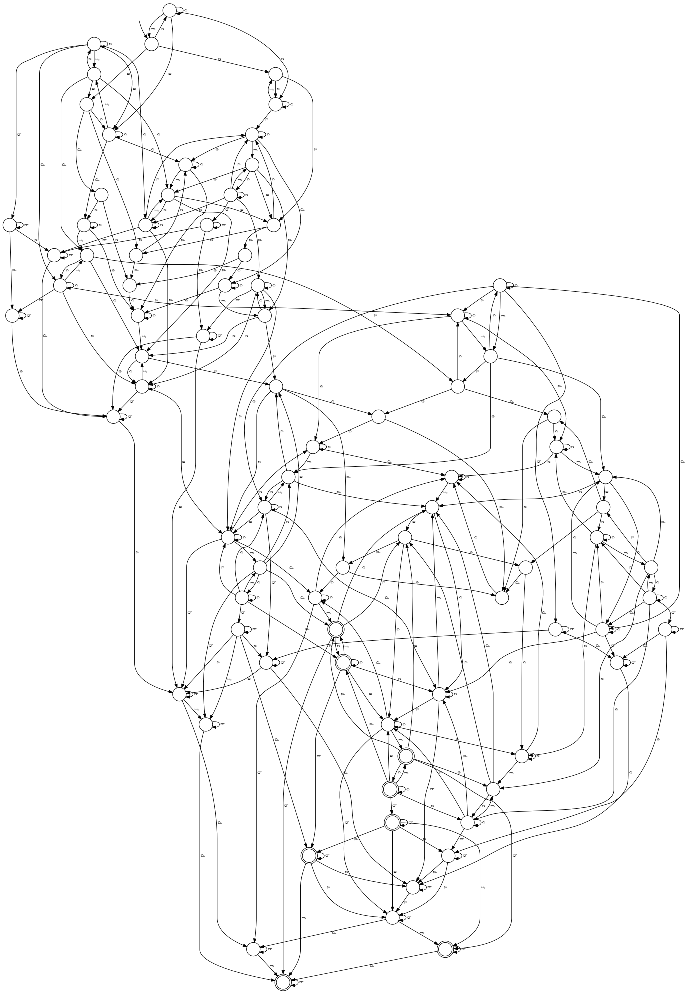


Figure 4.4: Finite state automaton of the model in Figure 4.3. The links with the constraints are completely abolished.

CHAPTER 5

Applications of Uncovering Hidden Dependencies

"Eighty percent of success is showing up."
— Woody Allen

In the previous chapter, Chapter 4, it was shown how dependencies in constraint-based declarative process models can get uncovered and represented as a series of bound propagations visualized through dependency structures. In this chapter, the applications that are enabled by this uncovering are researched in four forms.

First, it is scrutinized how the complexity of a declarative process model can be measured by making use of the dependency information of constraints used in a model. At the moment, very little research has been conducted on this subject. Even though declarative process models are often regarded as difficult to understand, no effort has been made towards quantifying the complexity of them. It is the goal to obtain a scoring mechanism for declarative process models to inform the user and modeler regarding their complexity.

Next, the implementation of the approach discussed in Chapter 4 is shown. This tool, called the *Declare Execution Environment*, was used for a user study with 193 novice process modelers that was performed to assess whether annotating a Declare model with dependency information can actually improve the understanding of the reader. Results show that, indeed, modelers are better capable of grasping the full behavior of the model because of the help that was provided through the tool.

Subsequently, an initial approach towards a different representation of declarative models is introduced, based on structuring the model according to dependency structures and their carrying activities. This is useful to offer a different angle that might better reveal the impact of certain activities on the further execution of the model.

Finally, a short digression along the connection of the dependencies to the R/I-net conversions of Chapter 3 is made by showing how the knowledge base used for explaining the bounds propagation can be used together with separate R/I fragments.

5.1 The Complexity of Constraint-Based Process Models

In this section, a complexity metric for declarative process models is proposed based on the insights provided by the dependency structures. First, the existing approaches used in traditional procedural models are discussed and evaluated for declarative models. Next, a new metric is proposed that is tailored towards the characteristics of constraint-based declarative process models.

5.1.1 Complexity

The complexity of software programs encompasses many aspects. In the classification of complexity proposed by [15], a distinction is made between psychological, computational, and representational complexity. It is argued that the former is the most important for cognitive complexity, which itself is comprised of problem complexity, programmer characteristics, and structural complexity. While the former two are either fixed (program complexity) or hard to control (programmer skills), the latter is often quantified by means of characteristics of code [66] or graph properties [101].

One of the most widely used metrics in traditional BPM is based on the cyclomatic complexity [101], which is defined for a strongly connected graph as $v(\Phi) = e - n + p$ with

- e the number of edges,
- n the number of vertices, and
- p the number of connected components, being the begin state and accepting states.

This metric reflects the number of execution paths possible in the model. It was adopted for (procedural) process models as the Control-Flow Complexity by [16, 54], which extends the cyclomatic complexity with typical routing constructs such as (X)OR splits and joins. Although it is possible to use this metric to give an estimate of how complex a declarative process model is, because v can be calculated on FSAs, it is hard to motivate the outcome as the link between execution model and constraint model is missing. Consider the example of Figure 4.3 of Section 4.3.4 on page 77. By constructing its execution automaton and applying Huffman reduction [76], a cyclomatic complexity of 243 is found. Removing *not succession(f,d)*, hence removing the hidden dependencies, reduces this number to 181. However, downsizing the hidden dependency structure by removing *alternate precedence(b,c)*, which causes an extra dependency structure nested in the main one, instead results in a score of 150. Removing *alternate response(a,b)* and thus removing part of a nested structure, results in a score of 190. Removing all *existence* constraints reduces the score drastically to 71, not surprisingly reflecting the fact that counting in FSAs is a tedious task.

For another illustration, the examples used in the experiment (for details see Section 5.2) were converted into executable finite state machines based on regular expressions and again Huffman reduction was applied. The complexity scores can be found in Table 5.1. Unfortunately, while the scores confirm the intuition that bigger models suffer from higher complexity, it is hard to explain from the FSM which factors actually influence the score. By performing a leave-one-out strategy, it is assessed which constraints drive the complexity, as can be seen in the table. However, while this might give an idea of how the constraints interact with that particular constraint, it remains to be seen how groups of constraints actually drive complexity. Also, this does not necessarily line up with the perceived complexity of the constraints. From these rolling experiments, it appears that mainly *precedence* and unary constraints blow up the automaton.

Clearly, there is a diverse impact on cyclomatic complexity. It is also questionable whether this metric is appropriate for measuring complexity in the end. First of all, the magnitude of the number is hard to interpret, especially since declarative process models are typically designed to underspecify the control flow. This results in models with a high score although the behavior is described with only a few constraints and activities. Secondly, the link between the magnitude of the model and the complexity of the behavior is not obvious. Constraints that reduce the size of the execution model, such as *absence*, still introduce complex behavior, as restricting the amount of times an activity can be executed has a big impact on the surrounding activities as well. The *existence* constraint on the other hand increases the size of the execution model significantly and is conceptually not much different and harder to understand.

	Transitions	**States**	**Accepting States**	**Cyclomatic complexity**
Model 1	**15**	**6**	**3**	**13**
Response(a,b)	9	4	3	9
Precedence(b,c)	13	5	3	12
Not co-existence(b,e)	26	8	2	21
Response(d,e)	11	5	3	10
Model 2	**65**	**25**	**1**	**42**
Exactly(a,2)	84	30	2	57
Existence(c,2)	70	25	2	48
Exactly(b,2)	75	25	3	54
Absence(d,3)	65	25	1	42
Alternate precedence(a,c)	141	45	1	98
Alternate response(b,d)	141	45	3	100
Model 3	**198**	**44**	**3**	**158**
Response(a,b)	106	28	3	82
Response(b,c)	258	54	3	78
Exactly(c,2)	118	25	3	97
Precedence(b,e)	164	35	2	132
Response(d,e)	101	27	3	78
Alternate precedence(e,g)	148	34	2	117
Response(f,g)	106	27	3	83
Model 4	**60**	**18**	**4**	**47**
Response(a,b)	37	13	4	29
Existence(b,1)	60	18	4	47
Alternate precedence(b,c)	36	12	2	27
Not succession(e,b)	45	13	2	35
Existence(c,1)	60	18	4	47
Response(d,c)	101	30	4	76
Existence(d,2)	32	10	4	27
Model 5	**800**	**144**	**6**	**663**
Response(a,b)	616	112	6	511
Response(b,c)	979	168	6	818
Exactly(c,2)	501	84	6	424
Precedence(b,e)	640	112	4	533
Response(d,e)	425	88	6	344
Alternate precedence(e,g)	614	112	4	507
Response(f,g)	441	88	6	360
Choice(a,j)	440	88	6	359
Not succession(i,j)	380	72	3	312

Table 5.1: Comparison of the cyclomatic complexity scores for the different models of Section 5.2, with and without each constraint. The hidden dependency-inducing constraints are colored blue, while the complexity scores that are higher than the baseline model are indicated in red.

5.1.2 A Metric for Complexity of Declarative Process Models

In this section, an appropriate complexity metric for constraint-based declarative process models is constructed, based on various metrics that are available in literature.

In many works, the importance of the interconnectedness of activities is put forward as one of the most important contributors that raise complexity [136, 52, 53], which also holds for declarative models as illustrated by the user-study in [64]. The discourse in Section 4.3 explains how constraints exhibit coupling behavior, which can be derived from a model by using dependency structures of a hierarchical nature. This corresponds with the *chunking* and *tracing* landscape graphs as proposed in [15] to express processing different levels of a varying number of steps in a computer program, which was later adapted for process models by applying it to process structure trees [172] as used in [52]. Hence, the concept of interconnectivity is used as a central notion for the metric.

The declarative process model complexity metric DC for a model DM is proposed as follows.

$$DC(DM) = \Sigma_{a \in A}(| \bullet a| + |a \bullet |) + \Sigma_{ds \in DS} S(ds, 1)$$

$$S(DS, l) = 1 + |\Pi^{ds}_{dep}| + \Sigma_{ds_{dep} \in DS^{DS}_{dep}} l \times S(ds_{dep}, l + 1)$$

It tackles the relationships of the constraints in various ways. First of all, all constraints are counted and binary constraints are counted twice ($\Sigma_{a \in A}(| \bullet a| + |a \bullet |)$, for every activity they involve). This lies in line with the fan-in, fan-out principle of [67], which is designed to measure the connection of an activity to its environment. Next, all the dependency structures are scored $\Sigma_{ds \in DS} S(ds, 1)$. For each structure, the sum is made over the main constraint (1), the number of dependent constraints ($|\Pi^{ds}_{dep}|$), and the scores of the nested structures ($\Sigma_{ds_{dep} \in DS^{DS}_{dep}} l \times S(ds_{dep}, l + 1)$). The deeper a structure is nested, the higher the score to reflect the fact that the dependency is very long and hence hard to grasp. Activities that are not connected to any other activity are not taken into account, as they do have no influence on the behavior through propagation and they are only directly affected by constraints that have a global impact on the activities such as the ones including the LTL *next*-operator, *init*, and *last* constraints. This contrasts with the activity complexity defined in [17], where the amount of activities is seen as a process model equivalent to the lines-of-code metrics [84]. The proposed metric here provides a *constraint-centric* approach.

Nevertheless, in this way all the relations of the model are summed into one number that serves as a complexity metric. For the example from Section

4.3.4, this would result in a score of 33. If the same exercise of leaving out constraints is repeated, the following scores are outputted:

- $DC(DM) = 33$
- $DC(DM(A, \Pi \setminus \pi_{notsuc}(f, d))) = 10$
- $DC(DM(A, \Pi \setminus \pi_{altprec}(b, c))) = 19$
- $DC(DM(A, \Pi \setminus \pi_{altres}(a, b))) = 27$
- $DC(DM(A, \Pi \setminus \{\pi_{exis}(a, 1), \pi_{exis}(c, 2)\})) = 23$

These scores are better capable to compare the impact of the interaction of the constraints through propagation, compared to, e.g., cyclomatic complexity because they reflect the size of the model, as well as the distance and interconnectedness of the constraints.

5.2 Empirical Evaluation Through User Study

In this section, it is empirically investigated whether the cognitive effort that hidden dependencies [60] cause in declarative process models can be relieved by making them explicit. Hence, the hypothesis to be rejected is *'Annotating declarative process models with information about hidden dependencies does not yield better understandability'*. The level of understanding is measured by the score that participants received for answering questions regarding the models they had to interpret.

5.2.1 Tool Support

The construction of the dependency structures has been implemented in the Declare Execution Environment, of which the implementation can be found at `http://processmining.be/declareexecutionenvironment`. The tool can read a Declare model saved from Declare Designer [183], which, during execution, is supported by descriptions for the hidden dependencies. A screenshot and an example can be found in Figure 5.1.

Furthermore, the dependency structures can be visualized next to the model as a directed graph as well. Finally, the trace created over the model by the user is displayed below the model, aiding the user in understanding the history of the current situation displayed over the model.

The execution semantics are provided by dk.brics.automaton [106] and consist of the product of the separate Declare automata expressed in regular expressions, as can be found in [22], [185], and Table 3.1.

Finally, the tool is capable of calculating the declarative process model complexity measure of Section 5.1.

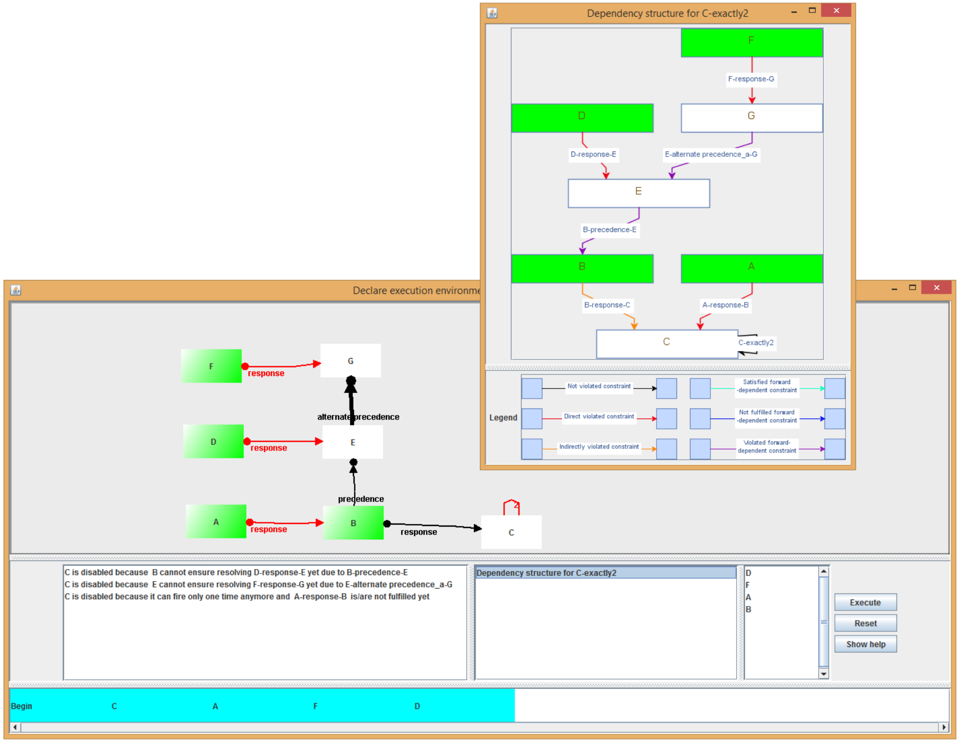
5.2.2 Experimental Setup

In the experiment, 193 students (see Table 5.3) enrolled in KU Leuven's *Business Analysis* course, in which they learn about both procedural and declarative process modeling, were asked to solve five questions for each of five different Declare models in a timespan of two hours. The students had the same modeling experience and background and can be considered novice business process modelers. The models are summarized in Table 5.2. Although not the main goal of the experiment, the models were constructed to test for different complexity scores (DC(model1)= 13, DC(model2)= 15, DC(model3)= 33, DC(model4)= 22, DC(model5)=40). They were chosen in order to have a good representation of the different constraints that can cause hidden dependencies, and their interaction.

- **Model 1:** focuses on the impact of the *not co-existence* constraint. The model contains a *not co-existence* constraint which disables a, b, and c after firing d or e. Because e disables b, c cannot fire anymore and a should not fire in order to prevent an unresolvable activation of *response(a,n)*. If d fires, e has to fire eventually and has the same effect. The other way around, firing a or b has the same effect on d and e.

- **Model 2:** focuses on the impact of unary constraint propagation. a and c are connected by an *alternate precedence* constraint and can happen exactly twice and at least twice respectively. Since for every occurrence of c a new occurrence of a is needed and a can only fire twice, c can also only happen exactly twice. Therefore, if a occurs, it has to wait for c to occur, as otherwise it has fired its maximum amount of times and cannot guarantee fulfilling the *existence* constraint of c. The same goes for b and d, connected by *alternate response*. Here, d can happen at most twice and b has to happen exactly twice. b can only happen again after the next d, hence d has to be able to occur at least as many times as b still has to happen.

- **Model 3:** focuses on the impact of simple forward and backward dependencies induced by *exactly(c,2)*. In the states where c can happen only once more, all dependencies it has with its dependent constraints need to be resolved: it has to be able to fire after activating *response(a,b)* and *response(b,c)*. Also, if f fires, *response(f,g)* needs to be resolved. This happens by firing g, which is only possible after activating constraints *alternate precedence(g,e)* and *precedence(b,e)*. If


Dependency structure for C-exactly2
F
F-response-G
D
G
D-response-E
E-alternate precedence_a-G
E
B-precedence-E
B
A
B-response-C
A-response-B
C
C-exactly2
Legend
Not violated constraint
Direct violated constraint
Indirectly violated constraint
Satisfied forward -dependent constraint
Not fulfilled forward -dependent constraint
Violated forward-dependent constraint
Declare execution environme
F
response
G
alternate precedence
D
response
E
precedence
A
response
B
response
C
2
Dependency structure for C-exactly2
D
F
A
B
Execute
Reset
Show help
Begin
C
A
F
D

C is disabled because B cannot ensure resolving D-response-E yet due to B-precedence-E
C is disabled because E cannot ensure resolving F-response-G yet due to E-alternate precedence_a-G
C is disabled because it can fire only one time anymore and A-response-B is/are not fulfilled yet

Figure 5.1: An example of a small Declare model with hidden dependencies and the corresponding dependency graph for *Exactly(c,2)*.

c fires twice before b has fired, it can never happen anymore as firing b leads to activating *response(b,c)*, which cannot be resolved by c anymore.

- **Model 4:** focuses on the impact of more advanced forward and backward dependencies induced by *not succession(b,e)*. b has to be able to activate *alternate precedence(b,c)*, so c can resolve the inevitable activation of *response(d,c)*, because d has to fire at least 2 times. Firing c before resolving *existence(d),2* will disable e, as b has to fire again to activate the *alternate(b,c)* constraint. Once e is fired (and c can fire), c becomes disabled as it can fire only one time more, until *existence(d,2)* is resolved and it can resolve *response(d,c)*. After this, both c and d become disabled.

- **Model 5:** focuses on the same impact as models 3 and 4, with extra *choice* and *not succession* constraints. This model is an extension to model 3, although containing two dependency structures. In this example, the extra *choice* constraint requires either a or j to fire. If i has fired and j has never fired and cannot fire anymore (due to *not succession(i,j)*), a has to be able to fire. If c can only fire once more, it becomes disabled until a fires and *response(a,n)* and *response(b,c)* can be resolved. If a is disabled after firing c twice (rendering it disabled due to the *response* constraints), i becomes disabled until j resolves *choice(j,a)*.

The exercises intentionally included overlaps to see whether participants progressed and learned to recognize the hidden dependencies when viewed from different angles. E.g., in model 5, all the different types of dependencies were brought together (in separate questions).

At the start of the test, students were provided instructions by making use of the example in Section 4.2, a model which was used as a foundation for models 3-5, but without the additional constraints and activities added. As such, the idea behind hidden dependencies was explained but not introduced explicitly as the subject of study. Furthermore, the introduction served as a tutorial for the participants to use the tool they were provided with.

The time participants needed to solve the questions was recorded. It was checked whether the questions that were answered in less than 20 seconds were incorporated into the final results, however, this was not the case. The students all correctly indicated that they ran out of time. There were participants that occasionally answered in less than 20 seconds based on previous answers, however, often this led to an incorrect answer as the difference often remained subtle. The students were urged to try to answer as correctly as possible, rather than to rush towards the end to fill in all questions. This

Model 1	**Model 2**	**Model 3**

Response(a,b)
Precedence(b,c)
Not co-existence(b,e)
Response(d,e)
	Exactly(a,2)
	Existence(c,2)
	Exactly(b,2)
	Absence(d,3)
	Alternate precedence(a,c)
	Alternate response(b,d)
		Response(a,b)
		Response(b,c)
		Exactly(c,2)
		Precedence(b,e)
		Response(d,e)
		Alternate precedence(e,g)
		Response(f,g)

Model 4	**Model 5**	

Response(a,b)
Existence(b,1)
Alternate precedence(b,c)
Not succession(b,e)
Existence(c,1)
Response(d,c)
Existence(d,2)
	Response(a,b)
	Response(b,c)
	Exactly(c,2)
	Precedence(b,e)
	Response(d,e)
	Alternate precedence(e,g)
	Response(f,g)
		Choice(a,j)
		Not succession(i,j)

Table 5.2: The different Declare models used during the experiments. The figures of the models used in the experiment can be found by following the link to the tool site.

meant that the questions of model 4 were left unanswered in 2 cases, the questions of model 5 in 19 cases.

In order to measure the impact of handing natural language descriptions and the visualization of dependency graphs, the students were divided into three groups which received a different version of the Declare Execution Environment. Group A could only see the Declare model and the constraint descriptions, but no color annotation nor dependency structure visualizations. Group B received a tool in which the enabled activities were colored green, and temporarily violated constraints were colored red, in a fashion described in [92] and similar to Declare Designer [183]. Also, the constraint descriptions were given. Finally, group C was given an environment with the same functionality as group B, but with extra descriptions concerning hidden dependencies, as well as the possibility to open a dynamic visualization of the dependency structures. These groups are represented in the results by the *session* variable.

The questions were aimed at uncovering to which extent the participants grasped the full impact of the blend of different constraints. They were asked to indicate which activities were enabled after firing a certain sequence, and why or how to reach a certain firing sequence. Since two out of three groups knew which ones were enabled, they could focus more on the second part of the question. An example question used for model 1 is ‘After firing d, which activities are still enabled? Explain.’. A full list of questions can be found in

		Gender		**Program**		
Group	**Participants**	**Male**	**Female**	**IS**	**Business**	**Others**
A	65	45	20	6	57	2
B	64	37	27	6	57	1
C	64	37	27	11	52	1

Table 5.3: The students were selected from 3 different study programs, however, it was made sure their distribution could not skew the results.

Appendix A.

Each question was scored on a 0 to 1 scale, where incomplete answers (usually because of overlooked hidden dependencies or incorrect use of constraints) were still awarded a score higher than 0. E.g., a student from group B who provides the correct set of enabled activities but fails to state that activity *c* in model 3 is not enabled because of hidden dependencies was still awarded 0.6. The explanation was taken into account so as to make a fair comparison with students in group A, who got no extra information, and therefore many times missed even these basic answers. Group C students that just copied extra descriptions provided by the tool also did not receive a grade of 1, as they did not prove to understand the model.

5.2.3 Results

In this section, the results are discussed for both the statistical analysis, as well as the insights gathered from the text-based answers students gave to the questions.

Quantitative Results

Given this setup, an experimental analysis can be conducted to investigate the impact of the environment students were given (i.e. session) on the score with a higher score indicating a better level of understanding. Figure 5.2 shows boxplots of the average scores over 5 questions, per model and per session. From the figure, it can be seen that for each model, an increase is observed in terms of the score when students are provided with additional annotations. Note that the data is available on the tool's web site.

The scores are skewed to the right, as it was easier to obtain a grade above 0.5 when answering correctly, and significantly more people passed than failed. The Shapiro-Wilk normality test [132] rejects that the score follows a normal distribution with a p-value smaller than 0.0001. Hence, for statistical analysis, non-parametric tests need to be used. In this case, the aligned-rank transformation approach for non-parametric ANOVA [186] was chosen for it provides a good alternative to the Kruskal-Wallis and Friedman tests as it can process interaction effects as well. Furthermore, the response variable is continuous and cannot make use of logistic regression without binning. The results can be found in Table 5.5.

Next to the session and model variables, it was tested whether program and gender had any explanatory power by performing an ANOVA analysis for $score = session + model + gender$ and all interaction effects as well as $score = session + model + program$. In both cases, the main effects with the variables were not significant on a p-threshold of 0.05, as can be seen in Table 5.5. However, some interaction effects were significant. Judging from a full regression performed on all variables and their interactions on the whole dataset, however, the effect sizes can be considered very small.

In Table 5.4, a pairwise mean comparison is done for the sessions and models. Clearly, there is a big difference between the three sessions, *offering proof to reject the hypothesis that tooling has no impact on the way the participants score for the questions.* When comparing the models, the results also show very strong differences in the estimated values, except for the difference between models 2-3 and 3-4 on a 0.05 significance level. Although the complexity scores for these models differ, the impact of the order of questions might have had an impact on the score. Furthermore, models of different sizes and with and without hidden dependencies need to be investigated to truly capture the validity of the complexity measure.

	estimate	**Std. Err.**	**df**	**t.ratio**	**p.value**
A-B	-722.729	126.1137	189.99	-5.731	<.0001
A-C	-1140.8	126.0914	189.86	-9.047	<.0001
B-C	-418.07	126.6632	190.34	-3.301	0.0033
1-2	701.1825	51.33051	4494.3	13.66	<.0001
1-3	588.4953	51.41334	4494.94	11.446	<.0001
1-4	490.6006	51.60979	4497.52	9.506	<.0001
1-5	866.8739	53.05569	4508.16	16.339	<.0001
2-3	-112.687	51.23619	4493.66	-2.199	0.18
2-4	-210.582	51.43256	4496.24	-4.094	0.0004
2-5	165.6915	52.87533	4506.67	3.134	0.015
3-4	-97.8947	51.51686	4496.96	-1.9	0.3172
3-5	278.3787	52.95978	4507.42	5.256	<.0001
4-5	376.2734	53.03212	4503.69	7.095	<.0001

Table 5.4: Comparison of the means of the different estimates, first for the three sessions, next for the models.

To evaluate the impact of each variable, a linear regression ($Score = \alpha \times model + \beta \times session + \gamma \times session \times model + \epsilon$) was fitted on the data.

	F	**Df**	**Df.res**	**Pr(>F)**
session	41.959	2	190.06	8.07E-16
model	78.5	4	4499.81	<2.22E-16
session:model	17.728	8	4499.78	<2.22E-16
session	7.4034	2	268.41	0.000742
model	7.0401	4	4475.07	1.20E-05
program	2.6998	2	243.96	0.069224
session:model	2.5618	8	4479.42	0.008726
session:program	2.0129	4	226.39	0.093563
model:program	3.9536	8	4479.96	0.000113
session:model: program	1.6487	15	4481.58	0.054136
session	37.2567	2	187.21	2.39E-14
model	69.36555	4	4488.33	<2.22E-16
gender	2.90195	1	187.2	0.090132
session:model	16.47918	8	4488.39	<2.22E-16
session:gender	0.64492	2	187.22	0.525869
model:gender	3.5061	4	4488.37	0.007277
session:model: gender	2.08561	8	4488.41	0.033791

Table 5.5: Three non-parametric ANOVA analyses with interaction effects. First including program, next including gender, and finally only between session and model.

Coefficients	**Estimate**	**Std. Error**	**t value**	**Pr(> |t|)**	
(Intercept)	0.64357	0.0164	39.237	<2.00E-16	***
model2	-0.20942	0.02309	-9.07	<2.00E-16	***
model3	-0.17773	0.02309	-7.698	1.68E-14	***
model4	-0.14204	0.02309	-6.152	8.30E-10	***
model5	-0.30587	0.02346	-13.037	<2.00E-16	***
sessionB	0.11611	0.02327	4.99	6.27E-07	***
sessionC	0.14768	0.02318	6.371	2.05E-10	***
model2:sessionB	0.02755	0.03277	0.841	0.400526	
model3:sessionB	0.06929	0.03277	2.115	0.034506	*
model4:sessionB	0.05505	0.03283	1.677	0.093664	.
model5:sessionB	0.19091	0.03369	5.667	1.54E-08	***
model2:sessionC	0.18004	0.0327	5.505	3.88E-08	***
model3:sessionC	0.11822	0.03277	3.608	0.000312	***
model4:sessionC	0.07797	0.03285	2.374	0.017655	*
model5:sessionC	0.22634	0.03338	6.78	1.35E-11	***
Residual standard error: 0.293 on 4683 degrees of freedom **Multiple R-squared:** 0.1688 **Adjusted R-squared:** 0.1663 **F-statistic:** 67.92 on 14 and 4683 DF, **p-value:** < 2.2e-16					

Table 5.6: Linear regression model based on the data gathered from the experiment with significance scores '***' 0, '**' 0.001, and '*' 0.01.

