

## Understanding non-compliance

***Citation for published version (APA):***

Ramezani Taghiabadi, E. (2017). *Understanding non-compliance*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven.

***Document status and date:***

Published: 16/01/2017

***Document Version:***

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

***Please check the document version of this publication:***

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

***General rights***

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

[www.tue.nl/taverne](http://www.tue.nl/taverne)

***Take down policy***

If you believe that this document breaches copyright please contact us at:

[openaccess@tue.nl](mailto:openaccess@tue.nl)

providing details and we will investigate your claim.

# Understanding Non-compliance

Elham Ramezani Taghiabadi

Copyright © 2016 by E.Ramezani Taghianbadi. All Rights Reserved.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Ramezani Taghiabadi, E.

Understanding Non-compliance / by E.Ramezani Taghianbadi.  
Eindhoven: Technische Universiteit Eindhoven, 2016. Proefschrift.

Cover design by Remco Wetzels

A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-94-028-0487-4

Keywords: Process Mining, Conformance Checking, Compliance Checking, Compliance Diagnostics

The work in this thesis has been sponsored by BOSS project.

Printed by IPSKAMP printing

# Understanding Non-compliance

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op maandag 16 januari 2017 om 16:00 uur

door

Elham Ramezani Taghiabadi

geboren te Esfahan, Iran

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter: prof.dr. J. de Vlieg  
1e promotor: prof.dr.ir. W.M.P. van der Aalst  
co-promotor: dr. D. Fahland  
leden: Univ.-Prof.Dipl.-Math.Dr. S. Rinderle-Ma (Universität Wien)  
prof.dr. S. Sadiq (The University of Queensland)  
dr. M. zur Muehlen (Stevens Institute of Technology)  
prof.dr.ir. H. A. Reijers (VU)  
prof.dr. S. Etalle

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

*To my parents...*



# Summary

## Understanding Non-compliance

This dissertation addresses the challenge of analyzing and improving process compliance, i.e., the ability of an organization to follow legal and contractual constraints in their daily operation. Using the design science research method, we first analyzed different requirements on various phases of a compliance analysis approach including compliance rule elicitation and formalization requirements, requirements on compliance checking techniques and requirements on diagnostics. By understanding requirements from literature and practise, we developed the following compliance analysis techniques, each of which was iteratively evaluated and improved in various case studies.

- Elicit and specify informal compliance rules in terms of compliance patterns without exposing users to technicalities of a specification language. The elicitation and formalization part of our approach bridges the gap between informal constraints and formal rules by providing repositories of configurable templates that capture details of rules as options for commonly-required compliance constraints. These options are configured interactively with end-users, using question trees and natural language.
- Detect violations of different types including violations from control-flow compliance rules (rules constraining the execution of activities and their sequence), temporal, data-aware, and resource aware compliance rules. We check compliance based on computing optimal alignments between



execution recorded in an event log and the “most likely compliant path” according to a formal model of a compliance rule. Such an alignment offers detailed diagnostics compared to current approaches of compliance checking. We are able to provide users diagnostics on violating process instances and activity instance, the rules that have been violated, and the frequency of each type of violation. In case of a violation, we can give diagnostics on what should have happened to make the process compliant.

- Aggregate compliance data and provide different statistics. We developed a compliance database to integrate various sources of compliance data. The data structure of this database provides us the flexibility to explore compliance data from different perspectives and prepare reports based on even complex criteria.
- Analyze the root-causes of violations by leveraging various data analysis techniques. This part of the work focuses on understanding non-compliance using the context of violations. We use association rule mining to detect meaningful relations between violations and their context. The result of this analysis enable business users to identify key insights in the large amount of data. Furthermore, we use classification techniques to differentiate violating and compliant patterns in data. These patterns can be used for predicting possible future violations.

The techniques developed in this thesis implemented within the process mining tool set ProM. Each technique has been evaluated on both synthetic and real-life data. We evaluated all techniques combined in a large case study in collaboration with the Dutch Employee Insurance Agency (UWV). The experiment was conducted using A/B testing involving three business units. Together with UWV, we chose a set of compliance rules and checked them in all the three business units. One of the business units was informed about the results of our analysis and we did not distribute the results to the other two units. The case study continued with a follow-up study. We monitored and compared the changes in compliance level of different business units. The results of the analysis show that using our approach, we can provide detailed diagnostics and enable root-cause analysis to improve compliance. The compliance level w.r.t. some of the compliance rules show up to remarkable improvement in the first group, the same effect is not observed in other business units. We received positive feedback from domain experts about the applicability of the approach.

# Contents

<b>Summary</b>	<b>vii</b>
<b>List of Figures</b>	<b>xvii</b>
<b>List of Tables</b>	<b>xxxix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Adherence to Regulations in Times of Big Data . . . . .	1
1.2 Problem Statement and Research Goal . . . . .	4
1.3 State of the Art in Compliance Management . . . . .	8
1.4 Thesis Approach and Results . . . . .	13
1.5 Thesis Road Map . . . . .	14
<b>2 Preliminaries</b>	<b>17</b>
2.1 Behavior . . . . .	17
2.2 Expressing Classical Event Logs as Behaviors . . . . .	22
2.3 Prescribed Behaviors . . . . .	23
2.3.1 Introduction to Petri nets . . . . .	24
2.3.2 Behavioral Notions: Process Instance Run, Run, and Model	28
2.3.3 Precise Specification of a Particular Model as a (Data-Aware) Petri net . . . . .	31
2.4 Aligning an Event Log to a Model . . . . .	36
2.5 Specifying Prescribed Behaviors from Different Perspectives . . .	43

2.6	Specifying behaviors with (Data-Aware) Petri nets Using an Open World Assumption . . . . .	45
2.7	Concluding Remarks . . . . .	46
<b>3</b>	<b>Requirements for Analyzing Compliance</b>	<b>49</b>
3.1	Dimensions of Compliance Constraints . . . . .	51
3.2	Compliance Analysis Overview . . . . .	52
3.3	Requirements for Compliance Analysis . . . . .	53
3.3.1	Requirements on Compliance Diagnostics . . . . .	53
3.3.2	Requirements for Compliance Checking Techniques . . . . .	60
3.3.3	Requirements for Event Logs as an Input for the Compliance Checking Technique . . . . .	63
3.3.4	Requirements for Elicitation and Formalization of Compliance Constraints . . . . .	63
3.4	Elicit and Formalize Compliance Rules . . . . .	65
3.4.1	Atomic Compliance Constraints Found in Literature . . . . .	65
3.4.2	Basic Considerations about Formal Compliance Rules . . . . .	69
3.4.3	Control-Flow Compliance Rules . . . . .	70
3.4.4	Formalizing Temporal Compliance Rules. . . . .	75
3.4.5	Formalizing Data-Aware and Resource-Aware Compliance Rules . . . . .	76
3.4.6	Atomic Compliance Patterns Vs. a Composite Compliance Model . . . . .	78
3.5	Compliance Checking . . . . .	83
3.5.1	Log Preparation . . . . .	83
3.5.2	Detection of Compliance Violations Using Alignments . . . . .	86
3.5.3	Detecting Control-Flow Compliance Violations Using Alignments . . . . .	87
3.5.4	Detection of Temporal Compliance Violations Using Data-Aware Alignments . . . . .	90
3.5.5	Detection of Data-Aware and Resource-Aware Compliance Violations Using Data-Aware Alignments . . . . .	92
3.5.6	Summary: Detecting Violations in all Dimensions . . . . .	95
3.5.7	Combining Compliance Violation Detection Results . . . . .	98
3.6	Providing an Overview of Diagnostics and Root-Cause Analysis of Violations . . . . .	102
3.7	Concluding Remarks . . . . .	106