From the results in Table 5.6, *it is clear that the impact of both the model as well as the session (and hence tool) is highly significant.* Post-tests to check the assumptions for linear regression included running a Durbin-Watson-test [45] which rejected the hypothesis for correlation among the residuals. Finally, it was tested whether the error terms were distributed normally, as can be seen in Figure 5.3. Observe that the data was also fitted for models with *gender* and *program* included. These models did not raise the R-squared values much (<0.18), hence hinting at little extra explanation power. Again, the same results can be observed. The results suggest that the proportion of the model for which the hidden dependencies hold affects the complexity, or at least the intricacies of counting the lower and upper bound, because the scores for model 2 were significantly lower for all groups, where group C enjoyed the biggest surge in terms of scoring.

Nevertheless, the participants of sessions 2 and 3 performed significantly better than the ones in session 1. Session 3 students performed best, although the gap is somewhat smaller. Hence, *it can be concluded that annotating Declare models with extra dependency information improves the ability of model users to correctly interpret the model.*

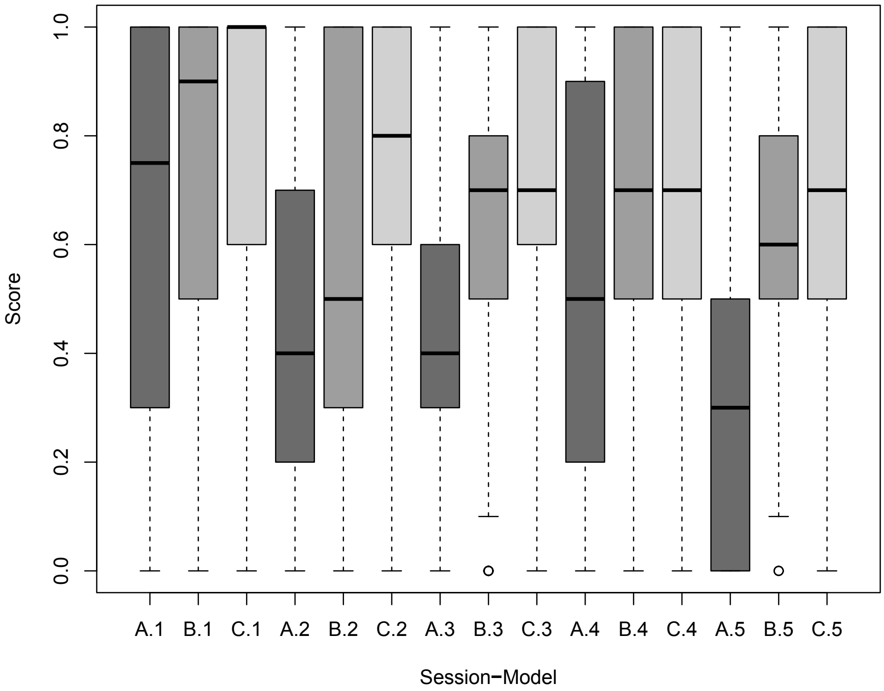


Figure 5.2: Boxplot of the average scores of 5 questions per model (1-5) and per session (A-C).

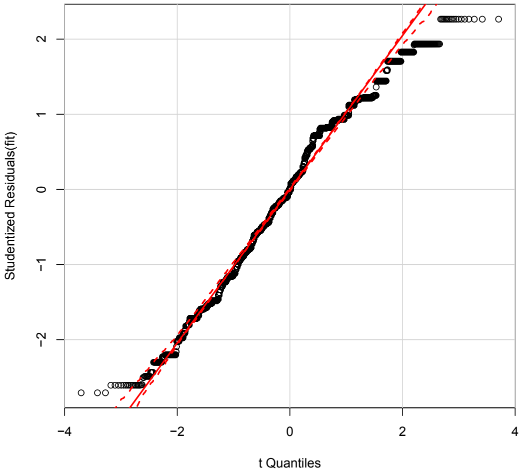


Figure 5.3: Q-Q plot of the error terms showing they are close to a normal distribution.

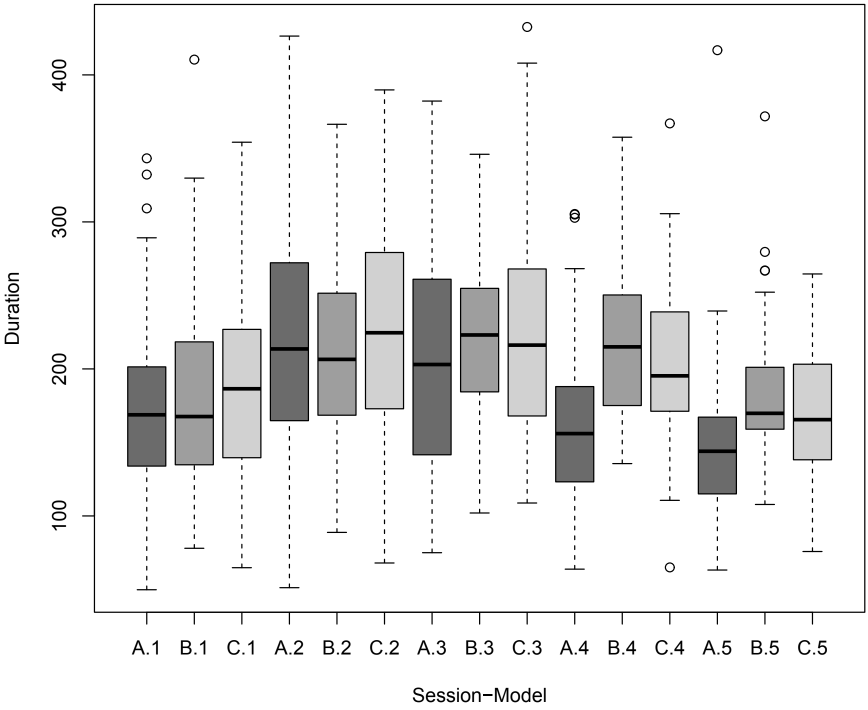


Figure 5.4: Time (in seconds) needed to answer the 5 questions for the models (1-5) per session (A-C).

Qualitative Results

Since the participants did not just give an answer in the simple form of 'a is now enabled' but had to motivate their answers, some extra observations could be made concerning the results. Although it was the case that the two groups with the more elaborate tool were better capable of seeing which activities were enabled and which constraints were violated, they still seemed to ignore these annotations. Especially group B sometimes ignored the coloring of the model as they did not understand some implications of the constraints. Participants often also bended the descriptions of the Declare constraints towards their understanding, hence starting to discuss irrelevant parts of the model. For the third group, this behavior was still present, although to a much lesser extent. Group A participants often found the hidden dependencies in the easier examples. Because they had no support they thought harder about the model, but nevertheless failed to find any (hidden) dependencies in the more elaborate examples. Furthermore, it was obvious from results that any type of *response* constraint was very hard to grasp for students in general. Especially the constraint description that was adopted from Declare tool [183] ('Whenever *a* happens, there needs to be a *b*...') seemed to cause confusion when used in a different context than in class. Some students started to interchange *response* and *alternate response* because of the "Whenever *A*..." part.

The effects of the duration were not significant for a particular combination of session and model, however, for the last two models there seems to be a difference in the amount of time needed to solve the questions as can be seen from Figure 5.4. Students from the first session spent less time on answering them, indicating either a lack of motivation, or just a clear lack of understanding leading them to believe that the questions had the same answers so they could rush towards the end. Since the models at first do not seem to be much more advanced than the previous examples, students missed the interplay of constraints and regarded them as non-existing.

Remarks

As in all empirical experiments, there are threats to validity that need to be addressed, the main ones in this case are:

- **Construct validity:** The experiment was threatened by the hypothesis guessing threat because students might figure out what the purpose of the study is, which could affect their guesses. We minimized this threat by hiding the goal of the experiment. Furthermore, it was only possible to test 5 models, which does not cover all the different dependencies. However, all the constraints that cause hidden dependencies were used in the questions to cover them at least once. To test the complexity measure, however, too few different models of different sizes were tested to make a fixed conclusion.

- **Internal validity:** The experiment had the maturation threat because subjects may react differently as time passes (because of boredom or fatigue). We solved this threat by dividing the experiment into different questions per model. Also, it was made sure there could be no interaction between the students of different sessions.

  The questions were always asked in the same order, mainly due to technical restrictions, and also to capture whether the participants progressed and learned from the previous examples. Not only over exercises, but within an exercise series the questions built upon each other to some extent. However, students received the base concept of how they needed to interpret a model with hidden dependencies during the introduction, giving them a clue for models 1 and 3-5.

- **External validity:** The experiment might suffer from interaction of selection and treatment: the subject population is limited to students. Although the number of subjects is quite high and their profiles quite balanced, the results can only be generalized to students, but the subjects might not be representative to generalize the results to professional modelers as well. It is, e.g., not possible to claim that the tool can help or improve Declare modeling efforts for more experienced users.

5.3 Refactoring Declarative Process Models

Hidden dependencies in process models are often built into the model because of a lack of experience. Modelers are not aware of all the consequences that the semantics of different constraints have, especially not when viewed in a bigger picture. Next to uncovering the dependencies in order to raise understanding of the interplay of constraints or reporting on the complexity of a model as done in the previous sections, another possible solution is to restructure a model in order to take away the dependencies, or to delineate them in a fixed part of the model. In this section, an initial approach to do so is devised by means of two examples.

5.3.1 Staged Declarative Models

Hidden dependencies that are caused by the constraints listed in Section 4.2 all have a common denominator in the fact that they only take effect after the occurrence of a certain activity, as detailed in Section 4.3. For *not*

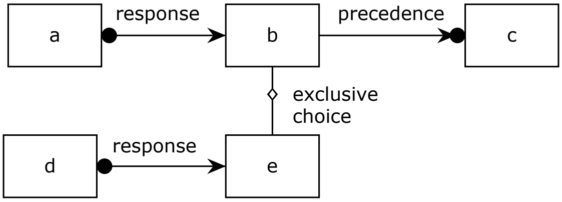


Figure 5.5: A small Declare model centered around an *exclusive choice* constraints.

succession(a,b) the dependency holds before *a* fires, for *exclusive choice(a,b)* and *not co-existence(a,b)* the dependency holds before either activity and their dependent activities are fired, and for *absence(a,n)*, the dependency holds while *a* has not fired $n-1$ times yet. Hence, in the models in which the dependencies are present, different stages exist that can be used to re-engineer the way the model is represented.

5.3.2 Example 1

Consider for example the small model in Figure 5.5. It contains an *exclusive choice* constraint which incurs the behavior that is detailed in Section 5.2 under Model 1. When *a* or *b* fire, *d* and *e* cannot fire anymore, while when *d* or *e* fire, *a* and *b* cannot fire anymore, and *c* will never be able to fire. Hence, the model is completely separated into two distinct parts that can never occur together. In Figure 5.6, the two separate models that exist next to each other are displayed. In the initial state, depicted as a green circle, the first activity is chosen. Depending on this choice, one of the two models that reside in the global model is activated. When no constraints are in a temporarily violated state, the model can end in a final, accepting state, depicted as the red circle. Here, the model is two-staged, as the first stage determines in which model the overall model resorts to after firing the first activity.

For *absence*, a similar procedure can be followed. The difference is that the model is not necessarily split up in the first stage, but rather that the split can occur at any given time in the process. This also implies that the intermediate constraint violation statuses influence which stage can be obtained or not.

5.3.3 Example 2

Consider the model in Figure 5.7. After *g* fires twice, *e* and *f* are disabled permanently as well, and the other activities' enabledness depends on the previous occurrence of *f*, as explained in Section 5.2 under Model 3. The

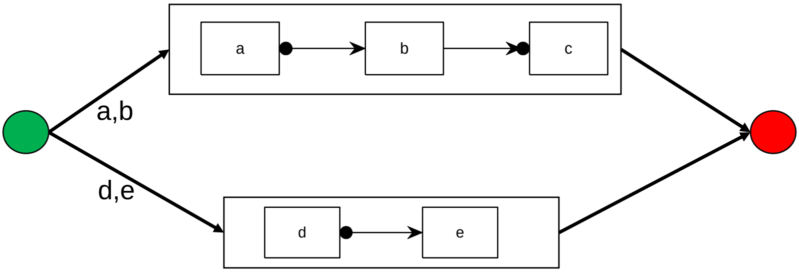


Figure 5.6: An alternative view on the model displayed in Figure 5.5.

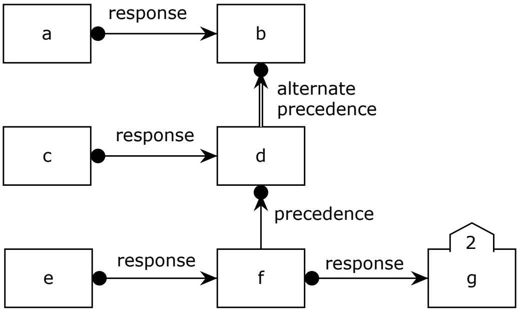


Figure 5.7: A small Declare model centered around an *absence* constraints.

first stage exists up until g fires the first time. Then, depending on which activities fire in the meantime, the next stages can be reached, but first all temporarily violated constraints that still exist in the e-f-g chain need to be resolved. Then, according to whether f has fired or not, the upper part of the model consisting of a, b, c, and d is still enabled.

Figure 5.8 displays the different stages of the model. g is considered in isolation, since only two occurrences of g occur. In Stage 1, if f or an activity from inside the dependency structure $DS_{precedence(d,f)}$ occurs followed by f, the model sheds the dependency structure. The same holds for Stage 3. When g occurs in Stage 1, Stage 2 is reached. Both Stage 2 and 3 share that the *response* constraint between e and f needs to be in an accepting state before g can fire for the final time. If the dependency structure has become independent of f, the four activities remain enabled, otherwise, no other activity is enabled and the model transitions to its final state.

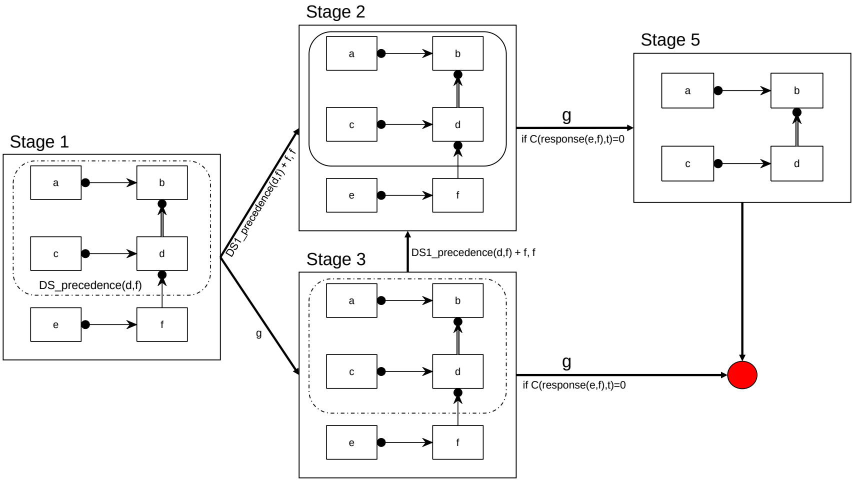


Figure 5.8: An alternative view on the model displayed in Figure 5.7.

5.4 Global Semantics

In previous discourses on the nature of the global semantics of constraint-based models, it was shown that constraints could not be executed separately without conjunction to avoid deadlocks and unresolved violations. However, now that all the interdependencies between constraints have been formalized and can be calculated on-the-fly, it is possible to steer the separate automata by the knowledge base of activity bound propagations that was discussed in Section 4.3.

This is especially useful for combining separate constraints without making the intersection of the behavior. The constraints are instantiated but provided with lower and upper bounds which illustrates how they interact at a given moment. Consider the example in Figure 3.22 on page 62, in a stripped down version modeled as a Colored Petri net which is behaviorally equivalent and augmented with a knowledge base transition, in Figure 5.9. The knowledge base should ideally be connected with every activity and collect information on all the states of the constraints (which are stored in the places), to propagate the correct upper and lower bounds to the activities. More precisely, the lower and upper bound can be replaced by a color set of a single integer, which can enable or disable an activity temporarily.

However, this approach cannot replace model checking in general, as the knowledge base can only avoid inconsistencies on the go. Hence, a full analysis is only reported after running a full state space exploration, unless an error has occurred sooner. Nevertheless, the information of upper and

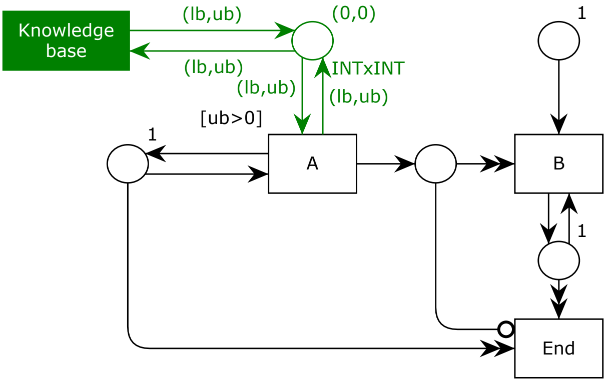


Figure 5.9: The example of Figure 3.22, now connected to a knowledge base which is able to provide the correct upper and lower bounds for the activities.

lower bounds can also be integrated with the bounds information of a Petri net in a mixed-paradigm model. By performing state space analysis, the bounds of transitions can be calculated. Combining the information of both analysis techniques can be used towards finding inconsistencies between both models, and can also intersect the behavior during execution.

Consider for example the mixed-paradigm model in Figure 5.10 to achieve a final marking where a single token is delivered to p_5. The lower bound of a, b, c, and d is 1. The upper bound of a and d is 1 as well, while the other activities are unbounded upwards. On the other hand, the Declare model has a restriction where f and g cannot co-occur, hence b and c cannot co-occur. Clearly, the lower bounds are in conflict as when running the model, either the lower bound of b or c is not met. In Chapter 8, a full mixed-paradigm model checking approach is proposed. However, using the bounds provides activity- and constraint-granular feedback during execution can, according to the type of model checking that is required, be preferred over the constraint-granular feedback that is provided by conjoining automata.

5.5 Conclusion and Future Work

This Chapter reported on the implementation and applications of uncovering dependencies between constraints in declarative process models. The first complexity metric tailored towards declarative process models was introduced by making use of the interdependency of constraints, rather than of control flow complexity, and was inspired by the current body of research on the cognitive effort required to interpret (declarative process) models.

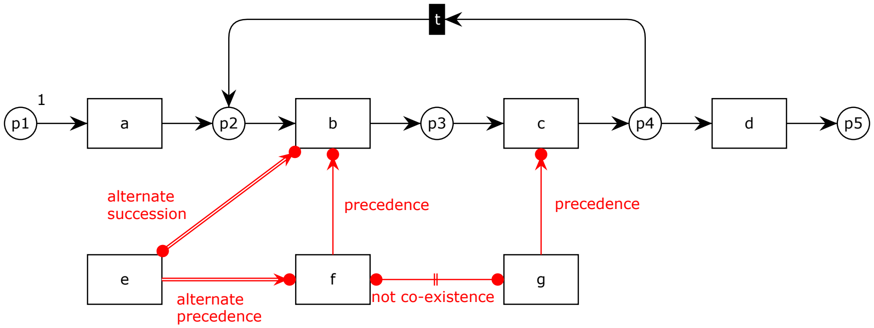


Figure 5.10: An example of a mixed-paradigm model that has conflicting behavior between both parts of the model.

The presented user-study with novice process modelers was carried out to investigate whether the effort of understanding a declarative model was lowered in case of hidden dependencies by uncovering them. Furthermore it was tested whether the scores combined with the complexity metric lie in line with the intuition used for the metric. Results suggest that the understandability was raised significantly by showing the information regarding the interplay of events, however, for determining a complexity measure further research needs to be performed to fine-tune the elements used for calculation.

Next, the refactoring of declarative process models was briefly touched upon, showing how constraints inflicting hidden dependencies have an impact on the global behavior by displaying them in a staged-like fashion. This approach can aid future modelers in constructing models and avoiding too convoluted behavior.

Finally, the usefulness of calculating the bounds on the occurrence of an activity was used in a model checking context, and the connection with Chapter 3 was made to illustrate how a global semantics can be established without making a full intersection of the different constraints that are used to construct the model.

Part III

Modeling and Enacting Mixed-Paradigm Process Models

CHAPTER 6

Mixed-Paradigm Process Modeling with Intertwined State Spaces

"They love to tell you
Stay inside the lines
But something's better
On the other side."
— John Mayer, No Such Thing

This part of the thesis deals with the comparison and simultaneous use of the declarative and procedural process (modeling) paradigms. The three chapters deal with the modeling, mining, and verification of mixed-paradigm models respectively.

First, mixed-paradigm process modeling is dealt with in this chapter. BPM in general often deals with the trade-off between comprehensibility and flexibility. Many languages have been proposed to support different paradigms to tackle these characteristics, as reviewed in Chapters 2 and 3. Procedural languages are often regarded as rather comprehensible, as the control flow can be understood from following the arcs and routing constructs, but have a harder time to support flexibility and ad-hoc changes. Declarative languages, on the other hand, offer a good way to model flexible processes, but face understandability challenges as extensively studied in Part 2. This chapter investigates in detail the scenarios in which combining both approaches is useful. It provides a scoring table for Declare constructs to capture their intricacies and similarities compared to procedural ones, and offers a step-wise approach to construct mixed-paradigm models. Such models are especially useful in the case of environments with different layers of flexibility and go beyond using atomic subprocesses modeled according to either paradigm.

6.1 Introduction

To be effective, process models need to be both expressive and understandable. To achieve a good balance, numerous languages have been proposed, each adding a certain aspect to the BPM language and tool sphere. The two main control-flow paradigms, extensively discussed in Chapters 2 and 3, and [59], both deal with the trade-off between comprehensibility and flexibility. The procedural paradigm is characterized by the use of explicit activity flows to express the activity paths through a process model, while the declarative paradigm is typified by a focus on curtailing behavior with activity-level rules rather than specifying entire activity paths, thus leaving many options for possible enactment. On the one hand, procedural models are regarded as rigid, but comprehensible, as they present the reader with what is possible in the process in a rather deterministic way. Declarative models on the other hand, leave much unspecified and therefore are harder to read, as the activity sequences allowed by the model remain implicit until they become visible during execution. Each paradigm also has solutions to leverage its issues with flexibility and comprehensibility. For example, in procedural process models one can loosen the typically explicit paths around a certain activity in, e.g., ad-hoc subprocesses, and in declarative models one can overly restrain a process path by using strong constraints such as the *chain* constraints, to obtain a stricter workflow.

This chapter investigates the possibilities of combining constructs of both paradigms in an intertwined model, supported by the semantics of both languages at the same time. More specifically, a combination of Petri nets with Declare is sought after. Mixed forms have already been discussed in [119] and [184], mainly focusing on execution. This chapter, however, rather focuses on the modeling effort itself and the constructs used for that task. Since many real-life processes are not completely flexible, nor completely fixed, the setup of mixing both paradigms offers business process modelers many applications. The contributions are as follows. Scenarios in which such models are useful are identified and the benefits they offer are explained. Also, the overlap and interplay of mixed-models' semantics and syntax is scrutinized by constructing a scoring table for Declare constructs. Constraints that obtain a higher score are more difficult to represent with Petri net-based constructs and hence are of a greater need in a mixed model. Finally, a step-wise approach is proposed for constructing mixed-paradigm models for future users, taking into account the different characteristics of both models.

Again, Petri nets and Declare are used to represent their respective paradigms. The R/I-net conversions of Chapter 3 are used to represent and score Declare constraints. In case they are used in a mixed-paradigm model, the converted model of separate constraints is behaviorally equivalent to a full conjoined model, as no interactions are present.

The remainder of this chapter is structured as follows. Section 6.2 gives a brief overview of both paradigms as well as the mixed form, and represents the different approaches graphically. Next, an example and use case for combining both paradigms is given in Section 6.3, after which Section 6.4 compares model constructs and different characteristics that are of high importance when mixing different paradigms. Finally, Section 6.5 provides a step-wise mixed-modeling approach and is followed by the conclusion which outlines future work in Chapter 6.6.

6.2 Procedural Versus Declarative BPM

The overview given in Chapters 2 and 3 is further extended with information that is of interest to elaborate on a mixed-paradigm approach. First, the procedural paradigm is briefly reiterated, followed by the declarative paradigm, and finally an overview of mixed approaches is established.

6.2.1 Procedural BPM

Process modeling has gained ground as a methodology to represent activities in a directed graph-like manner in order to capture and discuss business flows such as an ordering process, a customer journey, etc. [131]. For this purpose, many languages have been proposed, most notably Business Process Model and Notation [20], Petri nets [113], Event-Driven Process Chains [150], and Yet Another Workflow Language [157]. BPMN and EPC are often used in a business context and have been enhanced with numerous constructs supporting, e.g., message flows and ad-hoc processes. YAWL can be seen as an extension and effort to improve the stripped down Petri net execution semantics. Due to the simple, yet effective way Petri nets can capture flows and concurrency, they are widely used in numerous application domains. Their properties are well-studied and as such they remain very popular with researchers as well. Furthermore, the analysis techniques such as state space generation and soundness checks [151] make them the preferred language to which BPMN models and EPCs are translated to in order to provide firm execution semantics and model checking [42, 150].

6.2.2 Declarative and Flexible BPM

Flexibility has numerous forms, such as flexibility by design, deviation, underspecification, and change, which are described in [137]. Flexible process models tend to make use of these concepts, mostly in certain parts of the model, e.g., pockets of flexibility [134] and worklets [1]. These are approaches for enabling procedural models to include flexible behavior by postponing and underspecifying execution decisions until run-time.

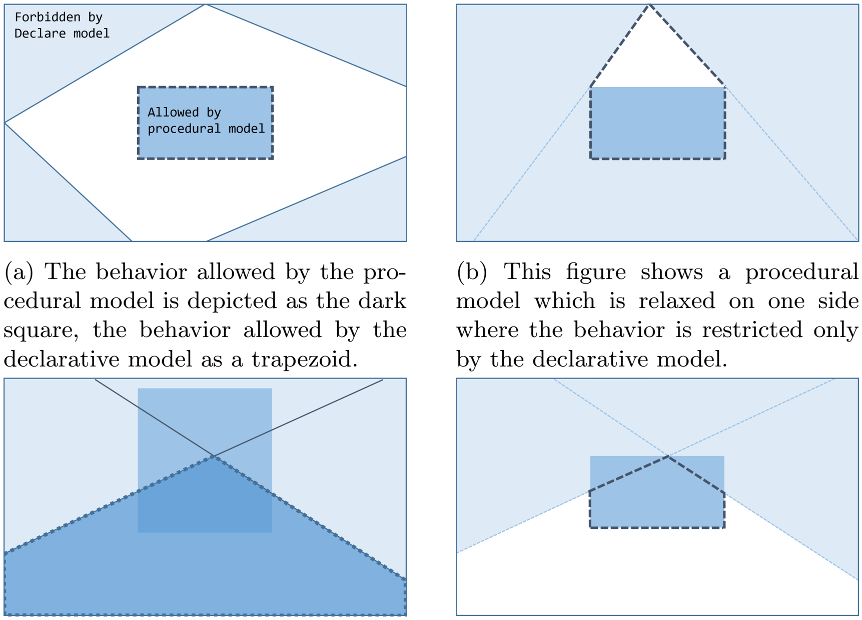
The major difference between procedural and declarative modeling is the way in which one approaches the model: either a specification of what has to happen (procedurally) is made, leaving no room for non-modeled behavior, compared to specifying what can happen, where everything that is not prohibited is possible (declaratively). Hence, declarative models leave more room for non-modeled behavior and thus are regarded as allowing more flexibility in the process execution. Declarative process models, and the event- and rule-driven Declare framework [118, 119] in particular, have gained attraction amongst researchers as a provider of a completely flexible solution.

The differences and characteristics of how modelers and users apply both paradigms, have been researched extensively by [127, 65, 49]. The outcomes suggest that, overall, it is very difficult to read Declare models due to the invisible execution of accepting and non-accepting behavior, the lack of clear sequences in the beginning and ending, the subtleties of the constraints, and especially the complex interaction of the different templates, as discussed in Part 2. Literature often suggests to model very sequential information with procedural languages, and to model flexible processes with declarative languages instead, in analogy to procedural and declarative programming.

The different types of behavior of the two paradigms can be depicted as in Figure 6.1. To the left, Figure 6.1a shows the traditional representation of both paradigms as in [119]. Usually, Declare is referred to as the model type that constraints behavior by activity-level rules, leaving options open for more flexible specification and execution, thus accommodating for everything that is not forbidden. Procedural models are depicted as very rigid process flows, containing only strictly regulated and delineated behavior.

6.2.3 Mixed Forms and Conversion

Modeling languages incorporating both paradigms also exist, but still focus rather on separate subworkflows, modeled with either procedural or declarative constructs, in order to keep the state spaces and execution semantics of these subworkflows separated. This has been proposed for, e.g., YAWL and Declare [160]. This approach is similar to pockets of flexibility. Thus, flexibility is introduced into some parts of the process in a hierarchical way. This



(a) The behavior allowed by the procedural model is depicted as the dark square, the behavior allowed by the declarative model as a trapezoid.

(b) This figure shows a procedural model which is relaxed on one side where the behavior is restricted only by the declarative model.

(c) The model is a pentagon using both model paradigms to account for the different levels of flexibility.

(d) This figures shows a procedural model which is even further restricted by declarative constraints.

Figure 6.1: Three layers indicating all the possible behavior of the activities and flow constructs contained in a model. The dotted line represents the outcome of a combination of declarative and procedural constructs in (b), (c), and (d).

is depicted in Figure 6.1b. Typically, a certain part of the model is loosened, e.g., the procedural model loosens a certain part which is now constrained by activity-level rules. The opposite is also possible, in that a very flexible main process contains some fixed sequences which can be easily captured by, e.g., a small Petri net fragment.