---

<b>4</b>	<b>Control-Flow Compliance Checking</b>	<b>107</b>
4.1	Checking Control-Flow Compliance Rules . . . . .	109
4.1.1	Example 1: Precedence . . . . .	110
4.1.2	Parameters for Computing Alignments to Influence Diagnostic Information . . . . .	115
4.1.3	Example 2: Bounded Existence of an Activity . . . . .	116
4.1.4	Example 3: Simultaneous Occurrence of Two Activities . . . . .	119
4.1.5	Principles of Designing Atomic Compliance Patterns . . . . .	125
4.2	Generating the Checking Attribute and Its Values . . . . .	127
4.3	Applying Control-Flow Compliance Checking Using Real-Life Event Logs . . . . .	128
4.3.1	Implementation in ProM . . . . .	128
4.3.2	Case Study Constraints and Results . . . . .	130
4.4	Elicitation of Formal Compliance Rules . . . . .	132
4.4.1	The Gap Between Informal Compliance Rules and Formal Specification . . . . .	133
4.4.2	A Methodology for Eliciting and Configuring Formal Compliance Rules . . . . .	136
4.4.3	Configurable Compliance Pattern Repository . . . . .	138
4.4.4	Question Tree . . . . .	141
4.4.5	Illustrating a Compliance Rule to a Domain Expert. . . . .	143
4.5	Supporting Domain Experts to Specify Compliance Constraints . . . . .	143
4.5.1	Implementation in ProM . . . . .	144
4.5.2	Elicitation of the Case Study Compliance Pattern . . . . .	144
4.6	Control-Flow Compliance Rule Elicitation and Checking of a Composite Compliance Model Versus a Set of Atomic Compliance Patterns . . . . .	147
4.7	Related Work . . . . .	148
4.8	Concluding Remarks . . . . .	152
<b>5</b>	<b>Temporal Compliance Checking</b>	<b>153</b>
5.1	Temporal Compliance Rules . . . . .	156
5.2	Technicalities of Temporal Compliance Checking . . . . .	158
5.3	A Generic Temporal Pattern . . . . .	167
5.4	Specifying a Set of Temporal Rules as Atomic Temporal Patterns or A Composite Compliance Model . . . . .	171
5.5	Log Abstraction for Temporal Checking . . . . .	174
5.6	Diagnostics in Temporal Checking . . . . .	179
5.7	Applying Temporal Compliance Checking on Real-Life Event Logs . . . . .	183

5.7.1	Implementation in ProM . . . . .	183
5.7.2	Case Study Compliance Constraints and Results . . . . .	188
5.8	Related Work . . . . .	191
5.9	Concluding Remarks . . . . .	194
<b>6</b>	<b>Data-Aware and Resource-Aware Compliance Checking</b>	<b>195</b>
6.1	Data-Aware and Resource-Aware Compliance Rules . . . . .	199
6.2	Methodology of Data-Aware and Resource-Aware Compliance Checking . . . . .	200
6.3	Compliance Checking of Rules Restricting Data Attributes . . . . .	201
6.4	Compliance Checking of Rules Restricting Activities When a Certain Data Condition Holds . . . . .	208
6.5	Diagnostics in Data and Resource-Aware Compliance Checking . . . . .	217
6.6	Specifying a Set of Data and Resource-Aware Compliance Rules as Atomic Patterns or as a Composite Compliance Model . . . . .	219
6.7	Case Study Constraints and Results . . . . .	219
6.7.1	Implementation . . . . .	220
6.7.2	Case Study 1: Constraint and Results . . . . .	223
6.7.3	Case Study 2: Constraints and Results . . . . .	224
6.8	Related Work . . . . .	227
6.9	Concluding Remarks . . . . .	229
<b>7</b>	<b>Compliance Diagnostics</b>	<b>231</b>
7.1	Motivating Example . . . . .	234
7.2	Overview of the Approach . . . . .	237
7.3	Compliance Checking . . . . .	237
7.3.1	Prescribed Behavior as a Composite Compliance Model . . . . .	238
7.3.2	Compliance Checking Based on Alignments . . . . .	240
7.4	Violation Statistics . . . . .	242
7.5	Identifying and Ranking Problems . . . . .	247
7.5.1	Generating Lists of Problem Statements . . . . .	248
7.5.2	Ranking Problem Statements by Domain Knowledge . . . . .	254
7.5.3	Ranking Problem Statements by Context Relevance . . . . .	255
7.6	Investigating Specific Problems . . . . .	261
7.6.1	Decision Trees for Compliance Violations . . . . .	261
7.6.2	Building Decision Trees from a Problem Statement . . . . .	263
7.6.3	Using Decision Trees to Analyze the Cause of a Violation . . . . .	264
7.7	Implementation and Case Study Results . . . . .	267
7.7.1	Setup of the Experiment . . . . .	267

---

7.7.2	Obtained Diagnostics . . . . .	270
7.8	Improving the Compliance Framework Based on User feedback . . . . .	273
7.9	Related Work . . . . .	278
7.10	Concluding Remarks . . . . .	279
<b>8</b>	<b>Integrating Compliance Results for a Precise Analysis</b> . . . . .	<b>281</b>
8.1	Integrating Compliance Data Using Compliance Database . . . . .	285
8.1.1	Sources of Compliance Data . . . . .	285
8.1.2	Compliance Database and Its Components . . . . .	289
8.2	Querying Information from the Compliance Database . . . . .	299
8.2.1	Exploring Compliance Data . . . . .	300
8.2.2	Analyzing Sets of Independent Events/Moves . . . . .	303
8.2.3	Analyzing Sets of Dependent Events/Moves . . . . .	304
8.3	Generating Enriched Event Logs . . . . .	304
8.4	Populating the Compliance Database with Relevant Data . . . . .	309
8.5	Proof of Concept . . . . .	309
8.6	Concluding Remarks . . . . .	311
<b>9</b>	<b>Case Study within UWV</b> . . . . .	<b>313</b>
9.1	Process Description . . . . .	314
9.2	Design of the Case Study . . . . .	319
9.2.1	Setup . . . . .	320
9.2.2	Case Study Scope . . . . .	321
9.3	Pilot Study . . . . .	321
9.4	Results and Observations of the First Study . . . . .	326
9.4.1	Results of the First Study: Business Unit 1 . . . . .	326
9.4.2	Results of the First Study for Business Unit 2 and Business Unit 3 . . . . .	331
9.5	Root-Cause Analysis and Dissemination of Results . . . . .	332
9.5.1	Analyzing Process Related Information . . . . .	333
9.5.2	Analyzing Combination of Process Related Attribute Val- ues with Occurrences of Different Violations . . . . .	338
9.5.3	Analyzing Client Related Information . . . . .	343
9.5.4	Analyzing Combinations of Client Related Attribute Val- ues with Occurrences of Different Violations . . . . .	344
9.5.5	Summary of Observations from the First Study . . . . .	344
9.6	Follow-Up Analysis . . . . .	347
9.7	Related Work . . . . .	352
9.8	Conclusions and Lessons Learned . . . . .	358

<b>10 Conclusions and Future Work</b>	<b>361</b>
10.1 Contributions of the Thesis . . . . .	361
10.2 Limitations . . . . .	365
10.3 Open Problems . . . . .	367
<b>A Repository of Control-Flow Compliance Rules</b>	<b>369</b>
A.1 Existence Category . . . . .	371
A.2 Dependent Existence Category . . . . .	372
A.3 Bounded Existence Category . . . . .	379
A.4 Bounded Sequence Category . . . . .	388
A.5 Parallel Category . . . . .	391
A.6 Precedence Category . . . . .	393
A.7 Chain Precedence Category . . . . .	401
A.8 Response Category . . . . .	409
A.9 Chain Response Category . . . . .	415
A.10 Between Category . . . . .	420
<b>B Repository of Temporal Compliance Rules</b>	<b>427</b>
B.1 Pattern Duration . . . . .	429
B.2 Delay Between Instances . . . . .	430
B.3 Validity . . . . .	430
B.4 Time Restricted Existence . . . . .	431
B.5 Repetition Time Constraint . . . . .	433
B.6 Time Dependent Variability . . . . .	434
B.7 Overlap . . . . .	434
<b>C Repository of Configurable Control-Flow Compliance Rules</b>	<b>437</b>
C.1 Existence . . . . .	438
C.2 Sequence . . . . .	441
C.3 Chain Precedence and Response . . . . .	445
C.4 Parallel and Between . . . . .	448
C.5 Dependent Existence . . . . .	452
<b>D Results of the Case Study within UWV</b>	<b>455</b>
D.1 Analyzing Process Related Association Rules . . . . .	455
D.2 Analyzing Client Related Association Rules . . . . .	460
<b>Bibliography</b>	<b>465</b>
<b>Acknowledgments</b>	<b>487</b>

<b>Curriculum Vitae</b>	<b>491</b>
<b>SIKS dissertations</b>	<b>493</b>





# List of Figures

1.1	Omnipresence of data allows us to leverage process mining and data mining techniques for data-driven analysis of compliance. . . . .	2
1.2	Life Inc. requires an integrated compliance solution which is generic, flexible, automated and provides detailed diagnostics. . . . .	4
1.3	Compliance Management life cycle. . . . .	8
1.4	The thesis road map gives the mapping between chapters of the thesis onto our compliance analysis approach. . . . .	12
2.1	Example of observed behavior recorded in an event log. . . . .	18
2.2	Behavior $B = (E, \#, \leq)$ consisting of events related to two process instances $p_1$ and $p_2$ . . . . .	20
2.3	Behavior $B' = (E', \#, \leq)$ consisting of events related to two process instances $p_1$ and $p_2$ . . . . .	21
2.4	Tree-view representation of the event log $B$ consisting of two process instances $p_1$ and $p_2$ . . . . .	23
2.5	The loan process modelled as a Petri net. . . . .	25
2.6	A data-aware Petri net modelling the control-flow and data perspective of the loan process. . . . .	27
2.7	Events in run $R_1$ are partially ordered. . . . .	29
2.8	Events in run $R_2$ . . . . .	30
2.9	The data-aware Petri net $DPN$ specifying four firing sequences $p_1, p_2, p_3,$ and $p_4$ . . . . .	32
2.10	Prescribed behaviors specified by $DPN$ . . . . .	33

2.11	The data-aware Petri net $DPN'$ specifying $R_1''$ , and $R_2''$ . . . . .	35
2.12	Prescribed behaviors specified by $DPN'$ . . . . .	36
2.13	The partially ordered alignment $A$ (c) between the event log $L$ (a) and the process model run $R_1$ (b). . . . .	39
2.14	The partially ordered alignment $A'$ (c) between the event log $L$ (a) and the process model run $R_1$ (b). . . . .	41
2.15	Alignment between the event log $L$ and run $R_1''$ shown in tabular form. . . . .	42
2.16	$DPN^{resource}$ : The data-aware Petri net modeling values of attribute <i>resource</i> as transition labels. . . . .	44
2.17	$DPN^{generic}$ : The data-aware Petri net can replay $R_1$ , $R_2$ and many more runs as long as they adhere to the constraints. . . . .	45
3.1	Thesis road map gives the mapping of the sections in Chap. 3 on to our compliance analysis approach. . . . .	50
3.2	Compliance constraints framework. . . . .	51
3.3	Compliance analysis overview. . . . .	53
3.4	Diagnostics about violations of the constraint $C_1$ and the violating activities. . . . .	55
3.5	Diagnostics about violations of the constraint $C_1$ extended with information about the violation types and their frequency. . . . .	56
3.6	Information on how the process should have been executed differently to be compliant. . . . .	58
3.7	The compliance checking technique should consider the impact of proposed compliant values globally in a process instance. . . . .	62
3.8	Compliance analysis overview: Elicitation and formalization of compliance rules. . . . .	65
3.9	A Petri net modeling an exclusive relation between activities $A$ and $B$ . . . . .	69
3.10	Petri net $N_1$ formalizing the sequence $\langle A, B \rangle$ . . . . .	71
3.11	<i>Declarative style Petri net</i> $N_2$ allowing for sequences where $A$ is directly followed by $B$ including $\langle C, A, B, D \rangle$ . . . . .	72
3.12	<i>Declarative style Petri net</i> $N_3$ formalizing different instances of the compliance rule. . . . .	73
3.13	$N_4$ formalizes a relaxed version of the response compliance rule by adding or removing some elements to $N_3$ . $N_4$ also allows for $\langle B, A, C, B, A, B, B \rangle$ . . . . .	74
3.14	A data-aware Petri net formalizing the temporal compliance rule. . . . .	76
3.15	A data-aware Petri net confining the execution of activity $A$ w.r.t. role and amount attributes. . . . .	77
3.16	Violations of log trace $t: \langle A, D, B, D \rangle$ against two micro compliance patterns. . . . .	79
3.17	Violations of log trace $t$ against the composite compliance model. . . . .	80