Execution semantics for truly intertwined state spaces exist as well. In [184], execution semantics for Petri nets and Declare automata are presented. Intertwined state spaces can also be constructed by mixing converted Declare constraints expressed in Petri net constructs with other Petri nets, thus obtaining a mixed-model. In [48], the possibility of converting a subset of DecSerFlow constraints, the predecessor of Declare, has been investigated. A full conversion is sought after in Chapter 3. The conversion of Declare constraints based on regular expressions has been researched in [124]. By making use of synthesizing finite state machines into Petri nets with the theory of regions [28], Declare constraints can be converted to Petri nets.

This technique is similar to enumerating all possible execution scenarios, as many duplicate activities are required to do so.

A process with mixed layers of flexibility which spread throughout the whole state space of the model cannot be captured by using solely subworkflows, as this setup requires the models to synchronize to a state before and after executing the sub-workflow. For instance, an activity which can appear to be rather flexible, i.e., without a fixed place in a sequence, but which still affects a procedural part of the model cannot be modeled outside of its subworkflow. In a true mixed-paradigm approach with intertwined state spaces, process behavior is restricted by making use of the most appropriate combination of subsets of both models, thus combining modeling constructs that restrict the process behavior in some directions, but relax behavioral constraints in other directions. This is depicted in Figure 6.1c, where a subset of both models constitutes the mixed-paradigm model. Still, one paradigm can dominate the other (e.g. the example in Figure 6.6 where the declarative part clearly dominates the procedural part), however, they can also have equal influence on activities. Furthermore, not only flexibility can be achieved, but also an especially strict specification. In Figure 6.1d, the Declare constraints cut into the procedural model, resulting in a less flexible model as sequence rules impose even further restrictions on the workflow.

6.3 Running Example of a Mixed Model

Consider the mixed-paradigm model in Figure 6.2 which contains a procedural backbone that is supplemented with a flexible component containing activities *Call customer* and *Start logging*. The flexible part starting with activity *Start logging* can execute irrespective of the behavior modeled in the procedural backbone, but still influences the main process. The inclusion of *chain response(Start logging, Call customer)* (i.e. after *Start logging*, *Call customer* has to happen next) disrupts the global model, as every activity but *Call customer* becomes disabled after firing *Start logging*. *Call customer* has to happen before *Send invoice* can ever occur (due to the *precedence* constraint), and *Close order* can only fire again after a new occurrence of *Call customer* (due to the *alternate precedence* constraint).

The combined use of procedural and declarative constructs results in an effective alternative solution, in-between solutions that would use declarative or procedural model constructs exclusively. By explicitly capturing the loop with Petri nets and reducing the amount of Declare constraints, readers and modelers can easily grasp the token game while a few verbose sequence rules (which can be found in Table 3.2) can explain the interplay of the flexible activities with the procedural net.

Modeling the same scenario with a procedural language such as Petri

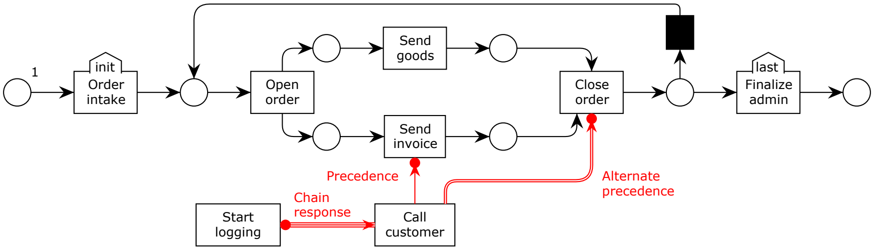


Figure 6.2: A very straightforward AND-split and -join based process model represented in a mixture of Petri nets and Declare in standard notation.

nets, results in either a model with many duplicate activities, or reset and inhibitor constructs, as depicted in Figure 6.3. Especially capturing *chain response* severely disrupts the main process which needs the incorporation of many (inhibitor) arcs that clutter up the model completely.

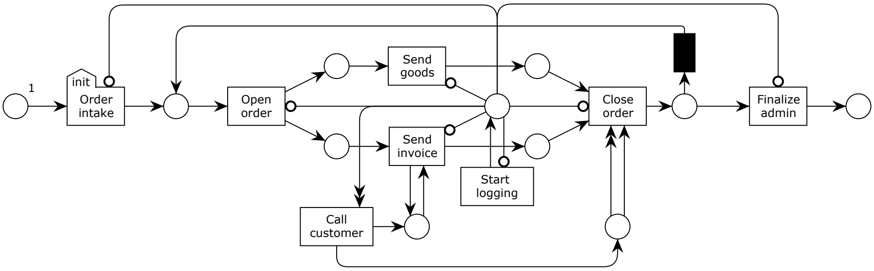


Figure 6.3: The same model as in Figure 6.2, but now solely in R/I-net constructs.

Using Declare, it is hard to capture the procedural backbone in a straightforward and comprehensible way. To capture the same behavior as the loop does, one needs many *alternate succession* constraints in which the loop remains hidden. Also, a *chain precedence* constraint is required to model the XOR-split at the end of the loop. By providing readers solely with the standard constraint description, interpreting the model requires a significant amount of cognitive effort. In the end, a Declare model is not executable unless transformed into an automaton. The FSA of the running example is displayed in Figure 6.5. The flexible activities are indicated in red. The state space is the same for all the models and in the automaton, it is clearly visible how the state spaces are intertwined. The procedural behavior only needs a few state transitions, while the flexible behavior requires the inclusion of many of them, even though only three Declare constraints are used in

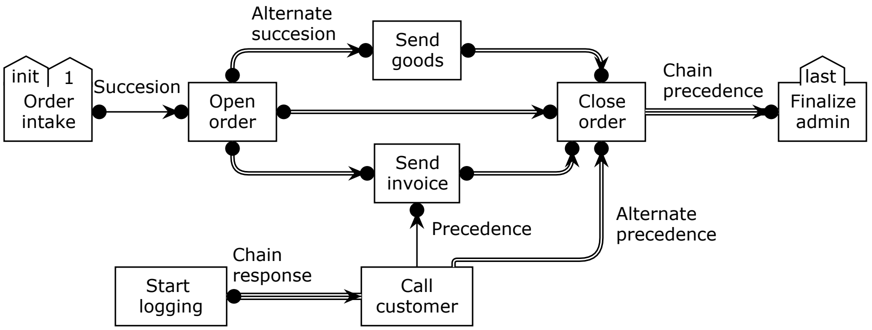


Figure 6.4: The same model as in Figure 6.2, but now solely in Declare standard notation.

the case of the mixed model in Figure 6.2. It shows just how much influence the flexible part of the model has over the global behavior in terms of the number of execution paths it invokes.

Observe that using a subworkflow for *Call customer* and *Start logging* is not possible. Since both activities affect the main workflow, one cannot simply model these activities in a concurrent subworkflow as, e.g., the impact of the *chain response* is global and not restricted to both activities involved. Therefore, mixed-paradigm modeling attempts that only allow a combination of paradigms by making use of fully separated subprocesses modeled with either type of constructs, are not able to model the desired behavior appropriately.

6.4 Constructs and Characteristics

Incorporating both modeling syntaxes and semantics into a single model requires carefully scrutinizing the different constructs and avoiding overlap as much as possible. In this section, a scoring mechanism for Declare constraints is presented according to different characteristics, which makes it possible to assess how straightforward it is to express them in R/I-net constructs, whether the constraint impacts global concurrency and global timing, and whether they inflict hidden dependencies. These characteristics play an important role for merging procedural and declarative process models.

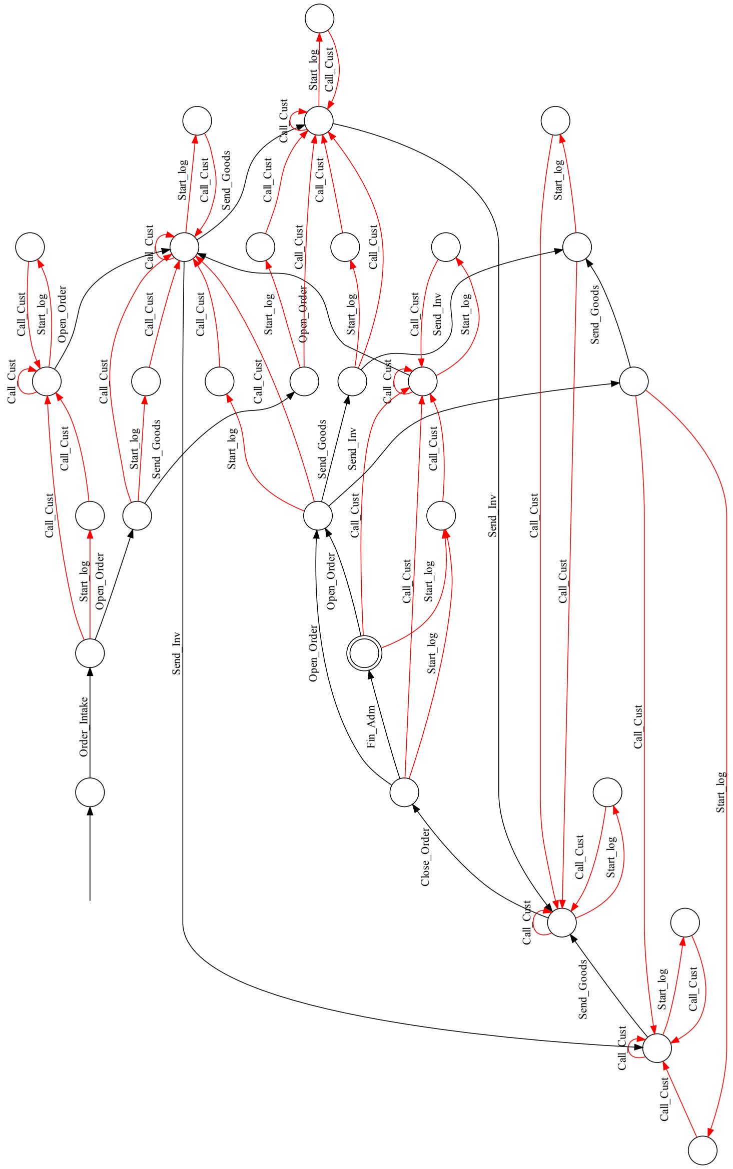


Figure 6.5: The automaton for the Declare model with the flexible activity transitions in red.

	R/I-net constructs					**Impact global concurrency (2)**	**Impact global timing (5)**	**Permanently disabling (10)**	**Interchangeability score**
	P/T	**A**	**R**	**I**	**Sum**				
Init	1			|T|-1	|T|				|T|
Last			|P|		|P|		✓		|P|+5
Existence	1	2			3		✓		**8**
Absence	1	2			3			✓	13
Exactly	1	2		1	4		✓	✓	19
Response	1	1	1	1	4		✓		**9**
Precedence	1	3			4				**4**
Succession	2	4	1	1	8		✓		13
Alternate response	2	2	2	1	7		✓		12
Alternate precedence	1	2	1		4				**4**
Alternate succession	2	4		1	7		✓		12
Chain response	1	1	1	|T|-1	|T|+2	✓	✓		|T|+9
Chain precedence	1	|T|-1	1	1	|T|+2	✓			|T|+4
Chain succession	2	2	2	|T|	|T|+6	✓	✓		|T|+13
Responded Existence	2+1T	3	2	1	9		✓		14
Co-existence	3+1T	6	3	1	14		✓		19
Not succession	1	1		1	3			✓	13
Not chain succession	1		|T|-2	1	|T|	✓			|T|+2
Not co-existence	2	2		2	6			✓	16
Choice	1		2	1	4		✓		**9**
Exclusive choice	3	2	2	3	10		✓	✓	25

Table 6.1: Scorecard of Declare constraints for the number of R/I-net constructs needed, and semantic characteristics. $|T|$ and $|P|$ stand for the number of transitions and places in the model respectively.

6.4.1 Construct-based Similarities and Differences

Declare consists of many constraint templates which have distinct features that require a large amount of Petri net constructs to mirror their behavior, as can be seen in the R/I-net construct column of Table 3.1. However, many other templates exist that can be straightforwardly represented with only a few Petri net constructs. Therefore, these constraints can be easily interchanged in mixed-paradigm models to avoid using different syntaxes. The advantage of R/I-net constructs is that the syntax immediately yields execution semantics. Each constraint is thus scored for the amount of places (P) and occasionally transitions (T), arcs (A), reset arcs (R), and inhibitor arcs (I) that is needed to express them. Each construct is scored for 1 point.

6.4.2 Impact on a Global Concurrency Level

Constraints can force activities, not directly related to them by other constraints, to be disabled. Hence they have a direct impact on global concurrency. Most notably, the *chain* constraints exhibit this behavior, as they can stop any activity from executing until a certain other activity has fired. Not only does this require many constructs such as inhibitor arcs or prioritized Petri nets to model this in a procedural model, they also impact the execution semantics of, e.g., a Petri net mixed with a Declare model containing *chain* constraint(s). This makes it harder to model and understand the behavior of such mixed-models. This is scored with 2 points in Table 6.1.

6.4.3 Impact on a Global Temporal Level

The concept of temporary violation is typical for rule-based approaches. It can be compared to (non-)final markings in a Petri net. In the R/I-net, a dedicated sink transition t_{sink} is used to indicate the current violation status of the model (firing it leads to the accepting marking of a single token in p_{sink}). If the transition is enabled, no temporary violations are present (permanent violations cannot appear by default). Adding this explicit monitor helps users grasp the status of the net. Many constraints make use of this construct. However, introducing it increases the total number of constructs needed and also requires a procedural model mixed with a Declare model to be able to also resolve the same temporary violation(s). This raises the efforts needed to model correctly. Since this has a major impact, especially on the possibility to synchronize Declare with any other models in terms of temporal consistency, this is scored with 5 points in Table 6.1.

6.4.4 Permanently Disabling

Some Declare constraints require activities to become permanently disabled when they become activated. Most notably, *absence, exactly, not succession, not co-existence*, and *exclusive choice* disable at least one activity for the rest of the execution, as illustrated in Chapter 4. The dependencies that are caused and propagated through the net have a vast impact on how separable the declarative process model is. Hence, it should always be checked how far the reach of the dependency structures extends over the activities, especially in mixed-paradigm models. Again, the activity bounds of a Petri net model can be verified next to the dependency structures of the Declare model, as illustrated in Section 5.4. Nevertheless, executing models containing these constraints is extra precarious. Hence, constraints inflicting such behavior are hard to incorporate in two semantics at the same time, thus a high score of 10 is given to such constraints.

6.4.5 Overview

Taking into account all these different aspects of the constraints, a final score is assigned. The lower the score, the better. Constraints with a score below ten are easily pluggable into a procedural model. Between ten and twenty, considerable care must be taken. For constraints with a score above 20, it becomes very tedious to include them in a procedural model. E.g., the *chain response* constraint template requires one place, one Petri net arc, one reset arc, and inhibitor arcs connected to all other transitions in the net but one ($|T| - 1$). Hence, the constraint impacts global concurrency, as it can stop all activities in the net but one, and impacts global timing as it can be in a temporarily violated state. In the end, it receives a score of $(1 + 1 + 1 + |T| - 1) + 2 + 5$. Since $|T|$ is included, using this constraint in bigger models with more transitions becomes tedious to the extent that every other transition should be checked for the way its behavior is related to the activities embedded by the constraint.

As can be seen from the last column in Table 6.1, only a few constraints are considerably straightforward to model, comprehend, and use in a mixed-paradigm model:

- The simple and alternating ordered constraints are not impeded by the fact that they do not expose sophisticated behavior nor many constructs. This is especially true for *precedence* constraints.

- Every constraint that impacts global concurrency or inflicts hidden dependencies causes severe synchronization problems. This includes, among others, the *chain*, *absence*, and *existence* constraints.

- Although their principle is simple, *not co-existence* and especially *exclusive choice* are very hard to incorporate in a mixed-paradigm model due to their impact on the enabledness of activities.

6.5 A Step-wise Approach for Mixed-paradigm Modeling with Intertwined State Spaces

The insights gathered from the previous section, as well as Chapters 4 and 5, a step-wise approach to mixed-paradigm process modeling is devised. The method is also illustrated with a well-known BPM example.

6.5.1 Step-Wise Mixed-Paradigm Modeling Approach

Table 6.1 can be used by mixed-paradigm modelers to assess the influence of certain constraints on the model and what the consequences of using them might entail. By applying the scores, it now also becomes possible to objectively start measuring different mixed-paradigm solutions in terms of comprehensibility (in terms of the amount of model constructs), and the semantic difficulties that are introduced. A step-wise approach to leverage the insights of the scores is as follows.

1. **Determine for each activity whether its behavior can be contained in a procedural workflow, or rather requires a looser setup with rules.** By indicating where in the process an activity can and should occur, it will reveal the extent to which it requires flexibility.

    - If the position of the activity is not fixed within the workflow, it is better to exclude it from the procedural model.
    - If the activity occurs a predefined number of times, Petri nets might be used, or a *Unary* Declare constraint. Otherwise, it may prove hard to use a token game around the activity, as an undesired amount of tokens might be pushed down the model, which could require adding silent transitions to model skipping steps.

2. **Determine which relationships are needed between the different model types.** In a mixed-model, there are 4 different types of relationships, given that activities are labeled 'Declarative' or 'Procedural' in step 1:

    - Declarative-Declarative,
    - Declarative-Procedural,
    - Procedural-Declarative,
    - and Procedural-Procedural.

    The second and third types constitute the real mixed cases. In the case of using them, it is advisory to consult Table 6.1 to check for characteristics towards violation and temporal issues. Generally, it is advised to avoid using binary Declare rules between activities solely present in the procedural part of a mixed-model. Although it is possible to do this, it is better to approach the procedural part from the outside to avoid internal anomalies such as deadlocks. Only the construction of the state space of a Petri net can show whether the resolution of, e.g., temporary violations is still possible. Hence, it is advised to avoid constraints that have, e.g., a global impact on concurrency and timing. Also, hidden dependencies propagate through the procedural model. Therefore, these constraints are best used in isolation within a declarative model. Safe connections between procedural and declarative parts are mainly *precedence* relations, and any constraint that does not impact global timing and hidden dependencies.

3. **Synchronize beginning and end points of both model types if possible.** By using *init* and *last* constraints in combination with a Petri net source and sink transition, the models are intertwined in a proper way. The inclusion of separate sink activities might be required.

4. **Check whether it is necessary to use two types of language constructs.** According to the scores in Table 6.1, several constraints are easy to model in Petri nets with R/I-net constructs. Replacing them, while still referring to them with their Declare constraint name, avoids multiple modeling notations. Furthermore, R/I-net constructs yield executable syntax, hence making the construction of an automaton obsolete in many cases (not, however, where there are hidden dependencies or multiple violation states).

6.5.2 Reworking an Existing Example with the Approach

In this section, it is shown how to transform a procedurally modeled order fulfillment process ([44], page 77), and expand it with declarative constructs. Also, it is shown where gaps still exist between the two approaches.

The setup of the order fulfillment process, however, is interpreted slightly differently than in [44]. In the scenario presented here, multiple orders can be made and at least three product shipments and payments have to have happened before the archiving of an order. Furthermore, the requests for raw materials can now only be done directly after checking their stock level, and obtaining the materials always has to happen directly after requesting them.

1. In the original model, every activity is rather fixed within the sequence. Due to the unspecified amount of occurrences of *Receive order* and its successors it becomes more interesting to use declarative constructs, as they are better capable of mixing different strings of activities while maintaining a somewhat structured process. Hence, everything up to *Confirm order* is rather declarative, while the shipping and invoicing processes are kept procedural.

2. Some relationships, as indicated in Table 6.1, are easier to express in Declare. Most notably, the use of *chain* relationships to indicate directly follows parity, and the use of *alternate precedence* for an unspecified amount of occurrences of activities around *Receive order* are more convenient and avoid the model from becoming too convoluted. It is more tedious to express that *Archive order* needs at least three occurrences of *Ship product* and *Receive payment* in Declare, as it is harder to count in regular languages than in Petri nets. Also, it is clearer to do so by keeping track of the tallying with tokens. Finally, some Declare constraints are used to connect the material and invoicing and shipping parts.

3. Beginning and end points are synchronized through the *init* and *last* constraints. In this case, the *last* constraint for *Archive order* has a global impact on the declarative part of the model as well, most notably on *Manufacture product*.

4. As can be seen in Table 6.1, the *precedence* constraints can be expressed with R/I-net constructs. *Not succession*, however, requires special care in this case, as it has an impact on dependent activities both in the declarative part as well as in the procedural part due to propagation of dependency (disabling *Receive order* also disables all succeeding activities in a *precedence* relationship).

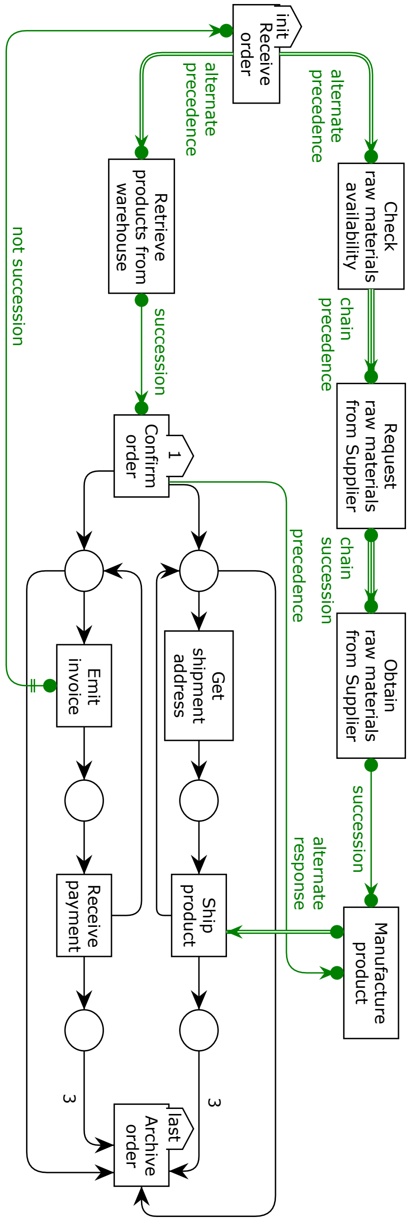


Figure 6.6: A well-known fulfillment process model reworked according to the step-wise approach.

In the end, using different syntaxes in mixed-paradigm models is also of interest as it can better indicate which parts of the model are procedural, and which ones are declarative. A trade-off is present between applying different representations and coloring, and making the model as uniform as possible in terms of syntax as well.

6.6 Conclusion and Future Work

This chapter explored the gap and similarities that exist between procedural and declarative process modeling approaches, focusing on understandability, syntax, and execution semantics. More specifically, it was researched which possibilities arise when both paradigms are combined when state spaces become intertwined, as opposed to previous approaches that only use separate atomic sub-workflows. It was found that there is a trade-off between syntax that yields execution semantics, and verbose Declare constraints with many implications for execution. A scoring table for Declare constraints was presented, which can be used for objectively assessing the complexity of mixed-models, enabling the comparison of different mixed-paradigm solutions and guiding modelers when faced with the challenge of selecting appropriate constructs. Finally, a step-wise approach was proposed for mixed-paradigm modeling, which consolidated the insights from the scoring table. An example was elaborated in which the trade-offs that arise when constructing mixed-paradigm models are illustrated and made explicit by making use of the step-wise method.

CHAPTER 7

Fusion Miner: Process Discovery for Mixed-Paradigm Models

"Wait I know you can love your sons and your daughters
And no one will tell you it's a wrong thing to do
But if you try to give your love
To more than one woman
The whole damn world is gonna look down on you."
— Paul Gilbert, One Woman Too Many

After spending a great deal of scrutiny on the human-driven type of process discovery, process modeling, *this chapter shifts its focus to the automated discovery of processes from historic data logs.*

As illustrated in Chapter 1, process mining is an interesting venture that has gained a lot of attention from the BPM research community in recent years. A strong emphasis lies on the automatic discovery of models for which numerous algorithms have been proposed already. Also for the declarative process paradigm, many techniques have been designed, which comprise a large part of all research on DPM as mirrored by the number of publications on the subject that were described in Chapter 2. So far, most discovery algorithms were limited to the derivation of single-paradigm models, which contain either procedural or declarative constructs, targeting the mining of strict and flexible processes respectively. *This chapter proposes the first fully-automated mining technique to discover procedural workflows combined with Declare templates to capture processes that are difficult to mine with only a single paradigm, i.e., workflows with different layers of flexibility.*

This approach provides process analysts with new discovery capabilities, including the retrieval of better fitting and more precise models with high comprehensibility. The main contribution consists of the Fusion Miner algorithm, which has been implemented in the process mining framework ProM as a plug-in.

7.1 Introduction

The field of process mining has gained a lot of traction in the last decade. Its main focus lies on the automatic retrieval and subsequent analysis of business process models and insights from data logs containing events [152]. As such, process mining can be a powerful approach for decision makers in terms of assessing and improving business processes.

As explained in Section 1.2.2, the three pillars of process mining focus on process discovery, enhancement and conformance checking. The former can be considered the primordial task in a process mining exercise, supporting the two latter pillars that are typically built on top of it. The goal of process discovery is to learn a process model from data in the form of an event log in the most comprehensive, comprehensible and correct way. In order to do so, many mining algorithms have been proposed, including, amongst others, Alpha Miner [158] and Heuristics Miner [178]. Generally, these miners retrieve procedural models, containing strict sequence information, such as Petri nets [113] and Causal nets [162].

More recently, declarative process modeling and mining has gained popularity and several discovery techniques such as, e.g., Declare Miner [95] have been implemented for discovery purposes. These miners derive rules from event logs to create models with a more flexible view on the information contained in the log, as any behavior that is not strictly forbidden is allowed.

In accordance with the “maps” view on process models proposed in [154], one retrieves different information by mining for different paradigms. Similar to reading maps with different perspectives which cover multiple layers of an area, it is possible to retrieve different paradigms at once to gain complementary insights from the information retrieved from the log. For example, combining street maps with altitude information can provide a deeper understanding of the explored area.

In line with this metaphor, *this chapter presents an approach based on combining procedural model fragments and declarative constraints in one map. As such, a new way of mining and representing process models is proposed.* This process mining approach exploits the different characteristics of both process paradigms, with each paradigm better capable of representing distinct behavioral aspects of an event log. In the literature, the combination of both paradigms into one discovery algorithm has received very little attention so far. Nonetheless, the prospect of learning richer, mixed-paradigm process models seems promising, especially in the context of semi- or unstructured processes. The proposed approach has been implemented in ProM[1] as a plug-in and borrows some key principles of the two most frequently used process mining techniques for each paradigm, Heuristics Miner and Declare Miner. The results show models that are fitting and precise for event logs that contain different layers of flexibility, meaning they show behavior consisting of both strict event sequences and rather loosely defined event occurrences. This improves current single-paradigm approaches regarding both conformance and comprehensibility.

The remainder of this chapter is structured as follows. Section 7.2 covers the related work on process mining in more detail. This is followed by Section 7.3, which reiterates the different blends of the procedural and declarative paradigm, applied to a mining context. Section 7.4 reveals the algorithm to mine mixed-paradigm models and discusses criteria to evaluate them. Section 7.5 then evaluates two simulated examples and one real-life case by comparing current techniques and the outcome of Fusion Miner. Section 7.7 concludes the chapter with a discussion of the contributions and results.

7.2 Related Work

Within the field of process mining, a strong emphasis is put on the automatic retrieval of business process models from event logs. The algorithms offer mining solutions that deal with aspects such as the trade-off between recall, precision, generalization, and the presence of noise in the event log [168]. Initially, procedural process mining approaches were put forward. Some well-known and widely used ones are the initial techniques of Hwang [79] and Agrawal [8], Alpha Miner [158], Heuristics Miner [178], Fuzzy Miner [61], ILP Miner [164], Inductive Miner [91], and Fodina [166]. They capture sequence constraints and parallelism by incorporating information supporting adjacency and (direct) succession in a process log, extended with (X)OR- and AND-split and —join information. Most of the constructs that are sought after have local semantics with some extensions for long-distance dependencies, but are calculated on log level. Approaches for stochastic Petri nets exist as well [130], which can be used for predicting future execution steps [129].

The declarative process paradigm quickly picked up on mining as well, and soon after its conceptualization DecMiner was introduced [89, 19, 18]. This mining algorithm learns SCIFF models by using inductive logic programming and is able to transform the output into DecSerFlow. Later, the Declare Maps Miner was introduced in [93] and improved in [95]. The al-

[1] http://www.processmining.org/

gorithm starts from finding frequent activity sets based on Apriori learning [7], for which instantiated Declare templates in LTL are made to replay the traces in the event log. Hence, the outcome is a set of Declare constraints with a certain support and confidence to express the usefulness of the results. Competing techniques that rather use regular semantics are MINERful++ [22] and UnconstrainedMiner [185], both achieving considerable performance improvements by relying on temporal statistics to construct a knowledge base to derive constraints, and a mixture of superscalarity and symmetry reduction respectively. They also rely on support and confidence and output Declare models.