3.18 Compliance analysis overview: Compliance checking. . . . .	83
3.19 Execution of activity <i>Demperidone administration (A)</i> is required after the activity <i>tube feeding (T)</i> when the value of <i>nutrition with multifiber (X)</i> was not increased by 2 kcal/ml. . . . .	84
3.20 Events in the log $L$ . . . . .	87
3.21 Preprocessed event log after adding control-flow checking attribute and shortening the event log. . . . .	88
3.22 Three process instance runs $P_{R1}$ , $P_{R2}$ , and $P_{R3}$ of the Petri net pattern. . . . .	89
3.23 An alignment between pre-processed event log of Fig. 3.21 and the process instance run $P_{R1}$ (in Fig. 3.22). . . . .	90
3.24 An alignment between between pre-processed event log of Fig. 3.21 and the run $P_{R2}$ (in Fig. 3.22). . . . .	90
3.25 An alignment between between pre-processed event log of Fig. 3.21 and the run $P_{R3}$ (in Fig. 3.22). . . . .	91
3.26 The temporal compliance rule formalized as the data-aware Petri net $DPN_{temporal}$ . . . . .	92
3.27 The control-flow alignment resulting from the alignment of log $L$ (in Fig. 3.20) and $DPN_{temporal}$ (in Fig. 3.26) . . . . .	92
3.28 The data-aware alignment resulting from the alignment of log $L$ and data-aware Petri net $DPN_{temporal}$ . . . . .	93
3.29 The resource compliance rule formalized as a data-aware Petri net $DPN_{resource}$ . . . . .	94
3.30 The event log $L$ of Fig. 3.20 is prepared for checking the resource compliance rule. . . . .	94
3.31 Data-aware alignment resulted from aligning the prepared log of Fig. 3.30 and $DPN_{resource}$ of Fig. 3.30. . . . .	95
3.32 Steps taken in each compliance checking techniques. Each dedicated compliance checking technique includes iterations of three main steps: log preparation, checking and log enrichment. . . . .	96
3.33 Generating enriched event log with diagnostics obtained from several alignments. . . . .	98
3.34 The event log $L$ enriched with compliance diagnostic information on compliance state of each event w.r.t. different rules, rule instances, violation types, and compliant values. . . . .	100
3.35 Generating enriched event log with diagnostics based on one overall alignment result. . . . .	102
3.36 Compliance analysis overview. . . . .	102
3.37 Statistics in different abstraction levels are built over enriched log with diagnostics including frequency of violations in total and per attribute (e.g. per activity). . . . .	103

3.38	Discovering process models that describe the behavior of cases violating a specific constraint compared to those that are compliant w.r.t. that constraint. . . . .	104
3.39	Correlations between violations and specific attribute values guide us in hypothesizing about the causes of violations. . . . .	105
3.40	Classifying violations and non-violations based on their context help us predict future violations. . . . .	105
4.1	Thesis road map gives the mapping of the sections in Chap. 4 on to our compliance analysis approach. . . . .	108
4.2	Atomic compliance pattern: Direct precedence. . . . .	111
4.3	Overview of the control-flow compliance checking methodology. . . . .	112
4.4	The example event log. . . . .	112
4.5	Abstracted event log obtained from step 1. . . . .	113
4.6	Shortened event log obtained from step 2. . . . .	113
4.7	Control-flow alignment obtained from step 3. . . . .	114
4.8	Atomic compliance pattern: Bounded existence of an activity exactly $k$ times. . . . .	117
4.9	Abstracted event log obtained from step 1. . . . .	118
4.10	Abstracted event log obtained from step 2. . . . .	119
4.11	Control-flow alignment obtained from step 3. . . . .	119
4.12	Atomic compliance pattern: Simultaneous occurrence of two activities. . . . .	120
4.13	An event log where activities have life-cycle attributes. . . . .	121
4.14	Abstracted event log obtained from step 1. Values of <i>activity name</i> and <i>life cycle</i> attributes are combined for the <i>checking attribute</i> . . . . .	122
4.15	Shortened event log obtained from step 2. The event $e_{12}$ is deleted during the shortening of the event log. . . . .	123
4.16	Control-flow alignment obtained from step 3. . . . .	124
4.17	Mapping transition in the compliance pattern to events of the log. . . . .	129
4.18	Mapping transition in the compliance pattern to events of the log. . . . .	129
4.19	Activity X-ray must be executed at least once per case. . . . .	130
4.20	A non-compliant case: activity <i>patient registration</i> should have occurred exactly once. . . . .	131
4.21	Activity <i>patient registration</i> must be followed by activity X-ray. . . . .	132
4.22	An example of a non-compliant case. Activity <i>X-ray</i> is executed in a wrong position. . . . .	133
4.23	Either CT-scan or MRI test must be taken for a patient. . . . .	134
4.24	An example of a non-compliant case. Activities <i>MRI</i> and <i>CT-scan</i> both are executed within a case. . . . .	135

4.25	The pattern enforces that a patient must be registered before receiving any treatment. . . . .	135
4.26	The pattern enforces multiple registration of a patient and direct sequence of <i>patient registration</i> followed by <i>X-ray</i> . . . . .	136
4.27	Control-flow compliance rule elicitation and formalization overview. . .	137
4.28	Sequence of <i>patient registration</i> and <i>X-ray</i> . . . . .	139
4.29	Configurable sequence of <i>Patient registration</i> and <i>X-ray</i> . . . . .	140
4.30	QT-phase1 (left), QT-phase2 (right). . . . .	142
4.31	<i>Elicit Compliance Rule</i> plugin: Example compliant and violating cases when a specific choice is made by the user. . . . .	146
5.1	Thesis road map gives the mapping of the sections in Chap. 5 on to our compliance analysis approach. . . . .	155
5.2	Executions of activities may overlap in different ways. . . . .	157
5.3	Overview of the temporal compliance checking methodology. . . . .	158
5.4	Example 1: Event log. . . . .	159
5.5	Example 1: Control-flow rule modeling cycles of three occurrences of activity antibiotic administration. . . . .	160
5.6	Alignment of the log of Fig. 5.4 to the control-flow rule of Fig. 5.5 showing a model-only move between $e_6$ and $e_7$ , i.e., once occurrence of $A$ was missing in the second rule instance. . . . .	161
5.7	Example 1, event log enriched with control-flow violations and temporal information. . . . .	161
5.8	Example 1, the temporal dimension of the rule <i>TR 1</i> modeled as a data-aware Petri net. . . . .	162
5.9	Example 1: Data-aware alignment of the enriched event log of Fig. 5.7 to the temporal compliance pattern of Fig. 5.8. . . . .	164
5.10	Projection of diagnostics on a timeline. . . . .	164
5.11	Enriched event log with control-flow and temporal diagnostics. . . . .	166
5.12	The generic temporal pattern. . . . .	167
5.13	The generic temporal pattern specifying delay between rule instances. .	168
5.14	Both $R_1$ and $R_1$ modelled together as a composite model by instantiating the generic temporal pattern. . . . .	172
5.15	Sequence of activities $\langle D, A, A, A, B \rangle$ . . . . .	173
5.16	A composite model for specifying rules <i>rule 1</i> and <i>rule 2</i> together. . . .	173
5.17	Rule 3, Event log. . . . .	174
5.18	Rule 3- Atomic pattern describing the control-flow rule instance. . . .	175
5.19	Rule 3- Control-flow alignment of the log of Fig. 5.17 to the control-flow rule of Fig. 5.18. . . . .	175

5.20 Rule 3- Enriched event log with diagnostics obtained from control-flow alignment in Fig. 5.19. . . . . 175

5.21 Rule 3- Preparing the event log of Fig. 5.20 for temporal compliance checking by abstracting from activities outside rule instances. . . . . 176

5.22 Rule 3- Temporal pattern instantiated to specify the delay between activities *A* and *B*. . . . . 177

5.23 Rule 3- Data-aware alignment of prepared log of Fig. 5.21 with the temporal pattern of Fig. 5.22 indicates a temporal violation. . . . . 177

5.24 Rule 3- Enriched event log with complete diagnostics obtained from data-aware alignment of Fig. 5.23. . . . . 178

5.25 The data-aware alignment will suggest that activity *B* must occur at time 2.179

5.26 The temporal compliance checking minimizes the total deviation of observed behavior from prescribed behavior. . . . . 181

5.27 The ‘Not-preferred’ control-flow alignment will lead to more temporal violations. . . . . 182

5.28 Temporal compliance checking using data-aware alignment plugin takes a control-flow alignment as input. . . . . 184

5.29 Instantiating the generic temporal pattern and definition of guards. . . 185

5.30 During temporal compliance checking, we allow the user to specify how the missing temporal values should be repaired. . . . . 185

5.31 Mapping transitions of the generic temporal pattern to events of the log. 186

5.32 Mapping variables in the temporal pattern to attributes in the log. . . . 186

5.33 Setting costs for different temporal violations. . . . . 187

5.34 Temporal compliance checking provides combined diagnostics including control-flow and temporal violations. . . . . 187

5.35 Violations versus handover of work. . . . . 189

5.36 Several hand-overs observed for a violating case within a short period of time. . . . . 191

5.37 Example of a compliant case processed by a single resource. . . . . 192

6.1 Thesis road map gives the mapping of the sections in Chap. 6 on to our compliance analysis approach. . . . . 197

6.2 Methodology for checking the first category of data-aware and resource-aware compliance rules. . . . . 200

6.3 Methodology for checking data-aware and resource-aware compliance rules of the second category. . . . . 201

6.4 An example event log partially shown for process instance  $p_1$ . . . . . 202

6.5 The event log, built upon original log of Fig. 6.4, is abstracted and shortened from irrelevant information. . . . . 202

6.6	An atomic pattern specifying the control-flow condition of <i>rule 1</i> . . . . .	203
6.7	The control-flow alignment obtained from aligning <i>PL</i> of Fig. 6.5 and the atomic pattern of Fig. 6.6. . . . .	204
6.8	The enriched log, built from control-flow alignment of Fig. 6.7, contains control-flow diagnostics. . . . .	205
6.9	The atomic pattern of Fig. 6.6 augmented with resource condition of <i>rule 1</i> . . . . .	205
6.10	The data-aware alignment of the enriched log of Fig. 6.8 to pattern of Fig. 6.9, indicates that the two activities were executed by different resources. . . . .	206
6.11	The enriched log obtained from data-aware alignment of Fig. 6.10 contains diagnostics about compliance of <i>rule 1</i> . . . . .	207
6.12	Part of an example event log of a patient in ICU. . . . .	208
6.13	The prepared log, built upon original log of Fig. 6.12, obtained from the first step. . . . .	209
6.14	The data-aware Petri net is modelled to capture the situation where the data condition of <i>rule 2</i> holds. . . . .	210
6.15	The data aware alignment, obtained from aligning prepared log of Fig. 6.13 and pattern in Fig. 6.14, indicates situations where <i>nutrition</i> is not increased by <i>2 kcal/ml</i> . . . . .	210
6.16	The enriched event log with data condition obtained from the data-aware alignment of Fig. 6.15. . . . .	211
6.17	The checking attribute combines the values of <i>data condition</i> and <i>activity name</i> at each event in the enriched log of Fig. 6.16. . . . .	212
6.18	The prepared log has a reduced size by shortening sequence of $\Omega$ events. . . . .	213
6.19	The atomic pattern specifying the control-flow condition of the rule. . . . .	214
6.20	The control-flow alignment indicates that in the second instance of the rule, activity <i>Demperidone administration</i> is skipped. . . . .	215
6.21	The enriched event log with diagnostics obtained from the control-flow alignment. . . . .	216
6.22	A process model in BPMN describing correct occurrences of activities <i>A, B, C</i> , and <i>D</i> . . . . .	217
6.23	The <i>Data and Resource Compliance Checking-Rules Restricting Attributes</i> plug-in checks the compliance of an event log against the first category of compliance rules. . . . .	220
6.24	The <i>Data and Resource Compliance Checking-Rules Restricting Activities</i> plug-in checks the compliance of an event log against the second category of compliance rules. . . . .	221
6.25	During the log preparation step, the activities that can read and write an attribute value are selected. . . . .	221



6.26	Mapping attributes of the compliance pattern to event log attributes. . .	222
6.27	Mapping transition labels of the compliance pattern to the values of the checking attribute in the event log. . . . .	222
6.28	Classification of applications based on their compliance result. . . . .	223
6.29	An example of a compliant case where nutrition was increased via tube feeding. . . . .	224
6.30	An example of a compliant case where <i>Demperidone</i> is administered to the patient when nutrition was not increased via tube feeding. . . . .	225
6.31	An example of a case where <i>Demperidone</i> is administered although nutrition was increased via tube feeding. . . . .	226
6.32	An example of a violating case where <i>Demperidone</i> is not administered to the patient although nutrition was not increased via tube feeding. . .	227
7.1	Thesis road map gives the mapping of the sections in Chap. 7 on to our compliance analysis approach. . . . .	233
7.2	A process model for procurement . . . . .	234
7.3	Overview of the approach. . . . .	238
7.4	The data-aware Petri net $DPN^{procurement}$ specifying admissible executions of procurement process example. . . . .	239
7.5	Parts of the event log $L$ showing four events from process instance $p_1$ . .	241
7.6	Alignment $A$ resulted from aligning event log $L$ and process instance run $R$ of process model $DPN^{procurement}$ . . . . .	241
7.7	Deviation statistics for procurement process example. . . . .	243
7.8	Statistics for attribute <i>material</i> (top) and detailed statistics for <i>material</i> = <i>glass</i> . . . . .	247
7.9	The structure of <i>problem lists</i> . . . . .	248
7.10	A Venn diagram representation of how <i>activity specific</i> sets of deviations referring to different problem statement levels are built. . . . .	250
7.11	Venn diagram representation of how <i>attribute specific</i> sets of deviations referring to different problem statement levels are built. . . . .	252
7.12	A decision tree classifying moves to those skipped <i>check off-site warehouses</i> and those did not, considering inventory level and material as predictor variables. . . . .	262
7.13	Part of the process model discovered from violating cases with inventory level above threshold. . . . .	266
7.14	The compliant procurement process analyzed for experimental results. .	268
7.15	Activity deviations statistics table of experimental result. . . . .	269
7.16	Parts of the model describing cases containing violating activity <i>outgoing payment</i> . . . . .	271