There exist some approaches on the verge between both paradigms, such as Fuzzy Miner [61], UnconstrainedMiner [185], and AGNEs Miner [58]. The former captures flexible processes by incorporating multiple perspectives such as social network and control flow information (note that this is different form of a mixed-paradigm approach). The algorithm makes it possible to visualize process models, called Fuzzy models, that highlight the sequences within the log that have the highest impact on these perspectives, which enables the capturing of flexible models better than, e.g., Alpha Miner. UnconstrainedMiner approaches the mixed-paradigm perspective the other way around, starting from a declarative rule mining implementation which can incorporate any constraint based on regular expressions. As such, the technique is able to incorporate very procedural constraints. A similar approach is used in [58], where the authors propose a discovery algorithm based on NS (No-Sequel), a declarative predicate which can be used to derive constructs that form a Petri net. A genetic variant was introduced in [170].

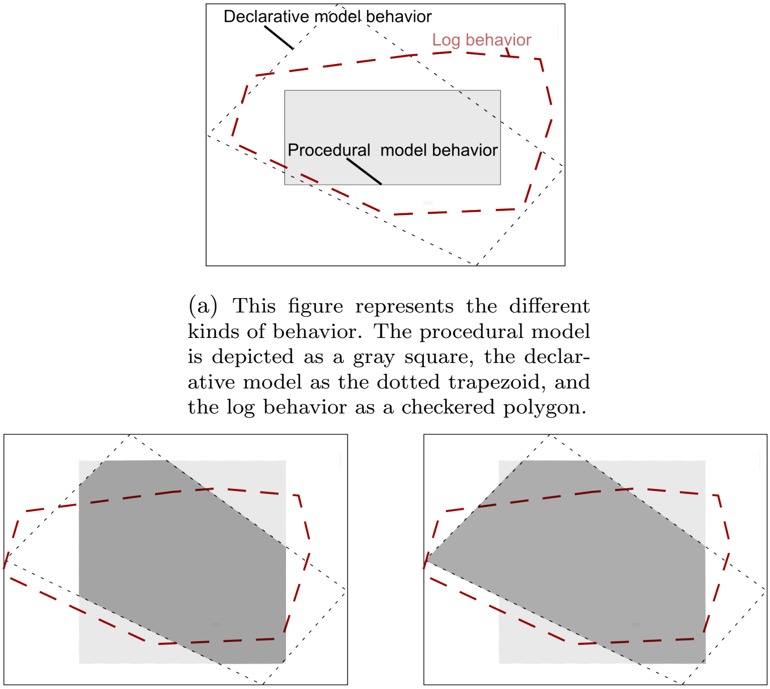
A real mixed-paradigm outcome, however, has not been pursued yet as such, with the very recent exception of [98] in which the authors break down the event log into a hierarchy and mine the different subprocesses according to the appropriate paradigm. While the purpose is somewhat similar to the one pursued in this chapter, there are some noticeable differences. First of all, the authors assume atomic subprocesses, which are still disjoint. This restricts synergies between both paradigms in terms of state space, as they cannot interact to mine and represent process behavior. Furthermore, while the authors propose an approach based on counting predecessors and successors which is somewhat comparable to direct succession in Section 7.4, applying a threshold on the exact number of predecessors and successors seems rather coarse for deciding on the structured versus unstructured nature of activities, especially for smaller logs. Therefore, the approach proposed here includes a configurable threshold based on the concept of entropy, which allows for making a more versatile trade-off between structured and unstructured behavior. In addition, the mandatory hierarchical structure of the mined hybrid model in [98] puts a limitation on its application to event logs where structured and unstructured behavior are much more intertwined. The approach presented in this chapter does not presume such a hierarchical structure. Finally, the approach in [98] is not setup as a discovery algorithm and rather serves as a log preprocessor.

7.3 Mixed-Paradigm Models for Mining

The models mined by Fusion Miner are built out of dependency graphs and Declare constraints. For modeling purposes, mixed-paradigm models have already been proposed for, amongst others, logic and procedural patterns [87], YAWL and Declare [119] and Petri nets and Declare [184], as documented in Chapter 6. The execution semantics of the first approach are based on calculating process trees and applying change operations on the tree for the constraints afterwards. The semantics used in the second one are not described explicitly, but they are handled by ensuring atomicity of execution by using subprocesses of either paradigm. The last approach provides an in-depth analysis of executing Declare rule automata and Petri nets. The authors suggest three ways of enactment (the last one only applicable to data-aware nets), of which the first approach called *Simple Simulation* is used in CPN Tools [181]. It boils down to constructing the automaton for each Declare constraint separately and updating them during execution. Making the conjunction of all separate Declare constraints, as described in Chapter 3, is called *Smart Simulation*, and merged on-the-fly with the Petri net it is combined with. In this chapter, it is assumed to execute mixed models of Declare and Petri nets as described in the last approach.

The choice for dependency graphs and Declare is founded on their application in two very well-known and supported mining algorithms, namely Heuristics Miner and Declare Miner (and other declarative process miners). The dependency graphs can be converted to Causal nets or Petri nets afterwards, as some examples in this paper show. Numerous techniques exist for this purpose and are incorporated in, e.g., Heuristics Miner. Declare models were defined previously in Chapter 2. However, here the set of activities in the Declare model are referred to as the set $D \subseteq A_{MPM}$, constituting the model $DM = (D, \Pi)$, as A_{MPM} represents the finite set of activities present in the mixed-paradigm model. A dependency net DN is defined as a tuple of activities or nodes B and the flow in the net $Z = B \times B$, constituting a directed graph $DN = (B, Z)$, with $B \subseteq A_{MPM}$.

The different behavior of the models over activities A_{MPM} can be represented as proposed in [119]. Figure 7.1a shows the declarative behavior in the trapezoid-like, dotted shape and the procedural behavior in light gray. While the figure was proposed for modeling, it is now applied to mining. Therefore, the behavior contained in the log should be included. It is represented by the checkered, unstructured polygon.



(a) This figure represents the different kinds of behavior. The procedural model is depicted as a gray square, the declarative model as the dotted trapezoid, and the log behavior as a checkered polygon.

(b) This figure shows the intersection of the behavior of both models in the dark gray area. Both models restrict each others behavior to provide a stricter outcome.

(c) This figures shows one possible subset of the union of the behavior of both models in dark gray, which is a possible outcome of the Fusion Miner approach as will be illustrated below.

Figure 7.1: Graphical representation of the behavior allowed by the models and present in the log.

We propose a mixed-paradigm model as follows: let MPM be a tuple $MPM = (B, Z, D, \Pi)$. As such, both rules and sequences are defined over the activities. The outcome of the model is then any part of DN and DM, $MPM \subseteq DN \cup DM$, which constricts the behavior of A_{MPM} in different ways:

- **A more narrow result:** $DN \cap DM$: by taking the intersection of the behavior allowed by both models, it becomes possible to more strictly describe the process flow. This is represented by Figure 7.1b, where the intersection is indicated as the dark gray part.

- **A mixed result:** $DN \cup DM$: by taking the union of all behavior, it becomes possible to capture behavior in the log that previously remained undiscovered (typically in the procedural model) or was too broadly captured (typically in the declarative model). By taking subsets of this union, one can more closely retrieve the behavior in the log. This is shown in Figure 7.1c, where one possible subset is indicated in dark gray.

The basic idea behind the mining algorithm explained below is the division of the set of activities A_{log} in the event log into the two sets introduced above, namely the procedural activities B and the so-called entropic activities D. We define entropic activities as activities of which the behavior is hard to capture in a strict procedural process flow. Arcs in mixed-paradigm models can be of the following type:

- $\mathbf{DD} \subseteq \mathbf{D} \times \mathbf{D}$**:** arcs between entropic activities are represented by Declare constraints.

- $\mathbf{DB} \subseteq \mathbf{D} \times \mathbf{B}$ **and** $\mathbf{BD} \subseteq \mathbf{B} \times \mathbf{D}$**:** arcs between entropic and procedural activities are either expressed in Declare constraints or present in the dependency net, with a prioritization of the former.

- $\mathbf{BB} \subseteq \mathbf{B} \times \mathbf{B}$**:** arcs between procedural activities are represented by the dependency graph.

As such, there exist activities that are either completely captured in the Declare model, $A_{DD} \in D$, only captured in the dependency net, $A_{BB} \in B$, and activities present in both models, $A_{DB_D/BD_D} \in D$ and $A_{DB_B/BD_B} \in B$. Note that $D = A_{DD} \cup A_{DB_D} \cup A_{BD_D}$ and $B = A_{BB} \cup A_{BD_B} \cup A_{DB_B}$. An overview is given in Table 7.1. It has to be avoided that the paradigms become too convoluted, as the mining of both model types simultaneously is based on heuristics and cannot assure that there are no contradictions in the state spaces of both models. In Chapter 8 a technique for checking the compatibility of the separate models is discussed. There exist other model checking techniques as well, such as the one described in [46], which can convert both models into automata. The synchronous product could yield errors which can be taken into account during process discovery. However, the overhead introduced by these techniques is significant. A further digression on how this can be used for discovery will be further elaborated on in Chapter 8 as well.

However, in order to keep both model types somewhat separated, Declare constraints are prioritized as mining and representation form, as the semantics of LTL formulae are richer than a relationship in a dependency graph. Declare constraints are mined for all activities, except the ones in

A_{BB}. If no constraints are discovered for an activity in D, the algorithm is resilient and puts the activity in B.

Activities	**D**	**B**
D	$D \times D$	$D \times B$
	Declare constraints	Declare Constraints
		(Dependency graph connection)
Contains	A_{DD}	$A_{DB_D} \cup A_{DB_B}$
B	$B \times D$	$B \times B$
	Declare Constraints	Dependency graph
	(Dependency graph connection)	
Contains	$A_{BD_D} \cup A_{BD_B}$	A_{BB}

Table 7.1: The four different types of connections in a mixed-paradigm process model.

7.4 Fusion Miner

The Fusion Miner implementation[2] is based on the combination of Heuristics Miner [178] and Declare Miner [96]. Starting from dependency information, it identifies activities that are connected to a relatively higher number of other activities in the dependency graph, as these can be considered a causal factor of the increase in potential process behavior. As such, they are targeted for inclusion in the set of activities that are subject to Declare mining in the second part of the discovery process. Finally, the Declare model can be pruned optionally and the mining result is displayed in a model containing both paradigms.

7.4.1 The algorithm

The algorithm starts off by calculating dependency measures for log $\mathcal{L}$ containing the activity alphabet $A_{\mathcal{L}}$ with Heuristics Miner. This results in a tuple $(DG, DS, B2B, LDD)$ with $DG = A_{\mathcal{L}} \times A_{\mathcal{L}}$ the flow relations in the dependency graph, DS the direct succession values between $A_{\mathcal{L}}$, $B2B$ the level 2-loop values, and LDD long distance dependency values. By analyzing the strength of the direct succession metric (which is controlled by a threshold d called Dependency threshold) between activities $A_{\mathcal{L}}$, one can retrieve the activities that are closely related in a small window containing local neighbors in the log. Activities that are connected to many others for a considerable number of times, or are somewhat but not strongly connected,

[2] The implementation and high resolution figures can be found at `http://www.processmining.be/fusionminer/`.

are candidates to be placed in the set of declarative activities $D \subseteq A_{\mathcal{L}}$. Others that have few but strong connections to neighboring activities, are candidates to remain in the procedural part of the model $B \subseteq A_{\mathcal{L}}$. Phrased differently, activities with unclear direct succession relations are targeted, as this can be an indication of the ad-hoc all-over-the-place occurrence of this activity, which results in non-structured and cluttered up sequential process models. Note that this approach also often captures the activities that cannot be fitted into the model and hence puts tasks that are connected only when the "All activities connected" option is chosen in Heuristics Miner in D.

Algorithm 4 displays the details of the calculations. It first checks for entropic activities and for this purpose a metric called Activity Entropy ($\overline{AE}$) is proposed. It is calculated as the average of the direct succession (DS) values between an activity and the others in the log where the dependency threshold d is not met (lines 3-4), stored in set AE. In other words, it captures weak dependencies. Procedural activities in a log will have a very low activity entropy, as most of the connections will be either strong ($> d$) or non-existing ($d \approx 0$). Based on a given entropy level threshold $0 \leq e \leq 1$ which is an input parameter of Fusion Miner, a proportion of the log is withheld. The different values $ae_a \in AE$ are ranked and $\lfloor |A_{\mathcal{L}}| \times (1 - e) \rfloor$ activities are kept in the sorted set E (lines 5-6). Furthermore, if there is a gap of $1/e$ between the values $ae_i, ae_j, i, j \in E$, the activities ranked below the gap are removed from E (lines 7-9). This procedure avoids introducing too many activities in D and as a consequence possibly too many rules between them. Note, however, that a fully declarative model can be obtained by using 1 for e. Declare rules mined for single activities (Π_1) are always included in the log.

7.4.2 Pruning and Constraint Choice

After the results of both miners are retrieved, an optional manipulation of the outcome is performed which cuts and adds some constraints, extending the approach proposed by [96]. *(Alternate) precedence* and *response* constraints are transformed into *succession* constraints when the antecedents and consequents involved appear exactly once, and *co-existence* is removed when both activities appear at least once. The *alternate precedence* constraints are reduced to *precedence* when the consequent only appears once, etc. After this pruning phase, a check for transitivity is performed, as, for example, newly introduced *precedence* constraints might be in a transitive relation with other *precedence* constraints, while their superior *alternate precedence* predecessors were not.

Also, since the Declare rule set contains over 20 entries and this set is combined with a procedural model, some constraints become obsolete or less

Algorithm 4 Fusion Miner algorithm

```
Input: $\mathcal{L}, A_\mathcal{L}$                                ▷ The event log $\mathcal{L}$ with activities $A_\mathcal{L}$
Input: $PR_{HM}$                                    ▷ The parameters for Heuristics Miner
Input: $e$                                               ▷ The entropy parameter value
procedure MineMixed-ParadigmModel($\mathcal{L}, A_\mathcal{L}, PR_{HM}, e$)
    $(DG, DS, B2B, LDD) \leftarrow HeuristicsMiner(\mathcal{L}, PR_{HM})$          ▷ Mine dependency info
    for $a \in A_\mathcal{L}$ do
        $ae_a \leftarrow \frac{\sum_{b,a\neq b}^{A_\mathcal{L}} DS(a,b)}{|A|-1}, \forall DS_{ab} < PR_{HM}.dependencyThreshold()$
        $AE \leftarrow ae_a$
    end for
    $sort(AE)$
    $E \leftarrow AE.top(A_\mathcal{L} \times (1-e))$   ▷ Take the $|A_\mathcal{L}| \times (1-e)$ activities with highest $ae$ value
    for $e_i \in E$ do
        if $\frac{ae_{e_i}}{ae_{e_{i+1}}} < \frac{1}{e}$ then                ▷ Stop when gap between entropy values is too big
            $D \leftarrow e_i$                                   ▷ Add $i$ to the set of entropic activities $D$
        elsebreak
        end if
    end for
    $decMap = DeclareMiner(\mathcal{L}, D)$                        ▷ Mine the Declare constraints
    Optional: $decMap = prune(decMap)$          ▷ Optional pruning of the Declare model
    $mixedParadigmNet = MergeModels(decMap, D)$
    return $mixedParadigmNet$
end procedure

procedure MergeModels($DM, D$)
    for $\pi(a, b) \in \Pi$ do
        if $a \in D \wedge b \in D$ then
            $DD \leftarrow \{a \xrightarrow{\pi} b\}$                  ▷ $\rightarrow$ represents a dependency connection,
        else                                                ▷ with $\xrightarrow{\pi}$ a labeled Declare connection.
            if $a \in D$ then
                $DB \leftarrow \{a \xrightarrow{\pi} b\}$
            else
                if $b \in D$ then
                    $BD \leftarrow \{a \xrightarrow{\pi} b\}$
                else
                    $BB \leftarrow \{a \rightarrow b\}$
                end if
            end if
        end if
    end for
    return $DD \cup DB \cup BD \cup BB$
end procedure
```

relevant:

- The *Choice()* constraints are captured by the XOR- and AND-joins and -splits in the procedural model. Furthermore, Declare rules often implicitly invoke such constraints.

- Negative constraints are left out by default, but can be included. These constraint templates tend to introduce a lot of behavior as everything that is not supported is captured by these constraints, but often add little extra insight into the model.

7.4.3 Fitness and Precision of Mixed-Paradigm Models

In order to fully assess the results of Fusion Miner, the fitness and precision of models derived from event logs should be evaluated to show how well the algorithm is capable of not only retrieving, but also narrowing down the behavior as described in the previous sections. Conformance metrics have been studied extensively, and numerous techniques exist. For a comprehensive overview of conformance metrics and techniques, the reader is referred to [169]. Typically, four metrics are considered for evaluating the conformance of a model:

- **Recall/Fitness:** scores how well the model is capable of replaying the events of the log without introducing any deviating procedures, such as skipping events in the log (move-on-log), or in the model (move-on-model),

- **Precision:** scores how accurately the behavior in the log is represented by the model, i.e., if the model allows for any behavior, it is very imprecise and underfitting,

- **Generalization:** scores whether the model still allows for behavior that is not seen in the event log, i.e., if the model only exactly captures the behavior in the log, it is very overfitting and scores low on generalization, and

- **Simplicity:** which scores the model for interpretability and readability.

As previously explained, Fusion Miner tries to fit the event log with different explanatory models. The overall fitness is often high, with a precision score which outperforms techniques that are commonly used to achieve a fitness or recall of 1, such as ILP Miner [164]. Note that, when the Declare model is able to capture most of the flexible behavior which was hard to represent in a procedural net with a high Declare rule support, the model will naturally end up with a high fitness value, oftentimes also reaching a perfect value of 1/100%.

In Chapter 8, a full conformance checking procedure will be presented and elaborated on. However, to illustrate the strengths of Fusion Miner in terms of fitness and precision, an initial conformance checking approach is put foward here to establish an intuition.

First, the model is replayed by using the *Smart Simulation* semantics of CPN Tools. This indicates whether the model is fitting or not, by using a one-step look-ahead strategy. Next, the conformance checking approach based on artificially generated negative events is applied. This technique was proposed by [58] and later improved by [171]. The metric that is used for precision is called Behavioral Precision p_B. The basic setup consists of the generation of artificial negative events, which, simply put, are inserted when there can be no evidence found in the log for their occurrence at a certain trace index. The evidence is based on the prefix of the event, which can be configured to take into account a certain window size. Afterwards, the well-known precision evaluation metric of statistical and data mining models can be applied, and $p_B = \frac{True\,Positives}{True\,Positives + False\,Positives}$.

7.5 Experimental Evaluation

To illustrate the problems that traditional single-paradigm miners face, three event logs are mined and analyzed below to illustrate where improvements can be achieved by Fusion Miner. Some basic metrics regarding the event logs are provided in Table 7.2. The first log contains a straightforward workflow which is distorted by the nearly random occurrence of an activity. The second example deals with a textbook case of a log with different layers of flexibility. The last one is a real-life log, previously used in [169, 176], which contains an incident-management process. The main insights are summarized at the end of this section in a list of contributions the Fusion Miner algorithm offers.

The first two logs are simulated using CPN Tools [181], which is able to execute mixed-paradigm models as indicated above, and recorded by using a technique based on [37]. The logs were checked afterwards for temporary violations of Declare rules to ensure that all constraints were fulfilled at the end of the simulation runs.

Mixed-paradigm models displayed in the text are represented as a combination of a dependency graph and Declare constraints. However, the conversion method that allows to convert dependency graphs into Petri nets, which is available in ProM, is used to retrieve the corresponding Petri net to obtain an executable mixed-paradigm model. The corresponding Petri net is also offered as an output by Fusion Miner. In the examples used in this

Event Log	**Loosely connected activity**	**PhD Process**	**Incident Management Log**
Number of activities	7	6	20
Number of cases	20	30	956
Number of events	786	402	9306
Number of distinct traces	10	30	212

Table 7.2: Basic metrics describing the event log used for the evaluation of Fusion Miner.

paper, the following representation is used in the figures:

- The full arcs represent the procedural behavior as introduced by Heuristics Miner. They form the dependency net of the model.
- The checkered activities exceed the entropy threshold.
- The activities filled with a dark (red) shade fulfill the *exactly1* constraint.
- The activities filled in light gray fulfill the *existence* constraints.
- The striped arrows represent Declare constraints, which are labeled with the template's name.
- The squared activities represent activities fulfilling the *init* and *last* constraints.

7.5.1 Example 1: Process Containing a Loosely Connected Activity

The first log, of which the simulation model can be found in Figure 7.2, represents a standard process with AND-split and -join which is repeated. Procedural discovery algorithms such as Alpha Miner and Heuristics Miner are perfectly capable of retrieving such procedural process behavior. However, introducing an activity which is only roughly tied to a strict position in the workflow such as *Z*, leaves the procedural miners guessing at its location in the process, as explained below. The activity is only connected to activity *C* with a *precedence* constraint, and to activity *D* with an *alternate precedence* constraint.

Issues with Currently Available Miners

Heuristics Miner places *Z* in a separate self-loop before the last activity, as can be seen in Figure 7.3. The Petri net derived from the dependency net, is shown in Figure 7.4, which contains concurrency information and thus more closely represents the original, simulated model. Note, however, that

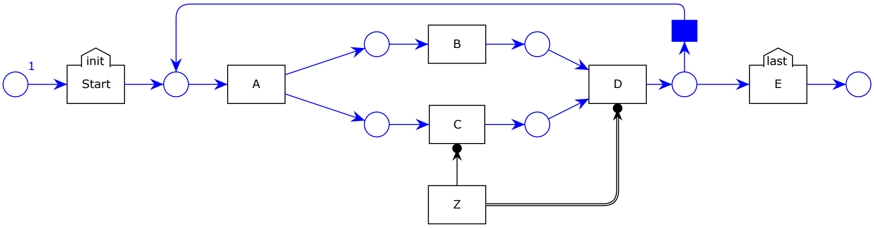


Figure 7.2: A Petri net containing a procedural workflow based around an AND-split and -join, extended with activity Z which can occur in numerous positions in the process. It is connected to activity C with a *precedence* constraint, and to activity D with an *alternate precedence* constraint.

Heuristics Miner is better capable of mining such flexible behavior by the introduction of invisible events. Furthermore, the settings can be changed to, e.g., take into account a lower dependency threshold. This results in a more fitting (and correct) model, but one with very loose workflows that capture all possible connections at any time. Still, the result includes only a very general way of enabling Z in the form of a loop, an outcome that is also seen for Inductive Miner in Figure 7.5.

Mining a log with Declare Miner would result in a correct deriving of temporal relationships, however, the strict control flow that is present between the activities other than Z is very hard to discern as can be seen in Figure 7.6. The overall readability of a declarative process model is one of its main drawbacks, as extensively discussed in the previous chapters.

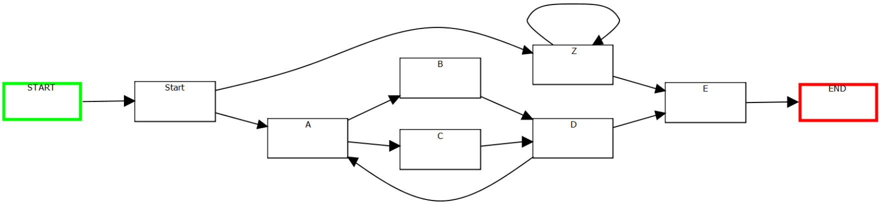


Figure 7.3: Result of Heuristics Miner (default settings) for the log of example 1.

Discovery with Fusion Miner

Fusion Miner is capable of mining a dependency graph and merge it with Declare constraints. This way, it enlightens the unclear sequence information of activity Z, which was not possible in Heuristics Miner and Inductive Miner,

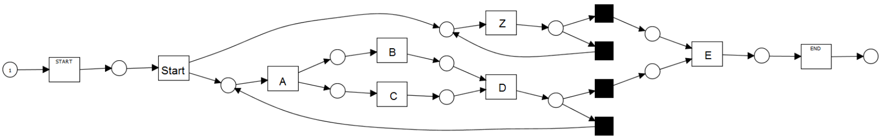


Figure 7.4: Petri net, derived from the dependency net in Figure 7.3.

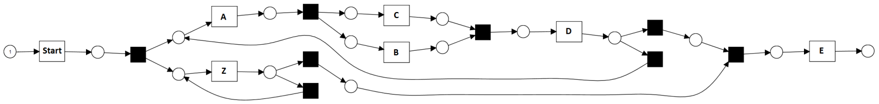


Figure 7.5: The first log mined with Inductive Miner (for a fitness of 100%).

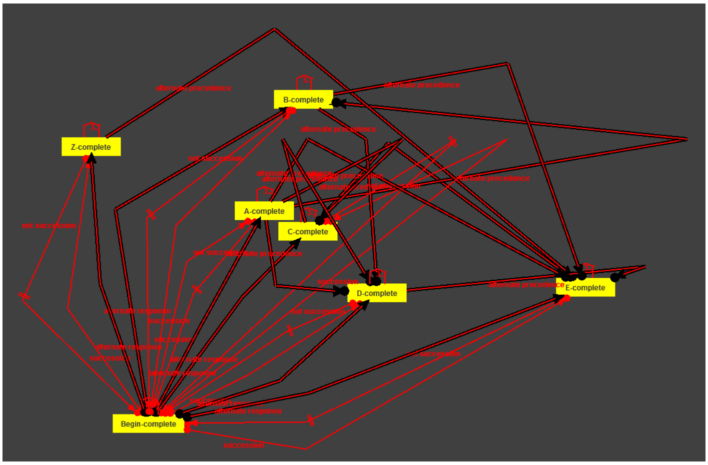


Figure 7.6: Result of Declare Miner for a support of 100 % for the log of example 1. The *not chain succession* constraints are excluded for readability.

and still retain the very procedural behavior of the rest of the net, which was hidden in the Declare model due to the large amount of constraints. Much like the explaining of extra residual value in statistics, Fusion Miner tackles the unknown by fitting the data selectively by using characteristics of both process model types. In this example, the infinite loop enabling activity Z all the time was better explained by introducing Declare constraints with Fusion Miner to reduce the state space of the model around Z, resulting in a more precise model as illustrated in Figure 7.7. The Petri net resulting from the dependency net in the mixed-paradigm model is shown in Figure 7.8.

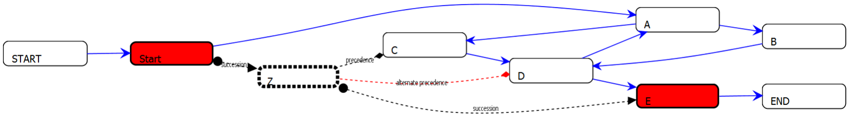


Figure 7.7: Result of Fusion Miner for example 1 with a rule support of 100% and $e = 0.4$.

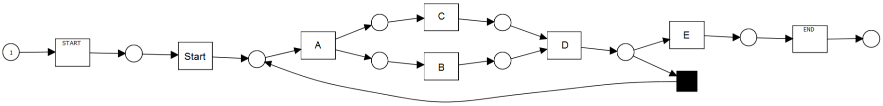


Figure 7.8: Petri net derived from the dependency net in Figure 7.7, containing the procedural behavior in the model.

Fitness and Precision

The results of Heuristics Miner, Inductive Miner, Declare Miner, and Fusion Miner all fit the log 100%. A trace in the log is selected that clearly shows where the result of Fusion Miner improves upon the three other ones. The results for both Heuristics Miner and Fusion Miner are included and can be found in Table 7.3. The cases for which Fusion Miner will be more precise than the outcome of Heuristics Miner are the appearance of Z before C, due to the *precedence* constraint and especially the appearance of Z before D, due to the *alternate precedence* constraint. For example, because Z does not show up until just before the second D, there are less false positives compared to the Petri net. If the behavioral precision scores are scrutinized, the following results are obtained: $p_{B_{HM}} = \frac{18}{32} = 0.5625$ and $p_{B_{FM}} = \frac{18}{30} = 0.6$. A slight increase is achieved by using Fusion Miner, while improving vastly in terms of comprehensibility compared to Declare Miner. Since Z occurs in an almost random fashion, there is not a huge gain in terms of precision, but Fusion Miner is still better capable of representing all the information that can be extracted from the log.

Ind.	**Act.**	**Enabled in MPM**			**Enabled in HM**			**Negative events**					
1	Start	Start			Start			D	E	A	B	C	Z
2	A	Z	A		A	Z		D	E	Start	B	C	
3	Z	Z	B		Z	B	**C**	D	E	Start	A	C	
4	C	Z	**B**	C	Z	**B**	C	D	E	Start	A	B	
5	Z	Z	**B**		Z	**B**		D	E	Start	A	B	C
6	B	**Z**	B		**Z**	B		D	E	Start	A	C	Z
7	Z	Z	**D**		Z	**D**		D	E	Start	A	B	C
8	D	Z	D		Z	D		E	Start	A	B	C	
9	A	**Z**	A	**E**	**Z**	A	**E**	D	E	Start	B	C	Z
10	B	**Z**	B	**C**	**Z**	B	**C**	D	E	Start	A	C	Z
11	C	**Z**	C		**Z**	C		D	E	Start	A	B	Z
12	Z	Z			Z	**D**		D	E	Start	A	B	C
13	D	**Z**	D		**Z**	D		E	Start	A	B	C	Z
14	E	**Z**	E	**A**	**Z**	E	**A**	D	Start	A	B	C	Z

Table 7.3: The precision calculation for model 1 for both mining outcomes, based on artificially generated negative events. The false positives are indicated in **bold**. The mixed-paradigm model clearly enables less activities throughout the replaying of the trace, while still retaining maximal fitness.