7.17	Part of the model describing cases containing unallowed executions of activity <i>goods receipt</i> . . . . .	272
7.18	Part of the model describing violating cases where activities <i>clearing credit</i> and <i>clearing debit</i> occurred before <i>invoice receipt</i> . . . . .	273
7.19	Part of the <i>problem list</i> obtained for the example log in our case study. . . . .	274
7.20	Resources frequently linked to violations. . . . .	275
7.21	Root-cause analysis of unallowed executions of activities <i>clearing debit</i> and <i>clearing credit</i> wrt attribute <i>resource</i> . . . . .	277
8.1	Compliance Analysis Overview. . . . .	281
8.2	Generating an enriched event log with diagnostics based on one alignment result. . . . .	282
8.3	Generating an enriched event log with diagnostics obtained from several alignments. . . . .	282
8.4	Thesis road map gives the mapping of the sections in Chap. 8 on to our compliance analysis approach. . . . .	284
8.5	Sources of compliance data. . . . .	286
8.6	Different clusters of compliance data. . . . .	288
8.7	Relational data model of the compliance database. . . . .	289
8.8	Parts of Fig. 8.7 are chsen that show data entities and their relations related to <i>compliance rule profile</i> . . . . .	290
8.9	An example compliance profile populated with two constraints and their rules. . . . .	291
8.10	Data entities and their relations related to <i>event data</i> . . . . .	291
8.11	Event data tables populated with an example <i>log</i> , two <i>traces</i> , and their <i>events</i> . . . . .	293
8.12	Data entities and their relations related to <i>prepared log</i> . . . . .	294
8.13	Entities and their relations of an example prepared log cluster populated with four <i>prepared logs</i> , eight <i>prepared traces</i> , and their <i>prepared events</i> . . . . .	295
8.14	Data entities and their relations related to <i>compliance checking result</i> cluster. . . . .	296
8.15	Example entities and their relations related to a compliance checking result. Figure above shows four <i>alignments</i> , ten types of <i>violations</i> , parts of <i>move</i> table, and parts of <i>rule_instance</i> table. . . . .	298
8.16	An example query in SQL requesting all the compliance rule that a process <i>p</i> is checked against. . . . .	300
8.17	An example query in SQL requesting all the moves within time period $[t_1, t_2]$ w.r.t. all relevant compliance rules that a process <i>p</i> is checked against. . . . .	301

8.18	The enriched event log with control-flow and temporal diagnostics. . .	306
8.19	Proposed framework for populating compliance database. . . . .	308
9.1	Life cycle of an unemployment application in UWV. . . . .	315
9.2	<i>Split testing</i> (A versus B) design of the case study including three business units and steps taken for each business unit. . . . .	319
9.3	Different phases of the case study and their timing. . . . .	322
9.4	Different steps executed during the pilot study. . . . .	322
9.5	Collecting data from different data sources. . . . .	325
9.6	Comparing compliance results obtained during the first study for different business units. . . . .	331
9.7	Filtered association rules mined between different violations and process related attributes in <i>Business Unit 1</i> . . . . .	333
9.8	Letter titles connected to violations of <i>Rule (2)</i> in each business unit. . .	335
9.9	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (1)</i> in <i>Business Unit 1</i> . . . . .	340
9.10	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (1)</i> in <i>Business Unit 2</i> . . . . .	341
9.11	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (1)</i> in <i>Business Unit 3</i> . . . . .	341
9.12	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (2)</i> in <i>Business Unit 1</i> . . . . .	342
9.13	Association rules mined between different violations and client related attributes in <i>Business Unit 1</i> . . . . .	343
9.14	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (1)</i> in <i>Business Unit 1</i> . . . . .	345
9.15	Violating patterns that distinguish compliant and violating activities w.r.t. compliance <i>Rule (1)</i> for <i>Business Unit 3</i> . . . . .	346
9.16	History of violations of <i>Rule (7)</i> in three business units. . . . .	351
10.1	The main contribution of this work are highlighted in the thesis road map.362	
10.2	Proposed compliance engine for real time compliance management. . .	368
A.1	<i>Existence. Activity Universality</i> compliance rule . . . . .	371
A.2	<i>Existence. Activity Absence</i> compliance rule . . . . .	372
A.3	<i>Dependent Existence. Exclusive</i> compliance rule . . . . .	373
A.4	<i>Dependent Existence. Mutual Exclusive</i> compliance rule . . . . .	374
A.5	<i>Dependent Existence. Prerequisite</i> compliance rule . . . . .	375
A.6	' <i>Dependent Existence. Inclusive</i> ' compliance rule . . . . .	376

A.7	<i>Dependent Existence. Substitute</i> compliance rule . . . . .	377
A.8	<i>Dependent Existence. Co-requisite</i> compliance rule . . . . .	378
A.9	<i>Bounded Existence of an Activity. Exactly k Times</i> compliance rule . . . .	379
A.10	<i>Bounded Existence of an Activity. At Least k Times</i> compliance rule . . . .	380
A.11	<i>Bounded Existence of an Activity. At Most k Times</i> compliance rule . . . .	381
A.12	<i>Bounded Existence of an Activity. Exactly k Times in a Row</i> compliance rule	382
A.13	<i>Bounded Existence of an Activity. At Least k Times in a Row</i> compliance rule	383
A.14	<i>Bounded Existence of an Activity. At Most k Times in a Row</i> compliance rule	385
A.15	<i>Bounded Existence of an Activity. Bursts of k Occurrences</i> compliance rule	386
A.16	<i>Bounded Existence of an Activity. n Bursts of k Occurrences</i> compliance rule	387
A.17	<i>Bounded Sequence of Activities. One to One Coexistence</i> compliance rule .	388
A.18	<i>Bounded Sequence of Activities. Coexistence</i> compliance rule . . . . .	389
A.19	<i>Bounded Sequence of Activities. Exactly k Times</i> compliance rule . . . . .	390
A.20	<i>Parallel. Simultaneous</i> compliance rule . . . . .	391
A.21	<i>'Parallel. During'</i> compliance rule . . . . .	392
A.22	<i>Precedence. Simultaneous or Before</i> compliance rule . . . . .	393
A.23	<i>Precedence. Direct</i> compliance rule . . . . .	394
A.24	<i>Precedence. Direct or Indirect</i> compliance rule . . . . .	395
A.25	<i>Precedence. At Least Once</i> compliance rule . . . . .	396
A.26	<i>Precedence. Direct Multiple Activities</i> compliance rule . . . . .	397
A.27	<i>Precedence. Direct or Indirect Multiple Activities</i> compliance rule . . . . .	397
A.28	<i>Precedence. Direct Multiple Different Activities</i> compliance rule . . . . .	398
A.29	<i>Precedence. Direct or Indirect Multiple Different Activities</i> compliance rule	399
A.30	<i>Precedence. Never Direct</i> compliance rule . . . . .	400
A.31	<i>Precedence. Never</i> compliance rule . . . . .	401
A.32	<i>Chain Precedence. Direct</i> compliance rule . . . . .	403
A.33	<i>Chain Precedence. Direct or Indirect</i> compliance rule . . . . .	405
A.34	<i>Chain Precedence. Never Direct</i> compliance rule . . . . .	406
A.35	<i>Chain Precedence. Never</i> compliance rule . . . . .	408
A.36	<i>Response. Simultaneous or After</i> compliance rule . . . . .	409
A.37	<i>Response. Direct</i> compliance rule . . . . .	410
A.38	<i>Response. Direct or Indirect</i> compliance rule . . . . .	411
A.39	<i>Response. At Least Once</i> compliance rule . . . . .	412
A.40	<i>Response. Direct Multiple Activities</i> compliance rule . . . . .	412
A.41	<i>Response. Indirect Multiple Activities</i> compliance rule . . . . .	413
A.42	<i>Response. Direct Multiple Different Activities</i> compliance rule . . . . .	414
A.43	<i>'Chain Response. Direct'</i> compliance rule . . . . .	415
A.44	<i>Chain Response. Direct or Indirect</i> compliance rule . . . . .	418
A.45	<i>Chain Response. Never Direct</i> compliance rule . . . . .	419

A.46	<i>Between. After-Before</i> compliance rule . . . . .	421
A.47	<i>Between. Simultaneously or After-Before</i> compliance rule . . . . .	422
A.48	<i>Between. After-Simultaneously or Before</i> compliance rule . . . . .	422
A.49	<i>Between. Directly After-Directly Before</i> compliance rule . . . . .	423
A.50	<i>Between. Simultaneously-Simultaneously</i> compliance rule . . . . .	424
A.51	<i>Between. Simultaneously or After-Simultaneously or Before</i> compliance rule	425
A.52	<i>Between. At Least One Other Activity</i> compliance rule . . . . .	426
B.1	The generic temporal pattern. . . . .	428
C.1	Configuration options in <i>Bounded Existence-Upper Bound</i> configurable rule.	438
C.2	Configuration options in <i>Bounded Existence. Lower Bound</i> configurable rule. . . . .	440
C.3	Configuration options in <i>Bounded Existence. Upper Bound</i> configurable rule. . . . .	440
C.4	Configuration options in <i>Cyclic Occurrence</i> configurable rule. . . . .	441
C.5	Configuration options in <i>Sequence of Activities</i> configurable rule. . . . .	442
C.6	Configuration options in <i>Sequence of Multiple Activities</i> configurable rule. . . . .	442
C.7	Configuration options in <i>Negative Precedence or Response</i> configurable rule.	443
C.8	Configuration options in <i>Bounded Sequence</i> configurable rule. . . . .	444
C.9	Configuration options in <i>Chain Precedence</i> configurable rule. . . . .	446
C.10	Configuration options in <i>Chain Response</i> configurable rule. . . . .	447
C.11	Configuration options in <i>Between</i> configurable rule. . . . .	448
C.12	Configuration options in <i>Not-in-Between</i> configurable rule. . . . .	449
C.13	Configuration options in <i>Parallel. During</i> configurable rule. . . . .	449
C.14	Configuration options in <i>Parallel. During (Sequence of Activities)</i> configurable rule. . . . .	450
C.15	Configuration options in <i>Parallel. Simultaneous</i> configurable rule. . . . .	451
C.16	Configuration options in <i>Inclusive, Pre-requisite and Co-requisite</i> configurable rule. . . . .	452
C.17	Configuration options of <i>Exclusive Compliance</i> rule . . . . .	453
C.18	Configuration options of <i>Substitute Compliance</i> rule . . . . .	453
D.1	Filtered association rules mined between different violations and process related attributes in <i>Business Unit 1</i> . The brown ellipses in the middle indicate violations related to different rules. The blue edges show associations of type <i>attribute value to violation</i> ( $A \rightarrow V$ ) and yellow edges indicate associations of type <i>violation to attribute value</i> ( $V \rightarrow A$ ). . . . .	456

---

D.2 Association rules mined between different violations and client related attributes in *Business Unit 1*. . . . . 460



# List of Tables

3.1	Categorization of control flow compliance rules. . . . .	66
3.2	Categorization of the 15 temporal compliance rules. . . . .	67
3.3	Collection of data-aware and resource-aware compliance rules. . . . .	68
4.1	Categorization of the 54 control-flow compliance rules. . . . .	109
5.1	Categorization of the 15 temporal compliance rules. . . . .	156
5.2	Temporal compliance violations. . . . .	189
6.1	Collection of data-aware and resource-aware compliance rules. . . . .	198
7.1	Event attributes of the synthetic log obtained for the procurement process.	240
7.2	Sets of different basic move types. . . . .	244
7.3	Different sets of moves related to a specific attribute. . . . .	245
7.4	Sets of compliance elements in compliance element collection $\mathcal{C}$ . . . . .	258
7.5	CPIR value and direction of some of the associations between 'activity check off-site was skipped' and its context. . . . .	261
7.6	Result of the survey on the visualization of the results in the <i>Compliance Framework</i> . . . . .	276
8.1	Number of scripts developed per component. . . . .	310
8.2	Summary of the dataset used as proof of concept. . . . .	310
9.1	Compliance rule types to be checked. . . . .	318



9.2	Two groups of data attributes (process and client related) that we included in case study dataset. . . . .	323
9.3	The volume of cases and events in datasets for each business unit.	326
9.4	Total number of rule instances and the violation frequency per compliance rule for each business unit. . . . .	327
9.5	Total number of letters per category and distribution of violations w.r.t. compliance <i>Rule 2</i> over letter categories. . . . .	328
9.6	The total number of letters per category and distribution of violations w.r.t. compliance <i>Rule 5</i> over letter categories. . . . .	329
9.7	The total number of letters per category and distribution of violations frequency w.r.t. compliance <i>Rule 6</i> per letter category. . .	330
9.8	Association rules mined for different business units between violations and their contextual data. . . . .	332
9.9	Filtered associations between different violations and <i>team 12Y</i> in the three business units. . . . .	337
9.10	Grouping of attributes for classifying violations versus non-violations.	339
9.11	Number of cases and events in datasets of each business unit . .	347
9.12	Follow-up study, summary of violations per compliance rule in three business units. . . . .	348
A.1	Categorization of the 54 control-flow compliance rules. . . . .	370
D.1	Summary of connections (between submission types and different violations) and (between application phases and different violations) in three business units. The associations are of type <i>attribute value to violation</i> ( $A \rightarrow V$ ) or of type <i>violation to attribute value</i> ( $V \rightarrow A$ ). . . . .	458
D.2	Summary of connections between decision categories and different violations in the three business units. The associations are of type <i>attribute value to violation</i> ( $A \rightarrow V$ ) or of type <i>violation to attribute value</i> ( $V \rightarrow A$ ). . . . .	459
D.3	Summary of connections between different violations and values of attributes <i>entitled for additional allowance</i> and <i>reasons of unemployment</i> . The associations are of type <i>attribute value to violation</i> ( $A \rightarrow V$ ) or of type <i>violation to attribute value</i> ( $V \rightarrow A$ ). . . . .	463

# Chapter 1

## Introduction

This chapter gives an introduction to an integrated compliance checking and analysis approach. We first discuss in Sect. 1.1 the challenges of legal constraints and regulations and how (Big) Data techniques provide new opportunities to ensure adherence to regulations, also known as *Compliance*. We then discuss various approaches to ensure compliance using a fictive example corporation in Sect. 1.2. In Sect. 1.3, we then identify a number of requirements for realizing an integrated compliance checking technique for *all* compliance-relevant activities of an organization and discuss the state of the art. We present in Sect. 1.4 our approach towards realizing such an integrated technique and summarize our results. Sect. 1.5 then provides a detailed overview of the structure of the thesis.