7.5.2 Example 2: PhD Process

As another example, a simple mixed-paradigm process model is provided in Figure 7.9. Both Declare constraints and Petri nets are combined to resemble the progress of a PhD student throughout his career, which contains the strict chain of activities containing a first and second seminar followed by a defense. Meanwhile, he/she creates content which is subsequently published in journals or presented at a conference, resembled by the *alternate precedence* constraints. This constraint expresses that both *Journal Paper* and *Conference* can happen after *Content Creation*, and again only after the next occurrence of the *Content Creation* activity. The first seminar cannot happen before a first contribution to a conference and the second seminar has to be preceded by a journal publication. Note that, while the mixed-paradigm model is fairly simple and understandable, the simulated event log presents characteristics that are typically found in real-life logs originating from complex processes.

Issues with Currently Available Miners

If one mines the log with Heuristics Miner, the algorithm is unable to retrieve the exact position and relation of the three activities *Content Creation*, *Conference* and *Journal Paper* as shown in Figure 7.10. While Heuristics Miner captures loops and invisible events to support the quite random appearance of *Content Creation*, it fails to capture the relation of *Journal Paper* and *Second Seminar*. Furthermore, the model is cumbersome to read due to the large number of invisible tasks needed to express the flexible nature of the relations. A similar result is found with the block-structured approach of

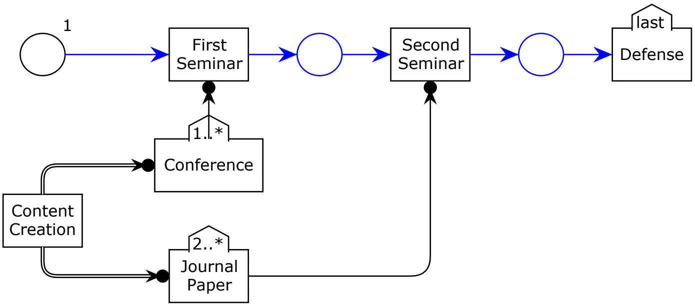


Figure 7.9: Workflow with different layers of flexibility representing the progress of a PhD student.

Inductive Miner, as can be seen in Figure 7.11.

Note that for Heuristics Miner it is possible to use other configurations, e.g., one with a lower dependency threshold. This would result in a model that better captures the behavior in the event log, but this solution would include a very generic model in which every transition can be executed in any order. The result would be similar to the one of Inductive Miner. The algorithm uses many silent transitions to model the arbitrary skipping of transitions. Flexible parts of a log that are not captured (well) by procedural models (as they remained either too restrictive or too general) can be represented with declarative constraints to retrieve them in a more correct and readable way. Although capturing flexible behavior might be possible with procedural models, the sequential information would end up in a very convoluted and unstructured graph of loops, splits and joins, and arrows pointing every direction due to the ad-hoc appearance of activities as can be seen in Figure 7.10. Since most Declare rules represent behavior that can be labeled as non-trivial token games (as was made evident by the conversions in Chapter 3), they are better able to represent such parts of an event log. For example, expressing *alternate response* in a Petri net is a challenging task, leading to the usage of artificial model constructs or different arc types to approximate the same state space.

Again, Declare Miner mines a fitting (for a rule support of 100%) model with low comprehensibility. The result is shown in Figure 7.12.

Discovery of the PhD process with Fusion Miner

Figures 7.13 and 7.14 show the discovered mixed-paradigm models in ProM. An overview of the different activity entropy values can be found in Table 7.4. Even for a small entropy value $e = 0.2$, and due to the $A \times (1-e)$ pruning step and a rule support of 100%, the activity *Content Creation* becomes subject

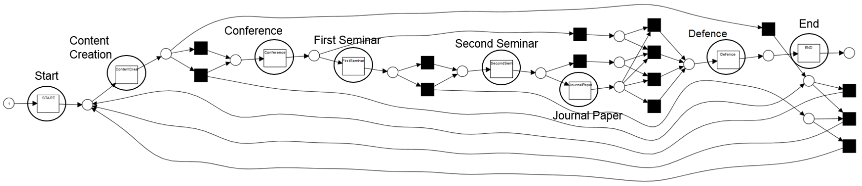


Figure 7.10: Result of Heuristics Miner for log 2 in Figure 7.9 (Default settings and with reduced invisible activities).

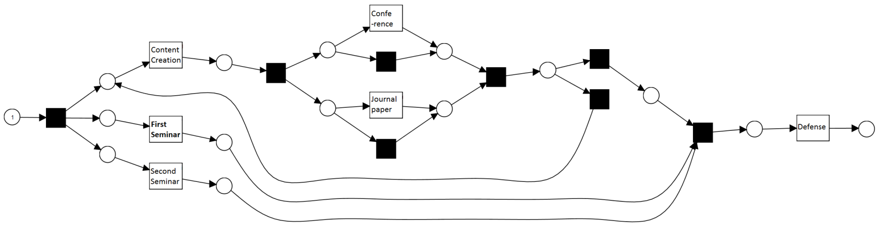


Figure 7.11: Result of Inductive Miner for log 2 (fitness set to 100%).

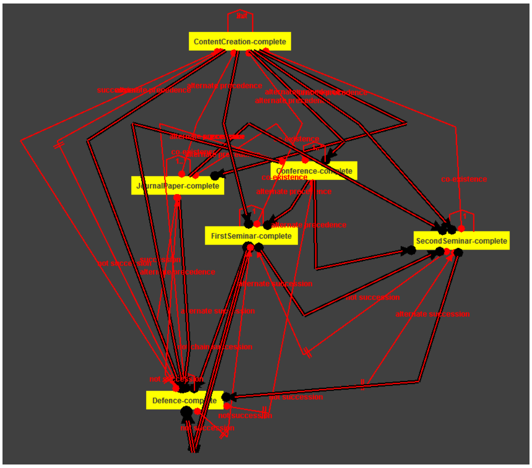


Figure 7.12: Result of Declare Miner for log 2. The rule support used is 100%.

to Declare constraint mining (Figure 7.13). By its constant enabledness it can appear anywhere in the workflow and clutter up the sequential process. By retrieving a few rules for the activity, Fusion Miner is able to represent it in a sense-making way in a mixed model. The model is already capable of capturing the initial model more correctly, as the relationships between

Activity	**Activity Entropy**	**Eligible for e = 0.2**	**Eligible for e = 0.5**
Content Creation (CC)	0.561	Yes	Yes
Journal Paper (JP)	0.557	No	Yes
Conference (CO)	0.52	No	Yes
Second Seminar (SS)	0.38	No	No
First Seminar (FS)	0.242	No	No
Defense (DE)	0	No	No

Table 7.4: The activity entropy values for the PhD example to illustrate the workings of the algorithm. For $e = 0.2$, only *CC* is taken into account, due to the $A \times (1 - e)$ pruning step of AE. For $e = 0.5$, *JP* and *CO* are also included in D.

Content Creation and the other activities are correct. The arc between *Second Seminar* and *Journal Paper* is still incorrect.

By raising the entropy level (Figure 7.14), more activities are added to the declarative set D, in this case *Conference* and *Journal Paper*. This makes sense given the model. Only constrained by the appearance of *Content Creation*, these activities are also rather unpredictable. Note that the procedural part of the model is becoming smaller and smaller, while the Declare constraints offer the same behavior and more.

If this approach is positioned in Figure 7.1, it would be categorized as an attempt to mine behavior more closely by retrieving the intersection of both outcomes, depicted in Figure 7.1b.

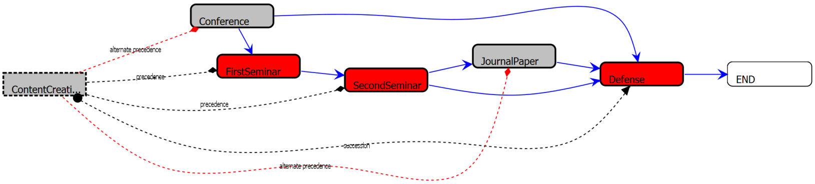


Figure 7.13: Result of Fusion Miner with $e = 0.2$ for log 2. The activity on the left is the only entropic one and is connected with Declare constraints to the other five. The arrow between *Second Seminar* and *Journal Paper* is still incorrect, but the *alternate precedence* constraints are better capable of capturing the behavior in a clear way.

Fitness and Precision

In Table 7.5, the second result (for $e = 0.5$) of Fusion Miner (represented as Declare and Petri net in Figure 7.15) is compared to ILP Miner's result for the PhD example, which can be found in Figure 7.16. ILP Miner is chosen as it generally guarantees a fitness of 1 and does not introduce in-

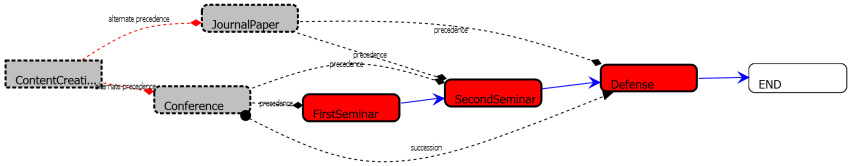


Figure 7.14: Result of Fusion Miner with $e = 0.5$ for log 2. When raising the entropy level in the miner, *Conference* and *Journal Paper* get included in the declarative part of the model. The relations between the activities are now all captured correctly.

visible transitions that blow up the number of possibilities to check for the most fitting path (as in cheapest alignment [4]). Inductive Miner faces this challenge, though it achieves reasonable fitness/precision balance after exploring the combination of invisible transitions that keep the lowest number of transitions enabled throughout the execution. Again, the false positives are indicated in bold. Notice that, even for a dependency measure of 0, Heuristics Miner is unable to derive a perfectly fitting model for the event log and is not used in the comparison.

For this trace in particular, the precision scores are $p_{B_{ILP}} = \frac{22}{13+22} = 0.629$ and $p_{B_{MPM}} = \frac{19}{19+5} = 0.792$ for $e = 0.5$.

As shown in Table 7.5, it is clear that ILP Miner's Petri net keeps a lot more activities enabled throughout the replay of the trace, while Fusion Miner's result enables less activities and thus triggers less false positives, resulting in a higher precision value. Note that, since this trace is short, the prefix window for negative event generation is small, resulting in artificial negative event generation only after some activities, punishing ILP Miner only towards the end. For longer traces, ILP Miner's outcome would produce even more false positives throughout the trace. The outcome of Inductive Miner is comparable. Nevertheless, it is clear that Fusion Miner outperforms traditional procedural mining algorithms especially for logs with different layers of flexibility. Compared to the first example, Fusion Miner is now much better capable of offering better results in terms of fitness/precision/comprehensibility.

The result of Declare Miner, as shown in Figure 7.12, also guarantees a fitness of 1 and a high precision, but the comprehensibility is really low. Furthermore, while Declare models usually aim for high flexibility, the outcome of this model is very strict, surpassing the purpose of Declare. Finally, the precision result of Fusion Miner as depicted in Table 7.5 is the same as the one of Declare Miner, as the resulting Declare model enables the same activities as the mixed-paradigm model during replay.

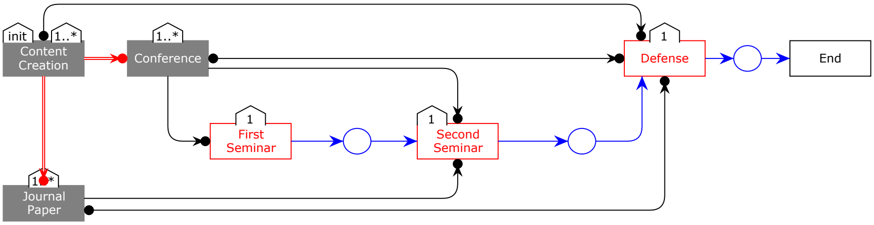


Figure 7.15: Mixed-paradigm model with $e = 0.5$ from Figure 7.14, for which the dependency graph is translated into a Petri net.

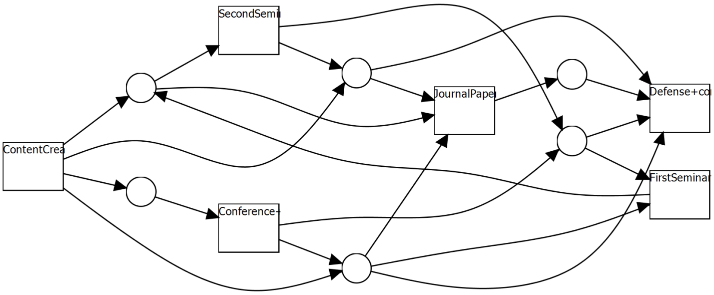


Figure 7.16: Result of ILP Miner for the PhD example log (default settings).

7.5.3 Example 3: Incident Management Process

This real-life example contains different steps of an incident management process, which starts off with the assignment of a help desk resource with *Status Assigned*. This activity can be succeeded by numerous other activities, such as the installment of a *Status Work In Progress* status, a *Pending* status, or the use of a special resource such as *Group GPC CORK* or *Group ORS MS BACKOFFICE*.

Issues with Currently Available Miners

The result of Heuristics Miner with default settings is given in Figure 7.17. The dependency net is large, as it tries to fit most of the log variants, which results in a spaghetti-like model. The result of Declare Miner is not included, as it was unreadable due to the enormous amount of constraints (over 100 constraints for a rule support of 100% and 150 for a support of 75%), which cannot be represented easily in a process map. Furthermore, it did not contain a lot of sequential information, such as *(alternate) precedence/succession*

Ind.	Act.	Enabled ILP						Enabled MPM				Negative events				
1	CC	*CC*						*CC*				CO	JP	FS	SS	DE
2	CO	CC	*CO*	JP	SS			CC	*CO*	JP						
3	JP	CC	*JP*	FS	SS			CC	*JP*	FS		CO				
4	CC	*CC*	FS					*CC*	FS			CO	JP			
5	CO	CC	*CO*	JP	FS	**SS**	DE	CC	*CO*	JP	FS	SS				
6	FS	CC	JP	*FS*	**SS**	**DE**		CC	JP	*FS*		CO	SS	DE		
7	SS	**CC**	**JP**	**FS**	*SS*	**DE**		**CC**	**JP**	*SS*		CC	CO	JP	FS	DE
8	JP	**CC**	*JP*	**FS**	**SS**	**DE**		**CC**	*JP*	**DE**		CC	CO	FS	SS	DE
9	DE	**CC**	**FS**	*DE*				**CC**	*DE*			CC	CO	JP	FS	SS

Table 7.5: The precision calculation for the PhD example for ILP Miner (default settings) and Fusion Miner. As can be seen in the *Enabled* columns, there are a lot of activities enabled all the time in the Petri net of ILP Miner, resulting in lower precision, which is punished by the high false positive rate (negative events indicated in **bold**). Fusion Miner's model enables much less activities while retaining the same fitness value. The abbreviations of the activities can be found in Table 7.4.

constraints, but rather many *choice* and *not chain succession* constraints. Declare Miner oftentimes has a hard time dealing with real-life logs, which is supported by the fact that UnconstrainedMiner (correctly) found numerous such constraints. However, the result of UnconstrainedMiner includes all supported rules, hence not a Declare model for a certain support.

Discovery with Fusion Miner

Fusion Miner depends on Declare Miner's result, which did not include many Declare templates kept for mining in the Fusion Miner algorithm. However, by using various Declare rule support values, it becomes possible to quickly derive various well- and less-supported flows in the dependency graph. The result shown in Figure 7.18, for a support of 75% and $e = 0.5$, shows many activities only executed once. In combination with the *Succession* constraints which yield the obligatory execution of the consequent, the main flow can be read from the figure. This highlights another benefit of using Fusion Miner: scrutinizing the log for frequent and less frequent workflow behavior.

Fitness and Precision

Since the model does not fit the log for 100%, the replaying algorithm cannot enact the model as calculating all the moves-on-model and moves-on-log manually requires a great deal of time and might not yield a good approximation of the best alignment. Through visual inspection, however, it was clear that for certain Declare rule support values, the result was very precise, as the combination of *exactly1* and *succession* constraints clearly delineated the workflow.

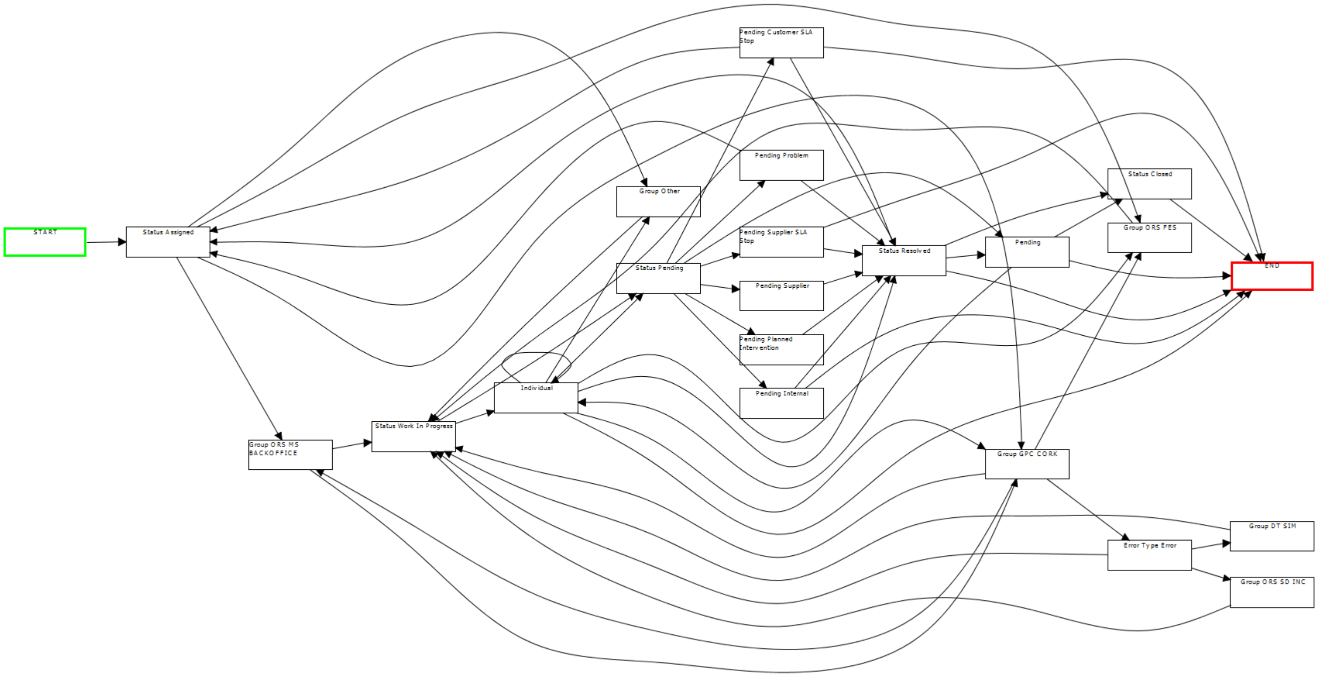


Figure 7.17: The result of Heuristics Miner (default settings) for the real-life log. The model displays most of the process variants, which results in a convoluted model in which a main process flow is hard to discern.

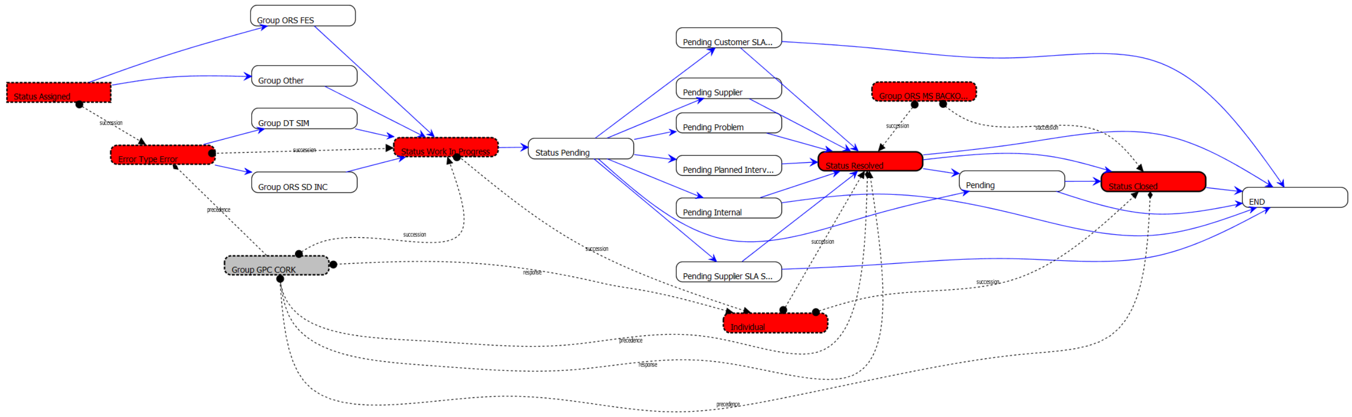


Figure 7.18: The mixed-paradigm result of Fusion Miner for the real-life example with $e = 0.5$ and a rule support of 75%. The main flow is clearly visible through the main Declare constraints, model variants can be deduced from the remaining procedural process constructs.

7.6 Summary

The previous examples have shown the main issues that traditional miners face when dealing with different layers of flexibility. The benefits that can be provided by Fusion Miner and its main use cases are:

- **More precise models:** Fusion Miner provides more precise models because:

  - Declare constraints have different semantics that are better suited to capture flexible behavior. They provide, e.g., richer semantics for loops (e.g. the *alternate* constraints) and duplicate tasks.
  - Declare constraints are derived via rule support on trace level. As such, they are, reinforced by their non-local sequence semantics, better capable of capturing long-distance relationships. Furthermore, they are mined for a certain support level which ensures the presence of the constraint, while many procedural miners use heuristics which cannot guarantee the end result to be a true reflection of the behavior in the event log.

  As shown in Figure 7.1b, the Declare model cuts off parts of the procedural model to make a more precise representation of the log behavior. While there is always a trade-off between precision and generalization, i.e., precise models are not always preferred, Fusion Miner offers parameters that can mitigate overfitting. Most notably the entropy level parameter allows to avoid overfitting flexible parts of a model to avoid introducing too many constraints.

- **More fitting models:** As illustrated in paragraph 7.3, Declare models often capture only the rough part of the workflow, depending on the support level they are granted and the templates that are used. While procedural models such as Petri nets can be relaxed to include more behavior by introducing extra connections and, e.g., invisible activities, resulting in an overly general model, Declare constraints can still capture rough or even very fitting information about the parts of the log that are rather flexible and would result in a very generic procedural model. Figure 7.1c illustrates this as the dark gray mixed model, which represents the combination of a procedural and declarative model, can cover extra parts where the light gray procedural model was not able to go. *By combining both, thus using Declare constraints where flexibility is needed while retaining procedural fragments, Fusion Miner can keep a high fitness value by relaxing procedural models with (precisely) fitting Declare rules.*

– **More comprehensible models:** Model comprehension is influenced by numerous factors [104, 54]. By combining paradigms suited for different granularities of flexibility, mixed-paradigm models are capable of representing processes in the form most suited to their nature. For example, parts of an event log which would typically be captured by a Petri net containing invisible activities, can perhaps better be explained by using some Declare constraints. Furthermore, Declare models can be presented in a simpler way for readers with procedural constructs which offer stricter models where possible. As shown in the previous examples, mined Declare models are often incomprehensible and use a vast amount of rules of which many are redundant.

The basic idea boils down to the exploitation of the stronger semantics of Declare constraints in procedural models in order to introduce more flexibility, while maintaining a procedural part in which strict sequences are still present where the process in the log does not need extra flexibility. Very loose or very procedural logs are not the target of the approach, but can be mined by choosing the correct settings. E.g., by choosing an entropy level e of 0 or 1, one mines a fully procedural or fully declarative process model respectively. As such, Fusion Miner provides an approach to incorporate any level of flexibility, which can also be used for log exploration.

More specifically, Fusion Miner is actually a mining framework rather than a discovery algorithm altogether. As will be illustrated in Chapter 8, Fusion Miner is not limited to using only Heuristics Miner for procedural models, and Declare Miner for declarative models. The algorithm is almost mining technique agnostic, and can cope with other Petri net-mining and Declare-mining algorithms as well.

7.7 Conclusion and Future Work

In this chapter, the challenge of mixed-paradigm process mining was addressed. Fusion Miner was proposed, a framework to mine for procedural and declarative process model constructs simultaneously to represent workflow behavior in an event log. For now, the framework incorporates two of the most widely used process mining algorithms that can be found in literature. Results show that this approach yields more precise models, especially in environments with multiple layers of flexibility, and more comprehensive models, as recapitulated in Section 7.6. The evaluation technique is still preliminary however, and will be further elaborated in Chapter 8, together with a different version of Fusion Miner capable of automatically learning the level of flexibility in a log.

CHAPTER 8

Model and Conformance Checking of Mixed-Paradigm Process Models in a Discovery Context

> *"Truth and honesty*
> *Can be two very different things*
> *But truth can be carelessly confessed*
> *And honestly the truth I do not ask*
> *For fear it's what I'll get."*
>
> — Blake Mills, Three Weeks in Havana

In the previous chapters, approaches for modeling and mining of mixed-paradigm models were constituted by using execution semantics based on Petri nets and Declare. However, there still might be friction between the two models, as modelers and mining algorithms cannot guarantee to have created a deadlock-free model where the Petri net and Declare constraints do not agree on the enabledness of activities and the possibility to reach a final accepting state. This chapter provides a way to offer an unified enactment model that can be used to verify the correctness of the models by checking *(i)* whether the behavior is aligned, and *(ii)* where the model can be improved according to errors that arise along the respective paradigms. The approach is based on intersecting both parts of the model by conjoining their state spaces to obtain a global automaton. Furthermore, the functionality of Fusion Miner is extended in a way to inspect which amount of flexibility is right for the event log, based on whether the models align for a certain entropy level. Finally, a conformance checking algorithm based on alignments is proposed that allows to take any Fusion Miner outcome as an input, i.e., both procedural, declarative, and mixed-paradigm models can be evaluated and compared for fitness, precision, and generalization. All the procedures are demonstrated with an implemented model checker and verified on real-life event logs.

8.1 Introduction

As illustrated in Chapters 6 and 7, mixed-paradigm models consist of a blend of procedural and declarative process models. More precisely, this comprises models which on the one hand contain fixed execution paths, while on the other hand incorporate activity-based rules. Constructing such models is not always straightforward, as one has to be able to grasp the intricacies of both parts, as well as the effect they have on one another. Because both models are mined or modeled separately, though over the same alphabet, many conflicts can occur. Especially for process discovery, a consistency problem regarding the internal behavior can occur. For instance, the procedural model might allow for an activity to be enabled, while the declarative model does not, or vice versa. In this case, the activity has to conform to the most restricting model and become disabled. This might cause deadlocks for the other model later on, where the activity is not enabled or did not enable another activity that is needed to reach the final state(s) of the model.

8.1.1 Motivational Example

Consider the model of Figure 5.10, repeated in Figure 8.1. A procedural Petri net model is combined with multiple Declare constraints. In order to reach a marking containing *p5*, *precedence(g,c)* and *not co-existence(f,g)* cannot reside in the model together, as b requires f to fire first, which means that c can never be executed as g cannot fire anymore and c is in a *precedence* relation with g.

The challenge is to find whether the behavior of both models is compatible and can be used as a whole, and if not, where the discrepancies reside. This might lead to insights into how the model types interact, e.g., the procedural model might be too restrictive to allow for the more flexible behavior of the declarative model. By pinpointing which constraints are not working out with the procedural model, the modeler or miner might react accordingly. This is the principle that is used by the model checking approach proposed in this paper. By incrementally matching both models, the maximal conjoined behavior is sought after, ideally yielding a full match of behavior. Furthermore, Fusion Miner is adopted to also recognize procedural models with a vast state space, i.e., Petri nets for which the reachability graph does not exhibit finite behavior. This reflects the presence of either

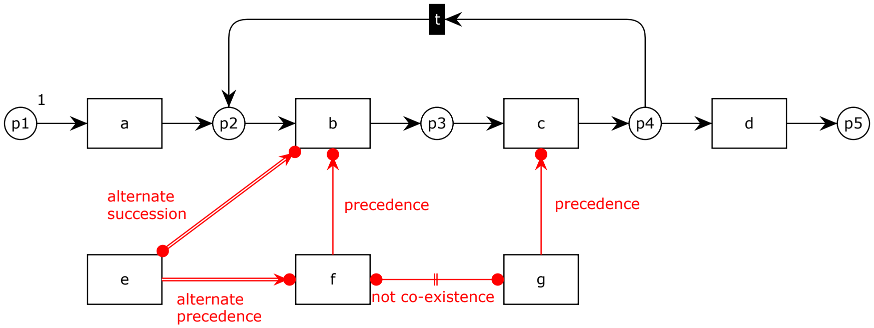


Figure 8.1: An example of a mixed-paradigm model containing inconsistencies.

a vast model, or of a model with many routing constructs for achieving a wide array of execution paths. The latter indicates that the model is either overfitting, or tries to capture a vast deal of the flexibility still, which conflicts with the aim of the approach to capture flexibility with the declarative process model. Hence, the algorithm adapts itself automatically to revise its outcome to shift the balance of the model towards the declarative part. Finally, as a global automaton is provided, it becomes possible to calculate the optimal alignments [4] which can be used to assess the conformance of event logs over mixed-paradigm models.

To achieve the results, Fusion Miner is adapted to use MINERful [22], a very efficient declarative constraint miner, rather than Declare Maps Miner in order to improve performance and to incorporate model checking capabilities. The new version is called FusionMINERful.

This chapter is organized as follows. Section 8.2 introduces the formalisms which are further used to explain the model checking and mining techniques in Section 8.3. Section 8.4 evaluates the approach on a real-life data log, which is followed by an in-depth conformance checking approach in Section 8.5, and the conclusion in Section 8.6.

8.2 Preliminaries

Declare and Petri nets were formalized in Chapter 3. However, some additional concepts are refreshed and put into context of a mixed-paradigm model. First, the alphabet of the Declare model previously defined as Σ is again denoted as Σ_{dec}, for which holds $\Sigma_{dec} = D$. This means the open-world assumption of Declare is not followed [117]. Secondly, the alphabet of the Petri net consists of all transitions, including the invisible λ transitions, $\Sigma_{PN} = \Sigma_T \cup \{T_\lambda\}$. A partial function $\alpha : \Sigma_{PN} \nrightarrow B$ exists, assigning transitions to the activities in the procedural model, with $t \in \Sigma_{PN}$, $t \notin Dom(\alpha)$ the invisible transitions. Note however, that all elements of Σ_{PN} are uniquely distinguishable when constructing the state space, i.e., all invisible transitions are also incorporated in the alphabet Σ_{PN} used in the state space Φ_{PN}. They will also be present in the global automaton Φ_{MPM} in case of conjunction, i.e., $\Phi_{MPM} = \Phi_{DM} \otimes \Phi_{PN}$. The alphabet of the mixed-paradigm model is then $\Sigma_{MPM} = \Sigma_{PN} \cup \Sigma_{dec}$.

8.3 Model Checking Approach

In the following section, the approach to check the consistency of a mixed-paradigm model is described. Thereafter, it is shown that it can be used in the context of mining to reduce the required computational expensiveness.

8.3.1 Model Checking

The model checking approach is based on automaton multiplication, as inspired by [25]. In order to verify whether a mixed-paradigm model does not have any conflicting states, both models are brought to the same execution model, being a finite state automaton. The algorithm starts from a mixed-paradigm model, being a Declare model DM and a Petri net PN possibly retrieved from Fusion Miner, and delivers the final FSA Φ_{MPM}. It is assumed that both models adhere to the activity topology of Table 7.1, i.e., there exist four different activity types, and all activities are subset of the global alphabet A. For the Petri net, the reachability graph is calculated (Algorithm 5, line 2), which is a transition system that can get converted to an FSA. The initial state is determined by the initial marking M_0, and the accepting states by either a given final marking, or a state in which there are no transitions anymore. Next, all the constraints in the declarative model Π are checked for compatibility with the reachability automaton by conjoining its executable form, also an FSA, with the procedural model (line 21). The semantics used for the Declare constraints are the regular expressions from Table 3.1, and are denoted as $\S_{sync}(\pi)$, $\pi \in \Pi$. In case the full declarative model does not conflict, the result Φ_{MPM} will contain the full behavior of both models without the conflicting states. If not, the declarative model with the most non-conflicting constraints will be returned (line 7-11). Another possibility is to check the declarative model up front with the approach discussed in [25] and available in the MINERful framework[1], which also checks for constraint redundancy.

Iteratively conjoining constraints to get the biggest set of non-conflicting Declare constraints might be computationally inefficient in case of very big

[1] `https://github.com/cdc08x/MINERful`

Algorithm 5 Model checking procedure for mixed-paradigm models.

```
Input: $DM$                                                    ▷ Declare model
Input: $PN$                                                    ▷ Petri net model
Output: $\Phi_{MPM}$
procedure calculateModel($DM, PN$)
    $\Phi_{PN} \leftarrow calculateReachabilityGraph(PN)$
    $\Phi_{MPM} = \emptyset$                                   ▷ Start with an empty model
    $b = 0$                    ▷ $b$ is the size of the currently largest constraint set found
    $V \leftarrow \emptyset$                                   ▷ Current constraint set
    for $\pi \in \Pi$ do                      ▷ $\Pi$ is the set of declarative constraints
        $\Phi_{sync}, V \leftarrow checkConstraintForConflicts(\Phi_{PN}, \Pi, \pi, V)$
        if $V = \Pi$ then         ▷ If all constraints are contained in $V$, output model
            $\Phi_{MPM} \leftarrow \Phi_{sync}$
            break
        end if
        if $|V| > b$ then              ▷ If the current constraint set found is larger than
            $b \leftarrow |V|$             ▷ the largest set found so far, save the model
            $\Phi_{MPM} \leftarrow \Phi_{sync}$
        end if
    end for
    return $\Phi_{MPM}$
end procedure

procedure checkConstraintForConflicts($\Phi, \Pi, \pi, V$)
    $V \leftarrow \pi$
    if $\Phi \otimes \S_{regex}(\pi) \neq \emptyset$ then      ▷ Check whether $\pi$ conflicts with $\Phi$
        $\Phi_{res} \leftarrow \Phi \otimes \S_{regex}(\pi)$
        for $g \in \Pi \setminus V$ do
            $checkConstraintForConflicts(\Phi_{PN}, \Pi, g, V)$
        end for
    end if
    return $\Phi_{res}, V$
end procedure
```

constraint sets. This can be resolved by introducing a priority scheme that checks unary and negative constraints (e.g. *exactly2* and *not succession*) last, because they often impose a high degree of interaction with other constraints [143]. Furthermore, this can also help to resolve the issue of choosing one constraint set over the other in the case of equal sizes.

Example

Consider the model in Figure 8.1 again. The reachability graph of the procedural model is relatively small and is presented in Figure 8.2. The initial marking is $M_0(p_1) = 1$. The algorithm iteratively tests whether the constraints in the declarative part of the model can be intersected with this state space. Clearly, *not co-existence(f,g)* and *precedence(f,b)* cannot coexist in this model, as c can never be executed when g is prohibited from firing through *not co-existence(f,g)*. Therefore, one of these constraints is disregarded by the model checker to provide a final execution automaton for the model. According to the priority strategy, this is the case for *not co-existence(f,g)*.

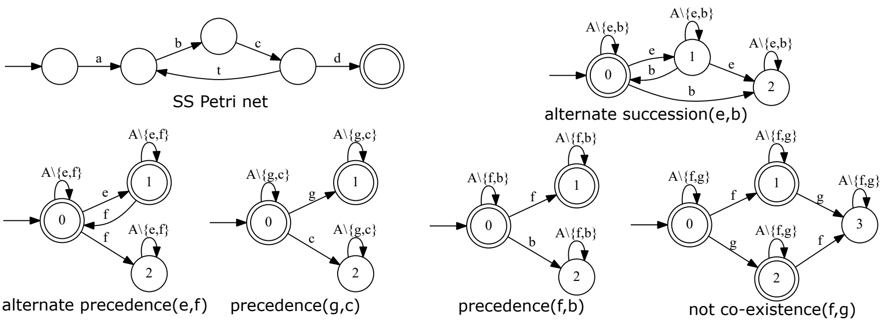


Figure 8.2: The state space of the Petri net and the Declare constraints' automata from Figure 8.1.

8.3.2 Better Mining with Model Checking Iterations

Next to checking models, the algorithm can also be used to guide the discovery of process models in event logs towards a better solution in the following way. Since calculating the reachability graph can be computationally expensive, the algorithm can be adapted as follows. Starting from the initially assigned value, the entropy level e is subsequently raised to increase the size of the declarative part versus the procedural one. In the extreme case, the algorithm resorts to mining solely a declarative model, for which the model checker is guaranteed to finish [25]. Hence, the algorithm actually checks the amount of flexibility that is present in the log and adapts itself accordingly to the amount of 'declarativeness' that is needed. This is called the *self-learning* capability of the approach.

The full details can be found in Algorithm 6. It keeps going while no model is found. First, Fusion Miner is used to mine for a mixed-paradigm model. Note that both DM and PN might be void, in case e is 0 or 1 respectively. In that case, it might be that the output is the state space automaton of either the declarative or procedural model on its own. In case $e = 1$, the same procedure to obtain an as large as possible set of constraint is sought after, for which the algorithm starts from an automaton that accepts the full alphabet A, as an empty one would not accept any conjoined constraints. In order to achieve the self-learning capability, the Fusion Miner algorithm (as documented in 7.4) is adapted to check whether the reachability graph of the procedural model can be calculated, containing less than a certain number of states (Algorithm 6, line 5). This number is calculated based on a threshold n, a model size multiplication coefficient, and the size of the procedural model, i.e., $n \times |S^{PN}| \times 1,000$. If the graph cannot be calculated, the entropy measure is raised by another threshold called *resilience coefficient*, r, to reiterate the process towards with an intent

Algorithm 6 FusionMINERful algorithm.

```
Input: $\mathcal{L}$                                                    ▷ $\mathcal{L}$ an event log
Input: $e$                                                    ▷ Initial entropy level value
Input: $n$                                         ▷ Model size multiplication coefficient
Input: $r$                                                ▷ The resilience measure value
Output: $\Phi_{MPM}$
 1: procedure $calculateModel(\mathcal{L}, e, n, r)$
 2:     $\Phi_{MPM} \leftarrow \emptyset$
 3:     while $\Phi_{MPM} = \emptyset \wedge e \leq 1$ do
 4:         $DM, PN, D \leftarrow FusionMiner(\mathcal{L}, e)$      ▷ Calculate models with FusionMiner
 5:         $\Phi_{PN} \leftarrow calculateReachabilityGraph(PN, n)$      ▷ Returns an empty set after
 6:         $V \leftarrow \emptyset$                                   ▷ $n \times |S| \times 1,000$ states
 7:         $b \leftarrow 0$
 8:         if $\Phi_{PN} \neq \emptyset \vee e = 1$ then
 9:             if $\Phi_{PN} = \emptyset$ then                         ▷ If the Petri net is empty,
10:                 $\Phi_P N = \Phi_A$               ▷ make an automaton that accepts the full alphabet
11:             end if
12:             for $\pi \in \Pi$ do
13:                 $\Phi_{sync}, V \leftarrow checkConstraintForConflicts(\Phi_{PN}, \Pi, \pi, V)$
14:                 if $V = \Pi$ then
15:                     $\Phi_{MPM} \leftarrow \Phi_{sync}$
16:                     break
17:                 end if
18:                 if $|V| > b$ then
19:                     $b \leftarrow |V|$
20:                     $\Phi_{MPM} \leftarrow \Phi_{sync}$
21:                 end if
22:             end for
23:         else
24:             $e \leftarrow e + r$
25:         end if
26:     end while
27:     return $\Phi_{MPM}$
28: end procedure
```

to retrieve a model consisting of a bigger declarative part and a smaller procedural one (Algorithm 6, line 20).

Example

Consider the procedural model produced by Fusion Miner for an entropy level of 0.4, a support of 100%, and a confidence of 50%, depicted in Figure 8.3. The procedural part of the model has to take into account many different ways of enabling d in between the other activities, introducing many silent transitions. Calculating the reachability graph will yield an enormous automaton, requiring computationally expensive conjoining operations. Because the reachability graph calculation is stopped after 32,000 ($2 \times 16 \times 1000$ when $n = 2$) states, the algorithm repeats its main procedure with an entropy level e which is increased by r. For $e = 0.4$ and $r = 0.1$, the resulting model eliminates the need for invisible transitions by removing d from the procedural workflow, as depicted in Figure 8.4.

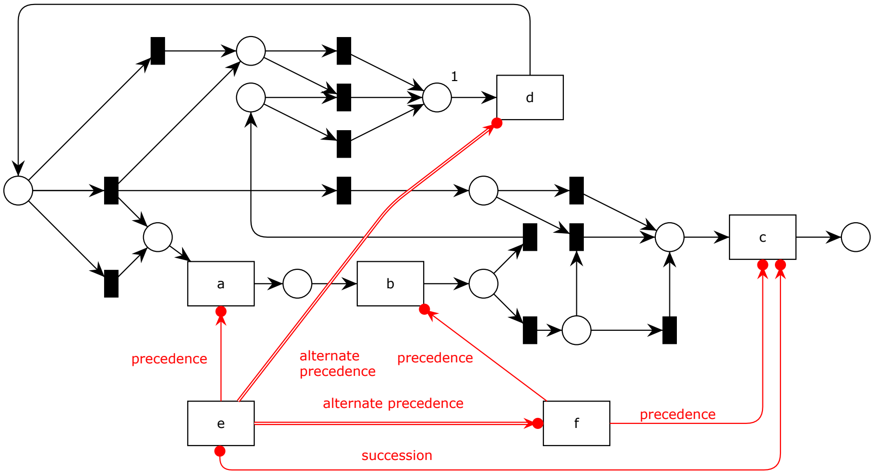


Figure 8.3: Mixed-paradigm output of FusionMINERful for an entropy level of 0.4.

8.4 Implementation and Evaluation

In this section, the implementation in FusionMINERful is introduced. Next, this process mining tool is used to evaluate the approach on the 2012 BPI Challenge [165] event log.

8.4.1 Implementation

Mining a mixed-paradigm model with intertwined state spaces was introduced by Fusion Miner [142]. This mining algorithm uses the notion of

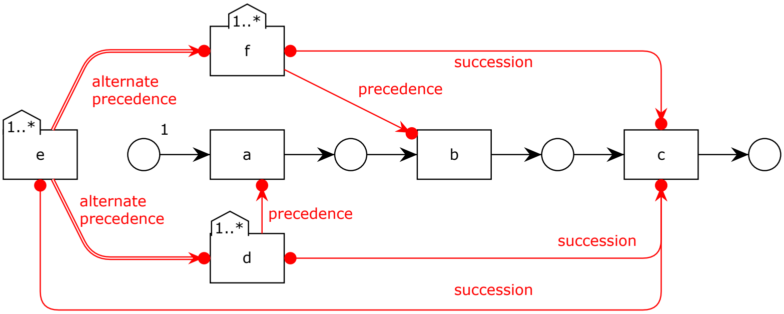


Figure 8.4: Mixed-paradigm output of FusionMINERful for the same event log after the entropy level was raised to 0.5.

entropy to find activity types in the log that do not fit a strict workflow well, based on the dependency information of Heuristics Miner [178]. Activities are divided into D, S, DD, and SS. F_D, F_{DS}, and F_S are mined, while F_{SD} is not considered to avoid too convoluted models that have a high risk of inconsistencies. For this work, a new version called FusionMINERful is used. This algorithm uses MINERful [22] to derive F_D and F_{DS} and Heuristics Miner to mine F_S. It also relies on the state space analysis tools which can be found in ProM. The implementation is compatible with ProM and can be found at `http://www.processmining.be/fusionminerful/`.

The final output model is represented as a dependency graph with Declare constraints [142], which can be converted to a Petri net with Declare constraints. This serves as the basis for the model checking approach. In the output, removed constraints are colored differently, and the implementation also includes the self-learning capability. Blue arcs comprise the procedural model, while black annotated arcs contain Declare constraints. Negative constraints are yellow, while constraints removed during verification are red. Declarative activities use dashed outlines, and gray and red coloring indicates *existence(A,1)* and *exactly(A,1)* respectively.

8.4.2 Application to the 2012 BPIC

The approach of model checking with FusionMINERful has been tested on the 2012 BPI Challenge event log (BPIC 2012) [165]. This log consists of three distinct subprocesses which will be treated separately, in analogy with the approach followed in [3]. The goal is to find out whether the model that is discovered is sound, and how well FusionMINERful can determine the level of flexibility that is needed for mining an informative process model. To test the self-learning capabilities of the algorithm, the initial entropy level is always kept at 0, giving the algorithm the chance to adapt itself according to whether a procedural finite state space can be constructed. The model size multiplicator level was set to 10, and the resilience measure at 0.1.

The first subprocess, X, does not contain any behavior that is too unstructured to handle in a procedural model, hence the algorithm does not raise the entropy level. In this case, the full model of Heuristics Miner is outputted, as can be seen in Figure 8.5. The second subprocess, subprocess Y, shows a low level of flexibility. In this case, two iterations finally churned out the process that can be seen in Figure 8.6. Finally, the last subprocess, Z, reaches the maximal entropy level of 1, indicating that the process can be considered very unstructured. Only the Declare model is outputted, as can be seen in Figure 8.7, which contains many negative constraints and one conflicting constraint which is removed from the model.

Overall, the approach is capable of detecting inconsistencies, although few appeared. This is in line with the intuition established in [25]. Further-

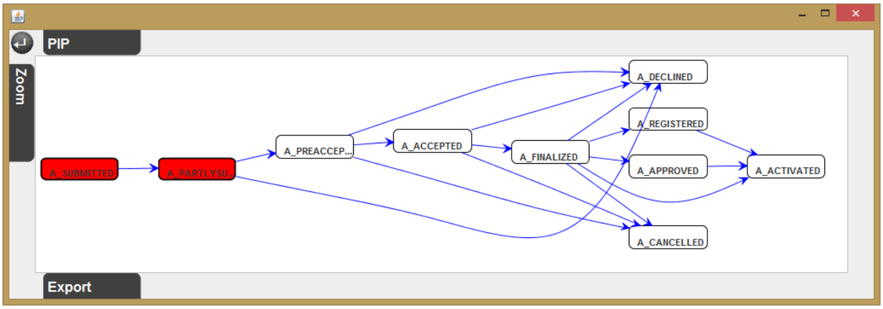


Figure 8.5: Output of FusionMINERful for the X subprocess of BPIC 2012. The set-up is: initial entropy level e of 0, resilience r of 0.1, and model size multiplicator n of 10. The entropy-level remains unchanged, indicating the log to contain procedural behavior.

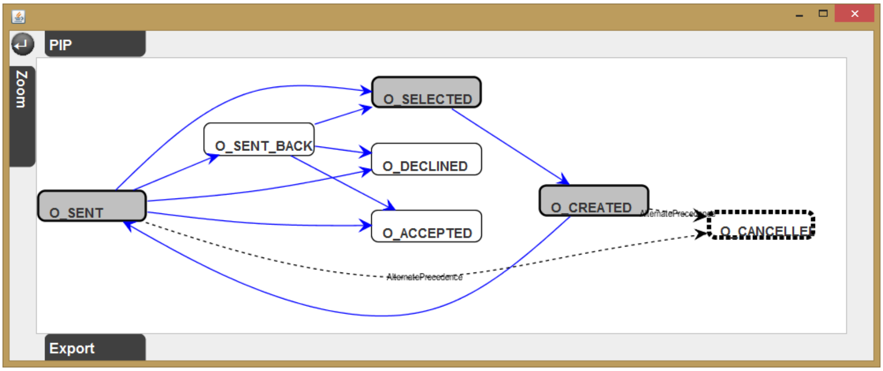


Figure 8.6: Output of FusionMINERful for the Y subprocess of the BPIC 2012. The set-up is: $e = 0$, eventually raised to 0.2, $r = 0.1$, and $n = 10$.

more, FusionMINERful is also capable of finding different levels of flexibility which approaches the results from Heuristics Miner and ILP Miner. These mining techniques also resort to imprecise many-to-many connections to reflect the unstructuredness that is present in the event logs, especially for Y and Z.

8.5 Conformance Checking

In the previous sections it was shown how an executable, consistent mixed-paradigm model can be retrieved both from modeling and mining efforts. This allows for applying conformance checking based on the insights of [4],

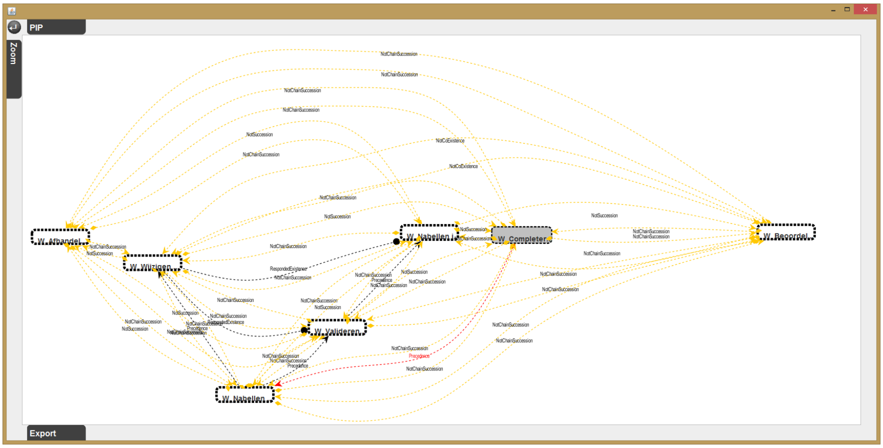


Figure 8.7: Output of FusionMINERful for the Z subprocess of the 2012 BPIC log. The set-up is: $e = 0$, eventually raised to 1.0, $r = 0.1$, and $n = 10$.

and especially the replaying suite of [35], which uses Büchi automata to analyze Declare models. Since the same principles can be applied to FSAs, this section will provide a tailored technique for mixed-paradigm models, and a comparison between the results of both procedural and declarative process mining algorithms.

8.5.1 Alignments

Starting from a mixed-paradigm model with alphabet Σ_{MPM}, an alignment is defined over the alphabets $A_{\mathcal{L}}^{\gg} = A_{\mathcal{L}} \cup \{\gg\}$ the log alphabet, and $\Sigma_{MPM}^{\gg} = \Sigma_{MPM} \cup \{\gg\}$ the model alphabet, where $\gg$ represents the skipping of an event in either the log or model respectively. Hence, there are four types of moves, being pairs of activities, possible when aligning log and model $(x, y) \in A_{\mathcal{L}}^{\gg} \times \Sigma_{MPM}^{\gg}$:

- (x, y) is a *move in log* if $x \in A_{\mathcal{L}}$ and $y = \gg$,
- (x, y) is a *move in model* if $x = \gg$ and $y \in \Sigma_{MPM}$,
- (x, y) is a *synchronous move* if $x \in A_{\mathcal{L}}$ and $y \in \Sigma_{MPM}$,
- (x, y) is an *illegal move* in other cases.

All the legal moves that can be found in an alignment are $A_{leg} = (A_{\mathcal{L}}^{\gg} \times \Sigma_{MPM}^{\gg}) \setminus \{\gg, \gg\}$. An *alignment* of a log and model trace $\sigma^{'}_{\mathcal{L}}$, $\sigma^{"}_{MPM}$ respectively, is a sequence $\gamma = \langle (x^{'}_1, y^{"}_1), ..., (x^{'}_n, y^{"}_n) \rangle \in A_{leg}$. An alignment can be assigned a cost, which is defined by the function $\mathcal{K}(\gamma) = \sum_{(x',y")\in\gamma} \kappa(x^{'}, y^{"})$, where $\kappa : A_{leg} \rightarrow \mathbb{R}_0^+$ the cost of an individual step in the alignment. For the sake of simplicity, it is assumed that:

- $\kappa(x, y) = 0$ when $x = y$,
- $\kappa(x, y) = 0$ when $x =\gg$ and $y \in \Sigma_\lambda$,
- $\kappa(x, y) = 1$ when $x =\gg$ and $y \in \Sigma_{MPM} \setminus \Sigma_\lambda$,
- $\kappa(x, y) = 1$ when $x = A_\mathcal{L}$ and $y =\gg$.

The set of all optimal alignments is denoted $\Gamma = \{\gamma_1, ..., \gamma_n\}$, and the corresponding model alignments $\Gamma_{\Phi_{MPM}} = \{\gamma_1^M, ..., \gamma_n^M\}$ with $\gamma_i^M = \{(x, y) \in \gamma_i \mid y \neq\gg\}$, being the alignments without moves on log. It is assumed that all traces are unique, and that there is no need to adjust for trace type frequencies in the calculations.

There are many different ways to calculate alignments, e.g., A*-based methods for Petri nets [4, 5], and Declare automata [35], which typically lead to an exploration of the complete state space in order to find the most optimal alignments. To illustrate the conformance of the mixed-paradigm models, however, a simple best-first algorithm is used to find the alignments Γ with the lowest cost. The one-alignment strategy is performed, for which not all optimal alignments are stored, but rather the first one with the lowest cost that is encountered. This is done in order to avoid computationally expensive operations that have to travel throughout the full search space. It was shown that in cases where few moves on log and model are present, this does not skew the result in a significant way [6]. Especially in this context, i.e., finding models with mining algorithms that achieve high fitness, it is not unreasonable to make that assumption.

The fitness of a trace $\sigma_\mathcal{L} \in \mathcal{L}$ can then be defined as

$$Fitness(\sigma_\mathcal{L}, \Phi_{MPM}) = 1 - \frac{\mathcal{K}(\gamma_{\sigma_\mathcal{L}})}{\mathcal{K}(\gamma^{ref}_{\sigma_\mathcal{L},\sigma_{MPM}})},$$

where $\gamma^{ref}_{\sigma_\mathcal{L},\sigma_{MPM}}$ is the reference alignment that only contains moves in the log or in the model.

To obtain precision and generalization, an *alignment automaton* $(\Sigma_{aa}, S_{aa}, \sigma_{aa}, \delta_{aa}, F_{aa})$ is constructed as follows:

- $\Sigma_{aa} = \Sigma_{MPM} \cup \gg$ is the alphabet of the automaton,
- S_{aa} is the set of prefixes of all model alignments in $\Gamma_{\Phi_{MPM}}$,
- σ^0_{aa} is the initial state, being an empty sequence,
- $\delta_{aa} : S_{aa} \times \Sigma_{aa} \to S_{aa}$ is a function that connects the prefix states, i.e., it is defined when $\delta_{aa} \otimes \langle s \rangle \in S_{aa} \neq \emptyset$,
- $F_{aa} \subseteq S_{aa}$ is the set of final states.

The automaton represents all unique paths traveled by the alignments throughout the model. They can be leveraged towards precision by pitching the set of possible executions in the model (i.e. enabled activities) after executing a trace σ, $av_{MPM}(\sigma) = \{s \in \Sigma_{aa} \mid \sigma \otimes \langle s \rangle \in \Phi^*_{MPM}\}$ with Φ^*_{MPM} all allowed traces by the model (Kleene operator), compared to the set of activities that have been actually executed for constructing the alignment automaton after replaying σ, $ex_{\mathcal{L}}(\sigma) = \{s \in \Sigma_{aa} \mid \delta(\sigma, s) \neq \emptyset\}$. Hence,

$$Precision(\Phi_{MPM}, \mathcal{L}) = \frac{\sum_{\sigma \in S_{aa} \setminus F_{aa}} |ex_{\mathcal{L}}(\sigma)|}{\sum_{\sigma \in S_{aa} \setminus F_{aa}} |av_{\Phi_{MPM}}(\sigma)|}.$$

For generalization, the same alignment automaton is used. A generalization metric tries to estimate to what extent a model is capable of allowing for other behavior than the one seen in the event log. It is typically difficult to accurately capture generalization, for it is hard to define unseen behavior. However, the following approach is typically used for processes [163, 35]. All the points in the event log where an activity can happen are scrutinized. For this, the set of non-final states in the alignment automaton $\mathcal{X} = S_{aa} \setminus F_{aa}$, and grouped in equivalence classes, for which holds that they represent the same state in the process model. If the automaton of Φ_{MPM} is represented as such, two classes $\sigma', \sigma" \in \mathcal{X}$ are considered equivalent iff $\delta^*_{\Phi_{MPM}}(\sigma^0_{\Phi_{MPM}}, \sigma') = \delta^*_{\Phi_{MPM}}(\sigma^0_{\Phi_{MPM}}, \sigma")$. By comparing the number of times a particular state $\mathcal{X}_i \in \mathcal{X}$ was visited during replay $\omega(\mathcal{X}_i)$, and the number of activities actually executed in that state $\Delta(\mathcal{X}_i)$, the following probability metric can be used to estimate the likelihood of a new activity occurring:

$$pnew(n, m) = \begin{cases} \frac{n(n+1)}{m(m-1)} & \text{if } m \geq n+2 \\ 1 & \text{otherwise} \end{cases}.$$

Generalization is then defined as:

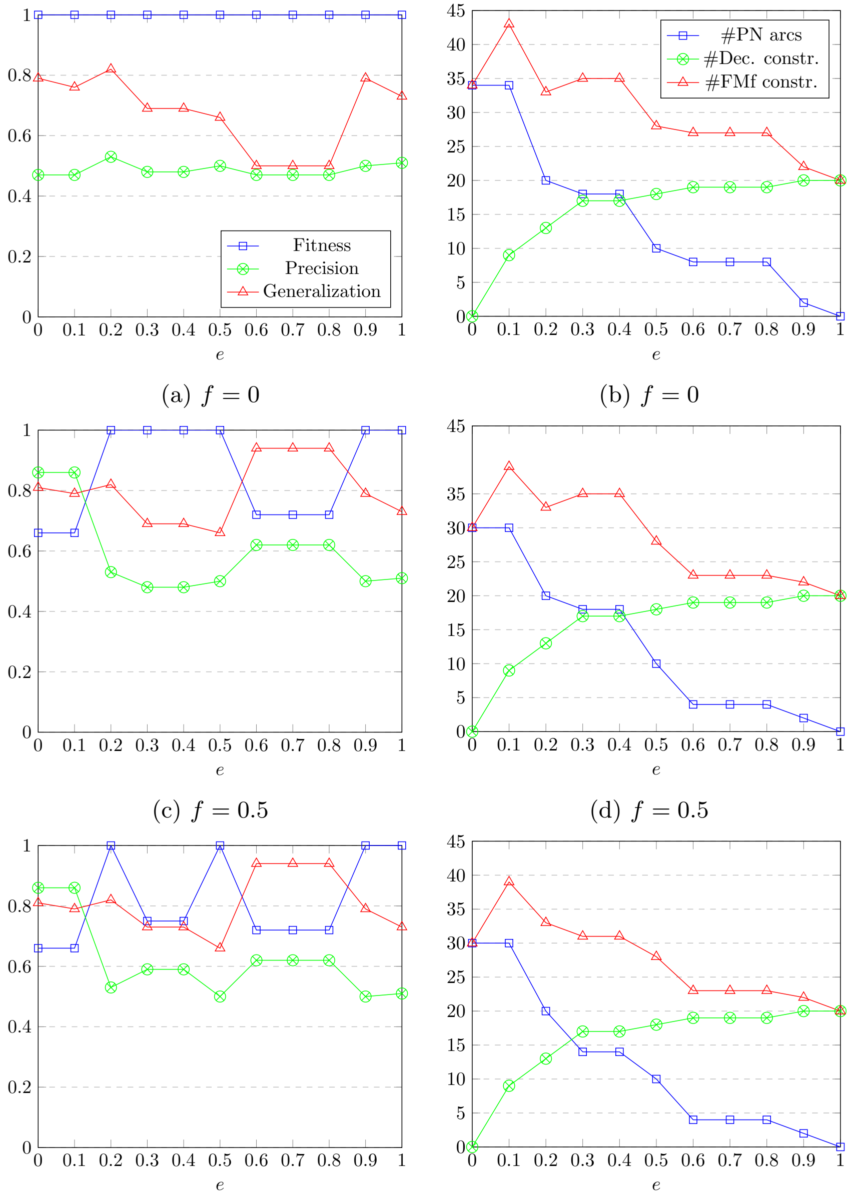
$$Generalization(\Phi_{MPM}, \mathcal{L}) = 1 - \frac{1}{|\mathcal{X}|} \sum_{\chi_i \in \mathcal{X}} pnew(\Delta(\chi_i), \omega(\chi_i)).$$

8.5.2 Experimental Setup

The alignment replay technique will now be applied on three different event logs. First of all, the event logs used for the examples in Section 7.5.1 (called log 1) and Section 7.5.2 are mined (called log 2). Secondly, the 2012 BPI Challenge log is analyzed and replayed according to the strategy outlined in Section 8.5, where a random sample of 1,000 traces was used in order to speed up calculations.

For the mining part, Inductive Miner [91] was used, more specifically the variant tailored towards infrequency, IMf, because it offers a threshold f that allows for regarding infrequent behavior as noise. Initially, the classic setup of Declare Miner and Heuristics Miner was considered, however, Heuristics Miner rarely churns out perfectly fitting models, and often resulted in low fitness scores, especially when paired with Declare Miner. Inductive Miner in this respect is better suited, as it churns out sound, block-structured Petri nets that are better compatible with Declare constraints. Next to f, the entropy e was altered to achieve different blends in FusionMinerFul. The support for mining Declare constraints was kept at 100%. The influence of both the support and confidence for retaining constraints, as well as the inclusion of negative constraints into the declarative mining result were tested without any significant differences in the results. Since FusionMINERful has a built-in checker, constraint sets that do not yield an executable model are disregarded, hence near-maximum support and confidence are typically always achieved. For the tests, a fixed threshold of 80% was used for confidence. The inclusion of negative constraints also does not influence the results, for they are also typically disregarded when the constraint set does not constitute an executable model, and they also often contain superfluous information as explained in 7.4.2. Finally, the following heuristic was used to suppress too long of a calculation time. First, once intermediate alignment results were longer than half of the trace, the algorithm was not allowed to find alignments shorter than that length. Secondly, a maximum of 10 seconds and 30.000 saved states was imposed on the calculations.

The results can be found in the graphs in Figures 8.8 through 8.10. On the left fitness is indicated in blue, precision in green, and generalization in red, and on the right the number of Petri net arcs is indicated in blue, the number of constraints in green, and the number of constructs used by FusionMINERful is indicated in red. On the left hand side of the graphs, the result of Inductive Miner is displayed, while on the right hand side, the result of MINERful is displayed. Everything in between is a produce of FusionMINERful. A detailed overview of the results can be found in Appendix B.



(a) $f = 0$

(b) $f = 0$

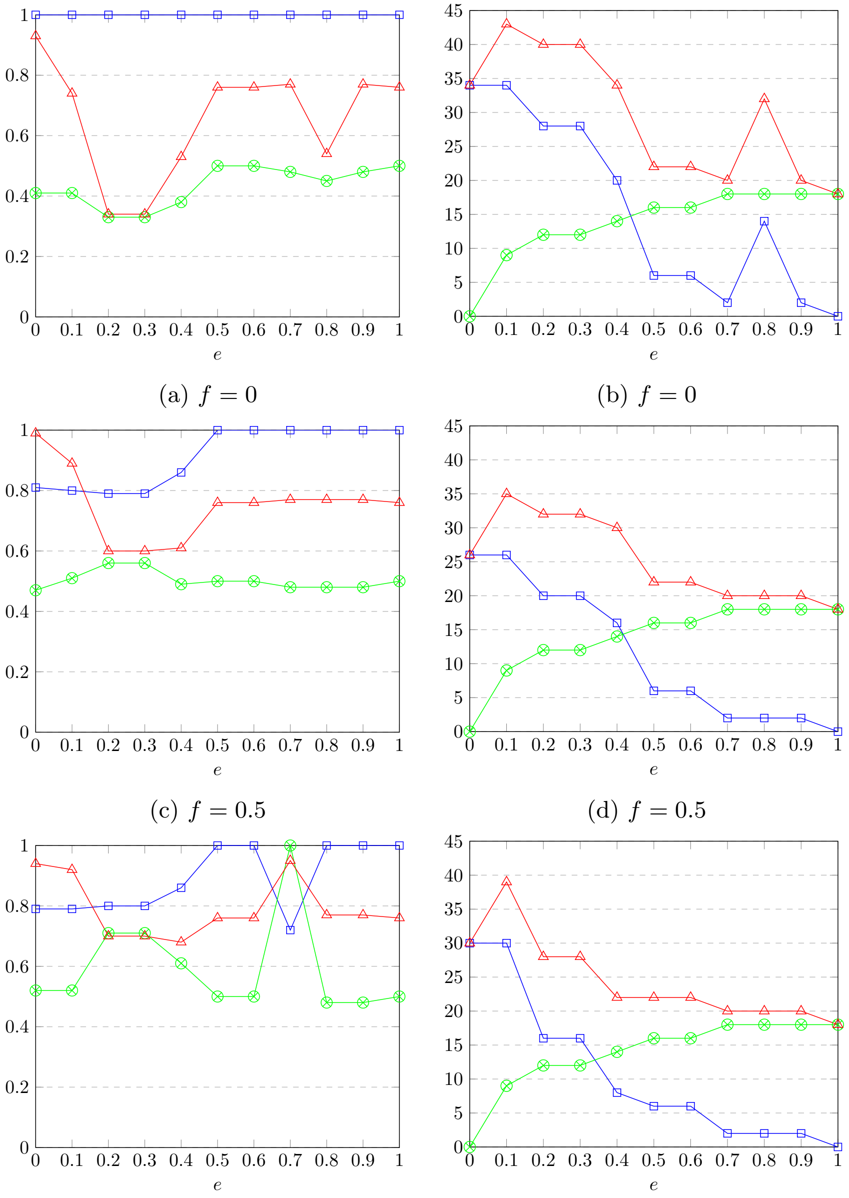
(c) $f = 0.5$

(d) $f = 0.5$

(e) $f = 1.0$

(f) $f = 1.0$

Figure 8.8: Conformance results for log 1.



(a) $f = 0$

(b) $f = 0$

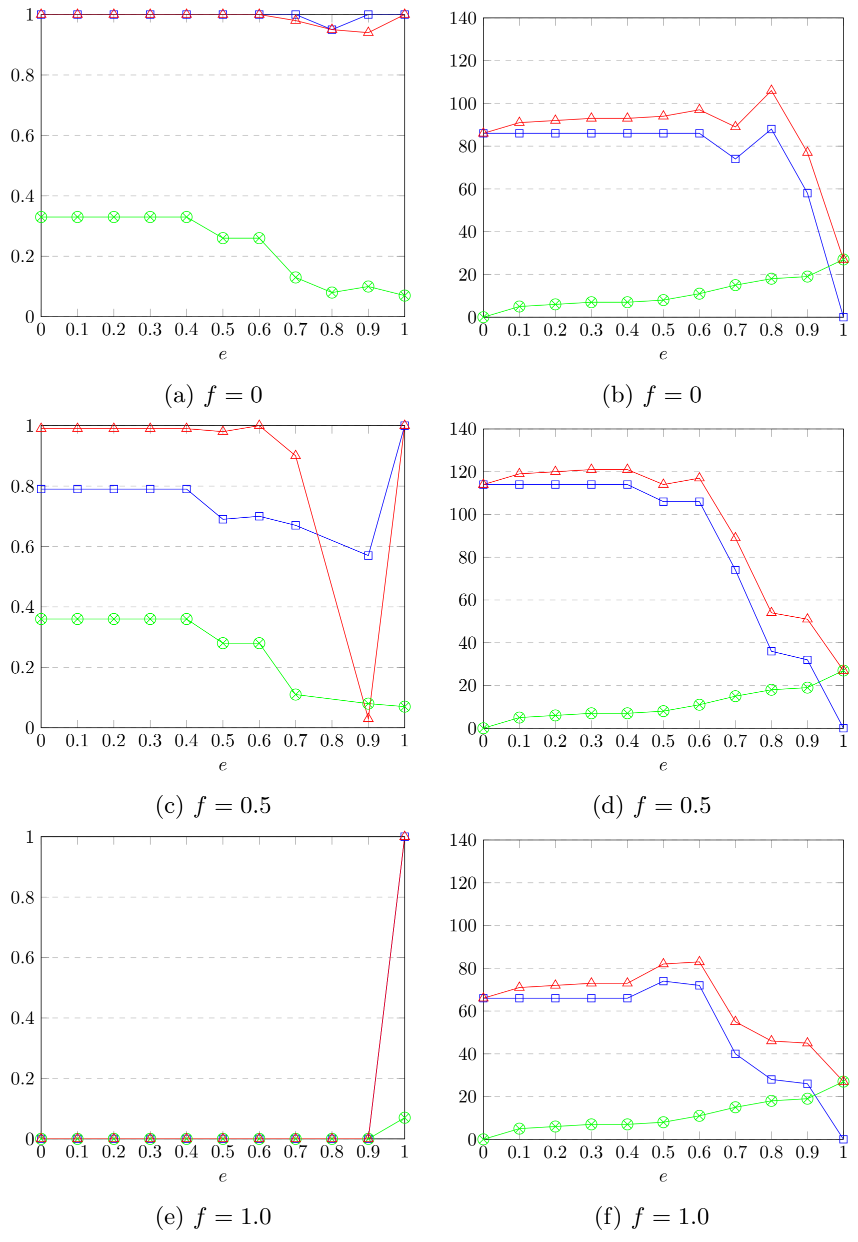
(c) $f = 0.5$

(d) $f = 0.5$

(e) $f = 1.0$

(f) $f = 1.0$

Figure 8.9: Conformance results for log 2.



(a) $f = 0$

(b) $f = 0$

(c) $f = 0.5$

(d) $f = 0.5$

(e) $f = 1.0$

(f) $f = 1.0$

Figure 8.10: Conformance results for the 2012 BPI Challenge log.

8.5.3 Discussion of Results

First of all, this experiment is, to the best of the author's knowledge, the first comparison of conformance results of both procedural and declarative process mining results. Second of all, it also shows in what ways mixed-paradigm solutions can offer different results. Lastly, it also offers extra insights into the conformance results churned out by declarative process miners with more examples and more extensive models than in [35].

In terms of fitness, it is clear that when f is set to 0, Inductive Miner finds a perfectly fitting model. Overall, the difference with the results of MINERful in terms of precision and generalization are small for log 1. In this case, FusionMINERful is able to achieve a slight bump in precision and generalization at an entropy level of 0.2. For log 2, MINERful achieves a slightly higher precision and lower generalization. FusionMINERful approaches the same values, when only a small part of the model is kept procedural as can be seen from the number of Petri net arcs present. For the BPI log, Inductive Miner yields a perfect fitness and generalization, and a low precision score. MINERful clearly underperforms in terms of precision, as can be expected from such a low number of constraints that were found. FusionMINERful achieves the same scores as Inductive Miner for a low entropy level, which retains the procedural model and adds some declarative constraints that do not influence the scores as they do not add any extra information. This is the effect of the low precision of the declarative model.

When the f value is raised, the story changes somewhat. Inductive Miner does not achieve 100% fitness, but also requires fewer Petri net constructs to fit the net except for $f = 0.5$ for the BPI log. For log 1, FusionMINERful does yield 100% fitness and achieves higher precision, and at higher entropy levels approaches MINERful's results which contain the best scores for all three metrics. Similar observations are made for f set to 1. For log 2, MINERful achieves the same precision and generalization scores as Inductive Miner, but for a fitness score of 1 instead of 80%. For higher entropy levels, FusionMINERful not surprisingly also achieves the same scores as MINERful. On the BPI log, Inductive Miner consistently outperforms MINERful, although for a higher level of f the conformance checking algorithm was not capable of replaying the traces within a reasonable amount of time. In this respect, the finite behavior of Declare actually provides a smarter way to mine and evaluate the behavior and the model. FusionMINERful again tries to incorporate best of both worlds, performing insignificantly better or similar to the best of both algorithms.

In the end, it is indeed easier for Inductive Miner to mine the rather procedural behavior of log 1, while MINERful copes better with the behavior of log 2. FusionMINERful achieves results in between the two algorithms. It is also confirmed that the self-learning feature of FusionMINERful achieves the expected level of entropy for all logs. For log 1, a lower entropy level is more appropriate, while for log 2 a higher one constitutes a better blend. Hence, it shows that the entropy level is actually a valid way to approach the level of flexibility that is present in the log. An important takeaway is that MINERful consistently churns out the lowest number of constructs, although Petri nets require at least two arcs for the same connection (as it connects transitions through places). In a way, there is not a huge difference between procedural and declarative process mining algorithms after all, although they are each slightly more inclined to perform well for a level of flexibility which is tailored towards their supporting paradigm. Nevertheless, the intuition established in [35] that declarative process models typically achieve a lower precision cannot be observed directly. On the contrary, since Declare constraints achieve different levels of constraining power in terms of behavior, they even are prone to overfitting, possibly explaining the lower generalization scores. The prime reason higher generalization scores are achieved with Petri nets is the presence of many invisible transitions that constitute a larger number of distinct traces that can be played over the net compared to the finite number of traces generated by FusionMINERful's FSAs. It also explains the timeouts that were witnessed during replay, as a greater number of possibilities needed to be checked upon finding an alignment.

8.6 Conclusion and Future Work

In this chapter, a model checking approach for mixed-paradigm models was proposed based on automaton multiplication. Furthermore, it was shown how this notion can be used in a mining environment to find a fit between procedural and declarative models to achieve models that accurately describe the log with the right level of granularity in terms of flexibility. This was implemented as the FusionMINERful algorithm.

Also, an alignment-based conformance checking algorithm was used in combination with FusionMINERful to compare the strengths and weaknesses of process mining algorithms making use of either the procedural and declarative process paradigm, or both. Although all types of miners achieve good results, they still tend to outperform the other on behavior that is more in line with the underlying paradigm. *FusionMINERful is capable of slightly improving or approaching the best of either algorithm in the form of a mixed-paradigm model.*

This chapter concludes the comparison and intertwined use of both paradigms.

Part IV

Epilogue

CHAPTER 9

Conclusions and Future Work

> *"I'm not saying these eventualities are exactly a picnic, but neither are they a picnic onto which a jumbo jet has accidentally fallen."*
>
> — David Mitchell

The previous chapters have dealt with various aspects of declarative process models and their accompanying subjects of discovery, execution, and verification. Next to that, the comparison and connection with procedural models was made for both modeling and mining.

This chapter aims to summarize the results, and to provide an overview of all future work that can be performed as follow-up studies in succession of this thesis.

9.1 Lessons Learned, Future Work: An Overview

This thesis started off with an in-depth study of the available literature on declarative processes produced by researchers since roughly 2006 in Chapter 2. The ten-year mark is a good time to reflect upon the insights achieved and the progress that was made. Hence, these chapters compared topics, frequently used techniques, languages, data, and applications and condensed them into useful insights. From there, it was pointed out that research up until now has mainly focused on very technical and narrowly-scoped topics, a trend prevalent in the whole field of BPM. The next step in this story should clearly focus on the usefulness of declarative languages in real-life settings. The semantics are defined, many problems were addressed, however, comparing existing declarative and procedural approaches in situations that require flexible, agile processes will greatly improve the value of the current body of work. On a personal note, the incorporation of (a slimmed down version of) the declarative semantics of the likes of Declare and DCR Graphs in CMMN/GSM models, might provide the best future path to investigate. This would serve the best opportunity to achieve a fit with data-aware, and decision modeling approaches.

Next, Chapter 3 introduced all formalities tied to Petri nets, and constraint-based declarative process models. In this chapter, all constraints that currently reside in the body of Declare templates were converted to R/I-net fragments. The benefits are twofold. First of all, it brings the two paradigms to the same process language. This makes it easier to integrate them in mixed-paradigm situations. In case intricate dependencies arise, they can be added by the dependency structures that were discussed in Chapters 4 and 5. Secondly, it shows that, for now, a declarative process model is a process model constituting of constraints, regardless of the way they are formalized. In this regard, even languages that are regarded as very procedural can be used in this way. In the end, flexibility is mainly achieved by modularity and underspecification. Hence, constraints ordered within fixed parts of a model, overarched by a small set of other constraints that provide global guidance to the behavior, are the most tailored towards actually achieving this goal.

The second part of this dissertation focused on understanding and uncovering (hidden) dependencies in constraint-based declarative process models. The main motivation that drives this research is the fact that understanding such models is in general regarded as cognitively challenging. Also, understanding the interrelations of constraints renders a modeler and analyst better capable of assessing the models which are constructed in order to build and verify processes that are too convoluted. In Chapter 4, an algorithm based on a knowledge base of propagations of activity bounds along constraints was proposed, which presents its findings in dependency structures that form an extra layer of annotation to an existing constraint-based model. The various applications were discussed in Chapter 5, which included a large user study that confirmed that the annotations actually relieved the impediment of hidden dependencies to a large extent. Also, formalizing dependency structures enabled the composition of the first cognitive complexity metric that is tailored specifically towards declarative process models. By making use of the different degrees of interrelations that are distinguishable in dependency structures, it is possible to provide a constraint-centric metric which aptly captures the peculiarities that make a model hard to grasp. Finally, an initial approach to refactor declarative process models according to the key execution points that cause hidden dependencies was introduced and an addendum concerning the use of the knowledge base in combination with R/I-nets was reported.

The three latter topics provide a considerable amount of future research tracks to be pursued. First of all, an extensive user study that is not only limited to models which contain hidden dependencies should be used to verify whether the proposed metric can actually reflect the cognitive effort that users face when constructing and reading a declarative process model. Furthermore, it should also be compared to the cyclomatic complexity in order to truly write off this metric as a valid quantification of complexity. Secondly, a fully-fledged and automated version for refactoring constraint-based declarative process models can be constructed. In the end, the impediments of hidden dependencies will then be abolished, and the behavior of the models will become more clear. Finally and foremost, it would be interesting to reevaluate the uncovering algorithm in order to incorporate data dependencies. It is not trivial to include data and especially decisions into processes [81]. As the output of one decision is sometimes directly related to the input of another one, data and decision dependencies also impose an ordering on the activities in a process. It is very hard to keep the decision and process compatible, especially in declarative process models where there exist dependencies that are not always visible on the surface. Nevertheless, it would be a challenging though rewarding task to merge pre- and post-conditions on activities, which are used for incorporating data into declarative process models [110]. The bounds propagation can be used to achieve a better understanding of how data that is included in a declarative process can be interpreted and managed. This would also greatly support techniques such as the combination of Declare and DMN [105].

The final part of this thesis compared the different approaches that are comprised of different blends of the declarative and procedural process paradigms. It was shown what constructs and scenarios are particularly benefiting from a mixed-paradigm approach in Chapter 6, and an automated process discovery algorithm was proposed in the form of Fusion Miner in Chapter 7. A self-learning version of Fusion Miner called FusionMINERful, was later proposed in Chapter 8, which also provided a comparative study of conformance results of both procedural, declarative, and mixed-paradigm models over three different event logs. There indeed exist situations in which one paradigm fits process behavior better than the other, although contrary to previous reporting, it is not the case that declarative process models exhibit low precision due to their flexible nature. Hence, one might regard DPMs as a strong alternative to procedural models in terms of a representation form for process mining results, for they can provide comparable conformance scores and user-friendly textual annotations of (part) of the event log. In this respect, mixed-paradigm models might even be regarded as the ultimate way to leverage the representational bias in process mining [153, 161] by providing both procedural parts and textual descriptions at once.

For the third time, it is worth mentioning that the incorporation of data into processes is a paramount topic for future research. Incorporating data in a mixed-paradigm model would already pose a challenging task, and is perhaps also one of the prime occasions on which using a (Colored) R/I-net for Declare constructs would deliver considerable benefits. Also, introducing data into a mixed-paradigm net might shift the focus of either paradigm, hence requiring an updated version of Figure 6.1. As mentioned earlier, it might be that the data is structured in a procedural way, hence imposing an order on the activities, where the rest of the connections can be patched up together with some overarching Declare rules, or this setup might get reversed as well.

For automated discovery, a lot of ground was covered in terms of conformance checking results. Still, many more factors can be studied. Since Fusion Miner/FusionMINERful is actually a mixed-paradigm framework rather than a fixed mining approach, it can be used with a multitude of Petri net- and Declare model-mining process discovery algorithms. In the latter case, there is no big difference in terms of outcome, rather than in performance, as it is not possible to find a different result for the same set of constraint templates. In the former case, however, there is still a big difference in terms of types of models that can be discovered from an event log. Hence, an even broader experiment can be set up to find the ultimate mixed-paradigm process mining blend. Also, this line of research should aim for calculating all optimal alignments to verify whether the divergence in terms of conformance metrics is not significantly different, as assumed in the current results. Notwithstanding the insights that might be gathered from doing so, this will inevitably lead to very expensive calculations.

9.2 A Final Word

After reading this long and final text, I (hopefully this is the first time in this document you are addressed in first person) hope that you enjoyed reading this thesis and have gained some actual and actionable insights. Now it is up to you. It is always possible to start over. As indicated in Figure 1.5 there is no *last* constraint and the *precedence* relationships leave room for plenty of repetitions. Indeed, those peculiar-looking arcs were not formatted like that by coincidence.

Thank you.

APPENDIX A

User Test Questions

Below, the questions asked during the experiment reported in Chapter 5 are listed. A mix of open and multiple choice questions was used. Participants had to answer the questions regarding the behavior of the model by detailing the possible execution scenarios, while the other, more general questions were offered in a multiple choice fashion with a 5 point range. At the beginning of the sessions, an example model, referred to as 'Model 0', was used to illustrate the tooling and setup of the test.

1. Indicate the session you are in below.

2. In what Master program are you?

3. Do you have any background in process modeling?

4. Open file Model 0 and answer the following question: Can C fire after executing the sequence C-A? Explain.

5. Still for Model 0 answer the following question: Can C fire after executing the sequence A-C? Explain.

6. Open file Model 1 and answer the following question: After firing B as the first activity, which activities are still enabled? Explain.

7. Still for Model 1, answer the following question: After firing E as the first activity, which activities are still enabled? Explain.

8. Still for Model 1, answer the following question: After firing A as the first activity, which activities are still enabled? Explain.

9. Still for Model 1, answer the following question: After firing D as the first activity, which activities are still enabled? Explain.

10. Still for Model 1, answer the following question: Can C ever fire after firing D? Explain.

11. How difficult was it to comprehend the model?

12. How useful was the tool for comprehending the model?

13. Open file Model 2 and answer the following question: After firing B as the first activity, which activities are enabled? Explain.

14. Still for Model 2, answer the following question: After firing A as the first activity, which activities are enabled? Explain.

15. Still for Model 2, answer the following question: After firing D as the first activity, which activities are enabled? Explain.

16. Still for Model 2, answer the following question: After firing A - B - C - D, which activities are enabled? Explain.

17. Still for Model 2, answer the following question: Give at least 5 sequences that can be produced by this model.

18. How difficult was it to comprehend the model?

19. How useful was the tool for comprehending the model?

20. Open file Model 3 and answer the following question: After firing the sequence A - C, which activities are enabled? Explain.

21. Still for Model 3, answer the following question: After firing the sequence F - D - D - C, which activities are enabled? Explain.

22. Still for Model 3, answer the following question: After firing the sequence A - C - D, which activities are enabled? Explain.

23. Still for Model 3, answer the following question: After firing the sequence A - C - B - D, which activities are enabled? Explain.

24. Still for Model 3, answer the following question: After firing the sequence C - C, which activities are enabled? Explain.

25. How difficult was it to comprehend the model?

26. How useful was the tool for comprehending the model?

27. Open file Model 4 and answer the following question: Initially, which activities are enabled? Why?

28. Still for Model 4, answer the following question: After firing the sequence B - E, which activities are enabled? Explain.

29. Still for Model 4, answer the following question: Give an example of a sequence for which all constraints are satisfied. Explain.

30. Still for Model 4, answer the following question: After firing the sequence B - E, how many times can C still fire? Explain.

31. Still for Model 4, answer the following question: After firing the sequence B - E, how many times can D still fire? Explain.

32. How difficult was it to comprehend the model?

33. How useful was the tool for comprehending the model?

34. Open file Model 5 and answer the following question: After firing the sequence I - C, which activities are enabled? Explain.

35. Still for Model 5, answer the following question: After firing the sequence C - C, which activities are enabled? Explain.

36. Still for Model 5, answer the following question: After firing the sequence D - A - C - F, which activities are enabled? Explain.

37. Still for Model 5, answer the following question: After firing D - A - C, can you make C enabled again? Explain.

38. Still for Model 5, answer the following question: After firing the sequence C - C, can you make I enabled again? Explain.

39. How difficult was it to comprehend the model?

40. How useful was the tool for comprehending the model?

41. Overall, how useful was the tool for answering the questions?

42. Overall, how useful was the option to open the constraint descriptions for answering the questions?

43. Overall, how useful was the coloring for answering the questions?

44. Overall, how useful was the option to execute the model and tracking the execution for answering the questions?

45. Overall, how useful were the extra remarks explaining some extra dependencies for answering the questions?

46. Overall, how useful were the extra dependency graphs for answering the questions?

47. How can the tool be expanded in order to help you answer the questions better?

APPENDIX B

Conformance Checking Results

This appendix contains the detailed results of the conformance checking experiments performed in Chapter 8, Section 8.5.

IM (f)	**Entropy (e)**	**Fitness**	**Precision**	**Generalization**	**#PN arcs**	**#Constraints**
0	0	1.000	0.475	0.786	34	0
0	0.1	1.000	0.475	0.759	34	9
0	0.2	1.000	0.533	0.821	20	13
0	0.3	1.000	0.484	0.694	18	17
0	0.4	1.000	0.484	0.694	18	17
0	0.5	1.000	0.505	0.656	10	18
0	0.6	1.000	0.474	0.496	8	19
0	0.7	1.000	0.474	0.496	8	19
0	0.8	1.000	0.474	0.496	8	19
0	0.9	1.000	0.499	0.786	2	20
0	1	1.000	0.508	0.728	0	20
0.5	0	0.665	0.863	0.810	30	0
0.5	0.1	0.665	0.863	0.789	30	9
0.5	0.2	1.000	0.533	0.821	20	13
0.5	0.3	1.000	0.484	0.694	18	17
0.5	0.4	1.000	0.484	0.694	18	17
0.5	0.5	1.000	0.505	0.656	10	18
0.5	0.6	0.719	0.625	0.942	4	19
0.5	0.7	0.719	0.625	0.942	4	19
0.5	0.8	0.719	0.625	0.942	4	19
0.5	0.9	1.000	0.499	0.786	2	20
0.5	1	1.000	0.508	0.728	0	20
1	0	0.665	0.863	0.810	30	0
1	0.1	0.665	0.863	0.789	30	9
1	0.2	1.000	0.533	0.821	20	13
1	0.3	0.749	0.587	0.725	14	17
1	0.4	0.749	0.587	0.725	14	17
1	0.5	1.000	0.505	0.656	10	18
1	0.6	0.719	0.625	0.942	4	19
1	0.7	0.719	0.625	0.942	4	19
1	0.8	0.719	0.625	0.942	4	19
1	0.9	1.000	0.499	0.786	2	20
1	1	1.000	0.508	0.728	0	20

Table B.1: Conformance checking results of FusionMINERful for log 1.

IM (f)	Entropy (e)	Fitness	Precision	Generalization	#PN arcs	#Constraints
0	0	1.000	0.406	0.934	34	0
0	0.1	1.000	0.410	0.742	34	9
0	0.2	1.000	0.327	0.343	28	12
0	0.3	1.000	0.327	0.343	28	12
0	0.4	1.000	0.378	0.526	20	14
0	0.5	1.000	0.504	0.764	6	16
0	0.6	1.000	0.504	0.764	6	16
0	0.7	1.000	0.480	0.775	2	18
0	0.8	1.000	0.450	0.543	14	18
0	0.9	1.000	0.480	0.775	2	18
0	1	1.000	0.504	0.764	0	18
0.5	0	0.807	0.475	0.986	26	0
0.5	0.1	0.801	0.510	0.892	26	9
0.5	0.2	0.788	0.558	0.595	20	12
0.5	0.3	0.788	0.558	0.595	20	12
0.5	0.4	0.856	0.494	0.609	16	14
0.5	0.5	1.000	0.504	0.764	6	16
0.5	0.6	1.000	0.504	0.764	6	16
0.5	0.7	1.000	0.480	0.775	2	18
0.5	0.8	1.000	0.480	0.775	2	18
0.5	0.9	1.000	0.480	0.775	2	18
0.5	1	1.000	0.504	0.764	0	18
1	0	0.786	0.520	0.937	30	0
1	0.1	0.786	0.520	0.922	30	9
1	0.2	0.796	0.713	0.698	16	12
1	0.3	0.796	0.713	0.698	16	12
1	0.4	0.859	0.612	0.684	8	14
1	0.5	1.000	0.504	0.764	6	16
1	0.6	1.000	0.504	0.764	6	16
1	0.7	0.718	1.000	0.951	2	18
1	0.8	1.000	0.480	0.775	2	18
1	0.9	1.000	0.480	0.775	2	18
1	1	1.000	0.504	0.764	0	18

Table B.2: Conformance checking results of FusionMINERful for log 2.

IM (f)	Entropy (e)	Fitness	Precision	Generalization	#PN arcs	#Constraints
0	0	1.000	0.326	1.000	86	0
0	0.1	1.000	0.326	1.000	86	5
0	0.2	1.000	0.326	1.000	86	6
0	0.3	1.000	0.326	1.000	86	7
0	0.4	1.000	0.326	1.000	86	7
0	0.5	1.000	0.258	1.000	86	8
0	0.6	1.000	0.258	1.000	86	11
0	0.7	1.000	0.130	0.984	74	15
0	0.8	0.949	0.084	0.949	88	18
0	0.9	1.000	0.095	0.944	58	19
0	1	1.000	0.071	1.000	0	27
0.5	0	0.794	0.365	0.987	114	0
0.5	0.1	0.794	0.365	0.987	114	5
0.5	0.2	0.792	0.363	0.989	114	6
0.5	0.3	0.789	0.361	0.988	114	7
0.5	0.4	0.792	0.362	0.990	114	7
0.5	0.5	0.693	0.283	0.979	106	8
0.5	0.6	0.697	0.282	0.997	106	11
0.5	0.7	0.669	0.109	0.904	74	15
0.5	0.8	NA	NA	NA	36	18
0.5	0.9	0.571	0.075	0.031	32	19
0.5	1	1.000	0.071	1.000	0	27
1	0	NA	NA	NA	66	0
1	0.1	NA	NA	NA	66	5
1	0.2	NA	NA	NA	66	6
1	0.3	NA	NA	NA	66	7
1	0.4	NA	NA	NA	66	7
1	0.5	NA	NA	NA	74	8
1	0.6	NA	NA	NA	72	11
1	0.7	NA	NA	NA	40	15
1	0.8	NA	NA	NA	28	18
1	0.9	NA	NA	NA	26	19
1	1	1.000	0.071	1.000	0	27

Table B.3: Conformance checking results of FusionMINERful for the 2012 BPI Challenge log.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Michael Adams, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Conferences*, pages 291–308, 2006.

[2] Arya Adriansyah. Replay a Log on Petri Net for Performance/Conformance Plug-in. Technical report, Technische Universiteit Eindhoven, 2012.

[3] Arya Adriansyah and Joos C. A. M. Buijs. Mining process performance from event logs. In *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, pages 217–218, 2012.

[4] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 55–64, 2011.

[5] Arya Adriansyah, Boudewijn F van Dongen, and Wil M P van der Aalst. Memory-efficient alignment of observed and modeled behavior. Technical report, Technische Universiteit Eindhoven, 2013.

[6] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Measuring precision of modeled behavior. *Inf. Syst. E-Business Management*, 13(1):37–67, 2015.

[7] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

[8] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, pages 469–483, 1998.

[9] Irene Barba and Carmelo Del Valle. A constraint-based approach for planning and scheduling repeated activities. In *COPLAS 2011 - Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, pages 55–62, 2011.

[10] Irene Barba, Andreas Lanz, Barbara Weber, Manfred Reichert, and Carmelo Del Valle. Optimized time management for declarative workflows. In *Enterprise,*

Business-Process and Information Systems Modeling - 13th International Conference, BPMDS 2012, 17th International Conference, EMMSAD 2012, and 5th EuroSymposium, held at CAiSE 2012, Gdańsk, Poland, June 25-26, 2012. Proceedings, pages 195–210, 2012.

[11] Irene Barba, Carmelo Del Valle, Barbara Weber, and Andres Jimenez Ramirez. Automatic generation of optimized business process models from constraint-based specifications. *Int. J. Cooperative Inf. Syst.*, 22(2), 2013.

[12] Kimon Batoulis, Andreas Meyer, Ekaterina Bazhenova, Gero Decker, and Mathias Weske. Extracting decision logic from process models. In *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, pages 349–366, 2015.

[13] Ekaterina Bazhenova and Mathias Weske. Deriving decision models from process models by enhanced decision mining. In *Business Process Management Workshops*, volume 256 of *Lecture Notes in Business Information Processing*, pages 444–457. Springer, 2015.

[14] Andrea Burattin, Fabrizio Maria Maggi, Wil M. P. van der Aalst, and Alessandro Sperduti. Techniques for a posteriori analysis of declarative processes. In *16th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2012, Beijing, China, September 10-14, 2012*, pages 41–50, 2012.

[15] S. N. Cant, D. Ross Jeffery, and Brian Henderson-Sellers. A conceptual model of cognitive complexity of elements of the programming process. *Information & Software Technology*, 37(7):351–362, 1995.

[16] Jorge S. Cardoso. Evaluating workflows and web process complexity. *Workflow Handbook*, 2005:284–290, 2005.

[17] Jorge S. Cardoso. Approaches to compute workflow complexity. In *The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006*, 2006.

[18] Massimiliano Cattafi, Evelina Lamma, Fabrizio Riguzzi, and Sergio Storari. Incremental declarative process mining. In *Smart Information and Knowledge Management*, volume 260 of *Studies in Computational Intelligence*, pages 103–127. Springer, 2010.

[19] Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. Exploiting inductive logic programming techniques for declarative process mining. *Trans. Petri Nets and Other Models of Concurrency*, 2: 278–295, 2009.

[20] Michele Chinosi and Alberto Trombetta. BPMN: an introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[21] Claudio Di Ciccio and Massimo Mecella. Studies on the discovery of declarative control flows from error-prone data. In *SIMPDA*, volume 1027 of *CEUR Workshop Proceedings*, pages 31–45. CEUR-WS.org, 2013.

[22] Claudio Di Ciccio and Massimo Mecella. A two-step fast algorithm for the automated discovery of declarative workflows. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, 16-19 April, 2013*, pages 135–142, 2013.

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

[24] Claudio Di Ciccio, Massimo Mecella, and Jan Mendling. The effect of noise on mined declarative constraints. In *Data-Driven Process Discovery and Analysis - Third IFIP WG 2.6, 2.12 International Symposium, SIMPDA 2013, Riva del Garda, Italy, August 30, 2013, Revised Selected Papers*, pages 1–24, 2013.

[25] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. Ensuring model consistency in declarative process discovery. In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 144–159, 2015.

[26] Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. Efficient discovery of target-branched declare constraints. *Inf. Syst.*, 56:258–283, 2016.

[27] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on information and Systems*, 80(3):315–325, 1997.

[28] Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving Petri nets from finite transition systems. *Computers, IEEE Transactions on*, 47(8):859–882, 1998.

[29] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume I*, pages 253–271, 1999.

[30] Elio Damaggio, Richard Hull, and Roman Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Inf. Syst.*, 38(4):561–584, 2013.

[31] Renata Medeiros de Carvalho, Natália C. Silva, Ricardo M. F. Lima, and Márcio Cornélio. Reflex: An efficient graph-based rule engine to execute declarative processes. In *IEEE International Conference on Systems, Man, and Cybernetics, Manchester, SMC 2013, United Kingdom, October 13-16, 2013*, pages 1379–1384, 2013.

[32] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, pages 1027–1033. AAAI Press, 2014.

[33] Giuseppe De Giacomo, Marlon Dumas, Fabrizio Maria Maggi, and Marco Montali. Declarative process modeling in BPMN. In *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, pages 84–100, 2015.

[34] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *BPM*, volume 7481 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2012.

[35] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, 47:258–277, 2015.

[36] Riccardo De Masellis, Fabrizio Maria Maggi, and Marco Montali. Monitoring data-aware business constraints with finite state automata. In *International Conference on Software and Systems Process 2014, ICSSP '14, Nanjing, China - May 26 - 28, 2014*, pages 134–143, 2014.

[37] Ana Karla A de Medeiros and Christian W Günther. Process mining: Using CPN tools to create test logs for mining algorithms. In *Proceedings of the sixth workshop on the practical use of coloured Petri nets and CPN tools (CPN 2005)*, volume 576, 2005.

[38] Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 18–33, 2014.

[39] Søren Debois, Thomas T. Hildebrandt, Morten Marquard, and Tijs Slaats. Hybrid process technologies in the financial sector. In *Proceedings of the Industry Track at the 13th International Conference on Business Process Management 2015 co-located with 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2015.*, pages 107–119, 2015.

[40] Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Concurrency and asynchrony in declarative workflows. In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 72–89, 2015.

[41] Peter J. Denning. A new social contract for research. *Commun. ACM*, 40(2): 132–134, 1997.

[42] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information & Software Technology*, 50(12): 1281–1294, 2008.

[43] Paul Dourish, Jim Holmes, Allan MacLean, Pernille Marqvardsen, and Alex Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In *CSCW*, pages 190–198. ACM, 1996.

[44] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management.* Springer, 2013.

[45] James Durbin and Geoffrey S Watson. Testing for serial correlation in least squares regression: I. *Biometrika*, 37(3/4):409–428, 1950.

[46] Alexandre Duret-Lutz and Denis Poitrenaud. SPOT: an extensible model checking library using transition-based generalized büchi automata. In *MASCOTS*, pages 76–83. IEEE Computer Society, 2004.

[47] Rik Eshuis, Richard Hull, Yutian Sun, and Roman Vaculín. Splitting GSM schemas: A framework for outsourcing of declarative artifact systems. *Inf. Syst.*, 46:157–187, 2014.

[48] Dirk Fahland. Towards analyzing declarative workflows. In *Autonomous and Adaptive Web Services*, volume 07061 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[49] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In *BMMDS/EMMSAD*, volume 29 of *Lecture Notes in Business Information Processing*, pages 353–366. Springer, 2009.

[50] Dirk Fahland, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of maintainability. In *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 477–488. Springer, 2009.

[51] Matthew E Falagas, Eleni I Pitsouni, George A Malietzis, and Georgios Pappas. Comparison of PubMed, Scopus, web of science, and Google scholar: strengths and weaknesses. *The FASEB journal*, 22(2):338–342, 2008.

[52] Kathrin Figl and Ralf Laue. Cognitive complexity in business process modeling. In *CAiSE*, volume 6741 of *Lecture Notes in Computer Science*, pages 452–466. Springer, 2011.

[53] Kathrin Figl and Ralf Laue. Influence factors for local comprehensibility of process models. *Int. J. Hum.-Comput. Stud.*, 82:96–110, 2015.

[54] Kathrin Figl, Jan Recker, and Jan Mendling. A study on the effects of routing symbol design on process model comprehension. *Decision Support Systems*, 54(2): 1104–1118, 2013.

[55] Stijn Goedertier and Jan Vanthienen. Compliant and flexible business processes with business rules. In *BPMDS*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[56] Stijn Goedertier, Raf Haesen, and Jan Vanthienen. EM-BrA2CE v0. 1: A vocabulary and execution model for declarative business process modeling. Technical report, KU Leuven, 2007.

[57] Stijn Goedertier, Christophe Mues, and Jan Vanthienen. Specifying process-aware access control rules in SBVR. In *RuleML*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer, 2007.

[58] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10: 1305–1340, 2009.

[59] Stijn Goedertier, Jan Vanthienen, and Filip Caron. Declarative business process modelling: principles and modelling languages. *Enterprise IS*, 9(2):161–185, 2015.

[60] Thomas R G Green. Cognitive dimensions of notations. *People and computers V*, pages 443–460, 1989.

[61] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007.

[62] Michel Hack. Petri net language. Technical report, 1976.

[63] Cornelia Haisjackl and Stefan Zugal. Investigating differences between graphical and textual declarative process models. In *CAiSE Workshops*, volume 178 of *Lecture Notes in Business Information Processing*, pages 194–206. Springer, 2014.

[64] Cornelia Haisjackl, Stefan Zugal, Pnina Soffer, Irit Hadar, Manfred Reichert, Jakob Pinggera, and Barbara Weber. Making sense of declarative process models: Common strategies and typical pitfalls. In *BMMDS/EMMSAD*, volume 147 of *Lecture Notes in Business Information Processing*, pages 2–17. Springer, 2013.

[65] Cornelia Haisjackl, Irene Barba, Stefan Zugal, Pnina Soffer, Irit Hadar, Manfred Reichert, Jakob Pinggera, and Barbara Weber. Understanding declare models: strategies, pitfalls, empirical results. *Software and System Modeling*, 15(2):325–352, 2016.

[66] Maurice Howard Halstead. *Elements of software science*, volume 7. Elsevier New York, 1977.

[67] Sallie M. Henry and Dennis G. Kafura. Software structure metrics based on information flow. *IEEE Trans. Software Eng.*, 7(5):510–518, 1981.

[68] Nico Herzberg, Kathrin Kirchner, and Mathias Weske. Modeling and monitoring variability in hospital treatments: A scenario using CMMN. In *Business Process Management Workshops*, volume 202 of *Lecture Notes in Business Information Processing*, pages 3–15. Springer, 2014.

[69] Alan R. Hevner. The three cycle view of design science. *Scandinavian J. Inf. Systems*, 19(2), 2007.

[70] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

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

[72] Thomas T. Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *EDOC*, pages 161–170. IEEE Computer Society, 2011.

[73] Thomas T. Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *FSEN*, volume 7141 of *Lecture Notes in Computer Science*, pages 343–350. Springer, 2011.

[74] Thomas T. Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Modular context-sensitive and aspect-oriented processes with dynamic condition response graphs. In *FOAL*, pages 19–24. ACM, 2013.

[75] Thomas T. Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Contracts for cross-organizational workflows as timed dynamic condition response graphs. *J. Log. Algebr. Program.*, 82(5-7):164–185, 2013.

[76] David A Huffman and Others. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[77] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno F. Terry Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *WS-FM*, volume 6551 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2010.

[78] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno F. Terry Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Noi Sukaviriya, and Roman Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS*, pages 51–62. ACM, 2011.

[79] San-Yih Hwang and Wan-Shiou Yang. On the discovery of process models from their instances. *Decision Support Systems*, 34(1):41–57, 2002.

[80] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.

[81] Laurent Janssens, Ekaterina Bazhenova, Johannes De Smedt, Jan Vanthienen, and Marc Denecker. Consistent integration of decision (DMN) and process (BPMN) models. In *CAiSE Forum*, volume 1612 of *CEUR Workshop Proceedings*, pages 121–128. CEUR-WS.org, 2016.

[82] Laurent Janssens, Johannes De Smedt, and Jan Vanthienen. Modeling and enacting enterprise decisions. In *CAiSE Workshops*, volume 249 of *Lecture Notes in Business Information Processing*, pages 169–180. Springer, 2016.

[83] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and CPN tools for modelling and validation of concurrent systems. *STTT*, 9(3-4):213–254, 2007.

[84] T. Capers Jones. Measuring programming quality and productivity. *IBM Systems Journal*, 17(1):39–63, 1978.

[85] Gerhard Knolmayer, Rainer Endl, and Marcel Pfahrer. Modeling processes and workflows by business rules. In *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2000.

[86] Esra Kucukoguz and Jianwen Su. On lifecycle constraints of artifact-centric workflows. In *WS-FM*, volume 6551 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2010.

[87] Akhil Kumar and Wen Yao. Process materialization using templates and rules to design flexible process models. In *RuleML*, volume 5858 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2009.

[88] Vitus Lam. Constraint-based reasoning on declarative process execution with the logics workbench. *Business Proc. Manag. Journal*, 21(3):586–609, 2015.

[89] Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. Inducing declarative logic-based models from labeled traces. In *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 2007.

[90] Yoann Laurent, Reda Bendraou, Souheib Baarir, and Marie-Pierre Gervais. Planning for declarative processes. In *SAC*, pages 1126–1133. ACM, 2014.

[91] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - A constructive approach. In *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2013.

[92] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *BPM*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2011.

[93] Fabrizio Maria Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. User-guided discovery of declarative process models. In *CIDM*, pages 192–199. IEEE, 2011.

[94] Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil M. P. van der Aalst. Runtime verification of ltl-based declarative process models. In *RV*, volume 7186 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011.

[95] Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2012.

[96] Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. A knowledge-based integrated approach for discovering and repairing declare maps. In *CAiSE*, volume 7908 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2013.

[97] Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013.

[98] Fabrizio Maria Maggi, Tijs Slaats, and Hajo A. Reijers. The automated discovery of hybrid processes. In *BPM*, volume 8659 of *Lecture Notes in Computer Science*, pages 392–399. Springer, 2014.

[99] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995.

[100] Mike A. Marin, Hugo Lotriet, and John A. van der Poll. Metrics for the case management modeling and notation (CMMN) specification. In *SAICSIT*, pages 28:1–28:10. ACM, 2015.

[101] Thomas J. McCabe. A complexity measure. *IEEE Trans. Software Eng.*, 2(4): 308–320, 1976.

[102] Jan Mendling, Hajo A. Reijers, and Jorge S. Cardoso. What makes process models understandable? In *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2007.

[103] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. Seven process modeling guidelines (7PMG). *Information & Software Technology*, 52(2):127–136, 2010.

[104] Jan Mendling, Mark Strembeck, and Jan Recker. Factors of process model comprehension - findings from a series of experiments. *Decision Support Systems*, 53(1): 195–206, 2012.

[105] Steven Mertens, Frederik Gailly, and Geert Poels. Enhancing declarative process models with DMN decision logic. In *BMMDS/EMMSAD*, volume 214 of *Lecture Notes in Business Information Processing*, pages 151–165. Springer, 2015.

[106] Anders Møller. dk. brics. automaton–finite-state automata and regular expressions for Java, 2010. Accessed on 09.11.2016.

[107] Marco Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[108] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative specification and verification of service choreographiess. *TWEB*, 4(1), 2010.

[109] Marco Montali, Paolo Torroni, Federico Chesani, Paola Mello, Marco Alberti, and Evelina Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundam. Inform.*, 102(3-4): 325–361, 2010.

[110] Marco Montali, Federico Chesani, Paola Mello, and Fabrizio Maria Maggi. Towards data-aware constraints in declare. In *SAC*, pages 1391–1396. ACM, 2013.

[111] Marco Montali, Fabrizio Maria Maggi, Federico Chesani, Paola Mello, and Wil M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM TIST*, 5(1):17, 2013.

[112] Nataliya Mulyar, Maja Pesic, Wil M. P. van der Aalst, and Mor Peleg. Declarative and procedural approaches for modelling clinical guidelines: Addressing flexibility issues. In *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2007.

[113] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[114] Object Management Group OMG Specification. Business Process Model and Notation version 2.0. Accessed on 09.11.2016.

[115] Object Management Group OMG Specification. Decision Model and Notation, 2015. Accessed on 09.11.2016.

[116] Object Management Group OMG Specification. Semantics of Business Vocabulary and Rules, 2015. Accessed on 09.11.2016.

[117] Maja Pesic. *Constraint-based workflow management systems: shifting control to users*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[118] Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006.

[119] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007.

[120] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 383–394. Springer, 2011.

[121] Artem Polyvyanyy and Mathias Weske. Flexible service systems. In *The Science of Service Systems*, Service Science, pages 73–90. Springer, 2011.

[122] Artem Polyvyanyy, Matthias Weidlich, Raffaele Conforti, Marcello La Rosa, and Arthur H. M. ter Hofstede. The 4c spectrum of fundamental behavioral relations for concurrent systems. In *Petri Nets*, volume 8489 of *Lecture Notes in Computer Science*, pages 210–232. Springer, 2014.

[123] Viara Popova, Dirk Fahland, and Marlon Dumas. Artifact lifecycle discovery. *Int. J. Cooperative Inf. Syst.*, 24(1), 2015.

[124] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. From declarative processes to imperative models. In *SIMPDA*, volume 1293 of *CEUR Workshop Proceedings*, pages 162–173. CEUR-WS.org, 2014.

[125] Jan Recker and Jan Mendling. The state of the art of business process management research as published in the BPM conference - recommendations for progressing the field. *Business & Information Systems Engineering*, 58(1):55–72, 2016.

[126] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. ADEPT workflow management system:. In *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 370–379. Springer, 2003.

[127] Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling-an academic dream or the future for bpm? In *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2013.

[128] Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, 1998. Springer.

[129] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In *ICSOC*, volume 8274 of *Lecture Notes in Computer Science*, pages 389–403. Springer, 2013.

[130] Andreas Rogge-Solti, Wil M. P. van der Aalst, and Mathias Weske. Discovering stochastic petri nets with arbitrary delay distributions from event logs. In *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 15–27. Springer, 2013.

[131] Michael Rosemann, Jan Recker, Marta Indulska, and Peter F. Green. A study of the evolution of the representational capabilities of process modeling grammars. In *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 447–461. Springer, 2006.

[132] J P Royston. An extension of Shapiro and Wilk's W test for normality to large samples. *Applied Statistics*, pages 115–124, 1982.

[133] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[134] Shazia W. Sadiq, Wasim Sadiq, and Maria E. Orlowska. Pockets of flexibility in workflow specification. In *ER*, volume 2224 of *Lecture Notes in Computer Science*, pages 513–526. Springer, 2001.

[135] Shazia Wasim Sadiq, Maria E. Orlowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, 2005.

[136] Laura Sánchez-González, Félix García, Francisco Ruiz, and Mario Piattini Velthuis. Measurement in business processes: a systematic review. *Business Proc. Manag. Journal*, 16(1):114–134, 2010.

[137] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil M. P. van der Aalst. Towards a taxonomy of process flexibility. In *CAiSE Forum*, volume 344 of *CEUR Workshop Proceedings*, pages 81–84. CEUR-WS.org, 2008.

[138] Stefan Schönig and Michael Zeising. The DPIL framework: Tool support for agile and resource-aware business processes. In *BPM (Demos)*, volume 1418 of *CEUR Workshop Proceedings*, pages 125–129. CEUR-WS.org, 2015.

[139] Mateus Ferreira Silva, Fernanda Araujo Baião, and Kate Revoredo. Towards planning scientific experiments through declarative model discovery in provenance data. In *eScience Workshops*, pages 95–98. IEEE, 2014.

[140] Natália C. Silva, César A. L. Oliveira, and Ricardo M. F. Lima. A solution to the state space explosion problem in declarative business process modeling (S). In *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, pages 26–29, 2013.

[141] Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio Maria Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *OTM Conferences*, volume 10033 of *Lecture Notes in Computer Science*, pages 531–551, 2016.

[142] Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. Fusion miner: Process discovery for mixed-paradigm models. *Decision Support Systems*, 77:123–136, 2015.

[143] Johannes De Smedt, Jochen De Weerdt, Estefanía Serral, and Jan Vanthienen. Improving understandability of declarative process models by revealing hidden dependencies. In *Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, pages 83–98, 2016.

[144] Johannes De Smedt, Jochen De Weerdt, Jan Vanthienen, and Geert Poels. Mixed-paradigm process modeling with intertwined state spaces. *Business & Information Systems Engineering*, 58(1):19–29, 2016.

[145] Yutian Sun, Wei Xu, and Jianwen Su. Declarative choreographies for artifacts. In *ICSOC*, volume 7636 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2012.

[146] James Taylor, Alan Fish, Jan Vanthienen, and Paul Vincent. Emerging standards in decision modeling. 2013.

[147] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[148] Roman Vaculín, Richard Hull, Terry Heath, Craig Cochran, Anil Nigam, and Piyawadee Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160. IEEE Computer Society, 2011.

[149] Han van der Aa, Henrik Leopold, Kimon Batoulis, Mathias Weske, and Hajo A. Reijers. Integrated process and decision modeling for data-driven processes. In *Business Process Management Workshops*, volume 256 of *Lecture Notes in Business Information Processing*, pages 405–417. Springer, 2015.

[150] Wil M. P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.

[151] Wil M. P. van der Aalst. Making work flow: On the application of petri nets to business process management. In *ICATPN*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2002.

[152] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[153] Wil M. P. van der Aalst. On the representational bias in process mining. In *WETICE*, pages 2–7. IEEE Computer Society, 2011.

[154] Wil M. P. van der Aalst. What makes a good process model? - lessons learned from process mining. *Software and System Modeling*, 11(4):557–569, 2012.

[155] Wil M P van der Aalst. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013.

[156] Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[157] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.

[158] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9): 1128–1142, 2004.

[159] Wil M. P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, 2005.

[160] Wil M. P. van der Aalst, Michael Adams, Arthur H. M. ter Hofstede, Maja Pesic, and Helen Schonenberg. Flexibility as a service. In *DASFAA Workshops*, volume 5667 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2009.

[161] Wil M. P. van der Aalst, Vladimir Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling*, 9(1):87–111, 2010.

[162] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Causal nets: A modeling language tailored towards process discovery. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 28–42. Springer, 2011.

[163] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[164] Jan Martijn E. M. van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2008.

[165] Boudewijn F van Dongen. Event Log for the BPI Challenge 2012, 2012.

[166] Seppe K. L. M. vanden Broucke. *Advances in Process Mining.* PhD thesis, KU Leuven, 2014.

[167] Seppe K. L. M. vanden Broucke, Filip Caron, Jan Vanthienen, and Bart Baesens. Validating and enhancing declarative business process models based on allowed and non-occurring past behavior. In *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 212–223. Springer, 2013.

[168] Seppe K. L. M. vanden Broucke, Cédric Delvaux, João Freitas, Taisiia Rogova, Jan Vanthienen, and Bart Baesens. Uncovering the relationship between event log characteristics and process discovery techniques. In *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 41–53. Springer, 2013.

[169] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. A comprehensive benchmarking framework (cobefra) for conformance analysis between procedural process models and event logs in prom. In *CIDM*, pages 254–261. IEEE, 2013.

[170] Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Declarative process discovery with evolutionary computing. In *IEEE Congress on Evolutionary Computation*, pages 2412–2419. IEEE, 2014.

[171] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. Determining process model precision and generalization with weighted artificial negative events. *IEEE Trans. Knowl. Data Eng.*, 26(8):1877–1889, 2014.

[172] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.

[173] Jan Vanthienen, Caron, Filip, and Johannes De Smedt. Business rules, decisions and processes: five reflections upon living apart together. In *Proceedings SIGBPS Workshop on Business Processes and Services (BPS'13)*, pages 76–81, 2013.

[174] H. M. W. Verbeek, Moe Thandar Wynn, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. Reduction rules for reset/inhibitor nets. *J. Comput. Syst. Sci.*, 76(2): 125–143, 2010.

[175] Barbara Weber, Shazia Wasim Sadiq, and Manfred Reichert. Beyond rigidity - dynamic process lifecycle support. *Computer Science - R&D*, 23(2):47–65, 2009.

[176] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf. Syst.*, 37(7):654–676, 2012.

[177] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410–429, 2011.

[178] Ton Weijters, Wil M.P. van der Aalst, and Ana Karla A de Medeiros. Process mining with the heuristics miner-algorithm. Technical report, Technische Universiteit Eindhoven, 2006.

[179] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, 2007. ISBN 978-3-540-73521-2.

[180] Michael Westergaard. Better algorithms for analyzing and enacting declarative workflow languages using LTL. In *BPM*, volume 6896 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2011.

[181] Michael Westergaard. CPN tools 4: Multi-formalism and extensibility. In *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2013.

[182] Michael Westergaard and Kristian Bisgaard Lassen. The britney suite animation tool. In *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 431–440. Springer, 2006.

[183] Michael Westergaard and Fabrizio Maria Maggi. Declare: A tool suite for declarative workflow modeling and enactment. In *BPM (Demos)*, volume 820 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[184] Michael Westergaard and Tijs Slaats. Mixing paradigms for more comprehensible models. In *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 283–290. Springer, 2013.

[185] Michael Westergaard, Christian Stahl, and Hajo A Reijers. UnconstrainedMiner: Efficient Discovery of Generalized Declarative Process Models. Technical report, Technische Universiteit Eindhoven, 2013.

[186] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *CHI*, pages 143–146. ACM, 2011.

[187] Stefan Zugal, Jakob Pinggera, and Barbara Weber. Creating declarative process models using test driven modeling suite. In *CAiSE Forum*, volume 734 of *CEUR Workshop Proceedings*, pages 1–8. CEUR-WS.org, 2011.

[188] Stefan Zugal, Jakob Pinggera, and Barbara Weber. The impact of testcases on the maintainability of declarative process models. In *BMMDS/EMMSAD*, volume 81 of *Lecture Notes in Business Information Processing*, pages 163–177. Springer, 2011.

[189] Stefan Zugal, Jakob Pinggera, and Barbara Weber. Toward enhanced life-cycle support for declarative processes. *Journal of Software: Evolution and Process*, 24 (3):285–302, 2012.

[190] Stefan Zugal, Pnina Soffer, Jakob Pinggera, and Barbara Weber. Expressiveness and understandability considerations of hierarchy in declarative business process models. In *BMMDS/EMMSAD*, volume 113 of *Lecture Notes in Business Information Processing*, pages 167–181. Springer, 2012.

[191] Michael zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479. Springer, 2008.

Doctoral dissertations from the Faculty of Economics and Business

A full list of the doctoral dissertations from the Faculty of Economics and Business can be found at:

www.kuleuven.ac.be/doctoraatsverdediging/archief.htm.