### **1.1 Adherence to Regulations in Times of Big Data**

The increasing power of compliance sources such as laws and regulations in governing business operations has led to a need for systematic approach for compliance management in corporations. The management (especially in larger corporations) assign annually a considerable budget to internal and external auditing projects to ensure authorities that they have enforced necessary controls in their operation.

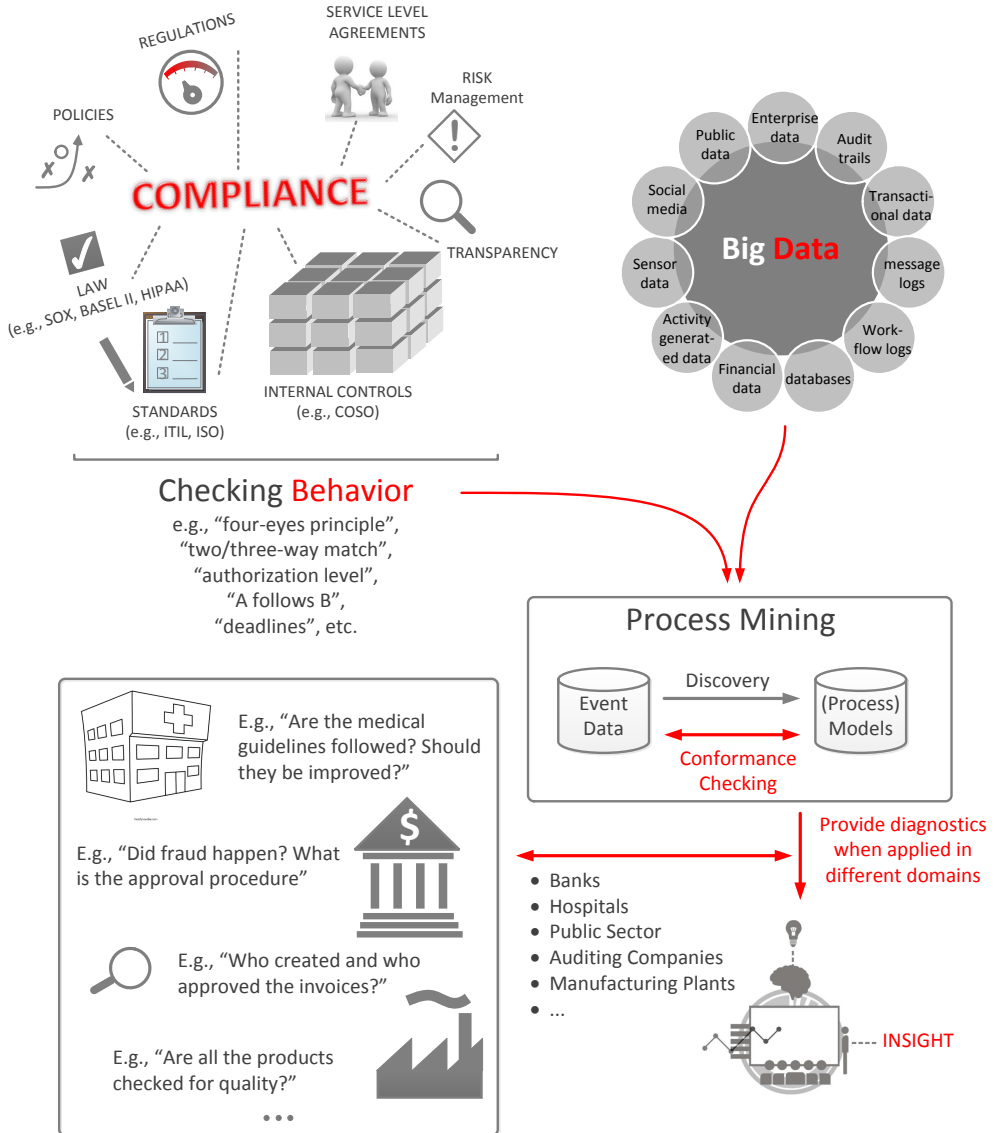


Figure 1.1: Omnipresence of data allows us to leverage process mining and data mining techniques for data-driven analysis of compliance.

Adherence to these compliance procedures also enables the management to be in control of their business and make their operations more transparent. The transparency obtained gives them a bargaining power in the market and attracts the trust and confidence of their stakeholders and business partners including their clients and suppliers.

Legislations such as Sarbanes-Oxley Act (SOX) Act of 2002, Basel II Accord of 2004 (scheduled to be superseded by BASEL III), and COSO Framework of 1992 superseded by its updated version of 2013, urge corporations to ensure and prove that they have implemented internal control systems. These regulations require corporations to keep monitoring and evaluating their internal control systems regularly.

Furthermore, it is not only laws and legislations that impose constraints on corporations. Industry guidelines and standards such as ISO standards or Information Technology Infrastructure Library (ITIL) for IT service management, in many cases are demanded by clients or business partners of corporations in each industry. Apart from legal consequences of non-compliance, not complying to the standards can cause the cost of losing a client or a business opportunity as well.

Running compliance related activities incurs costs. However, the cost of non-compliance can be staggering for organizations. According to Pricewaterhouse Coopers (PwC) Crime Survey 2011<sup>1</sup> involving 250 Dutch companies, antitrust(cartel) agreements or financial fraud caused a direct loss of more than 500,000 euro in a quarter of cases. The crime survey of 2014<sup>2</sup> shows a relative increase of 13% in crimes related to bribery and corruption from what reported in 2011.

Banks, hospitals, manufacturers, service provider, organizations in public sector and almost every organization run information systems to support their operations. These information systems can record detailed processing steps, data accesses, message logs, and as illustrated in Fig. 1.1, many more types of information. More and more organizations are inclined to store these logged data at various granularity levels as the storage has become cheap and processing power has increased.

At the same time, new technologies in data analysis and in particular process mining [134] are providing opportunities to systematically observe processes at a detailed level. The aim of this thesis is to extend and develop techniques that can compare recorded event data to compliance constraints to answer and verify

---

<sup>1</sup><http://crimesurvey.pwc.nl> accessed on 2014.04.14

<sup>2</sup><http://www.pwc.com/crimesurvey> accessed on 2015.01.15

questions such as: *Are the processes compliant or not? Which of the guidelines are violated? What kind of violations are observed? Can a pattern be detected among violations? etc.*

Data is a strategic asset for organizations and as data-driven strategies take hold, they will become an increasingly important point of competitive differentiation. In this thesis we leverage from various process mining and data mining techniques to analyze compliance of business processes and answer concrete compliance related questions in various domains including health-care, banking, auditing, public sector, and etc.

## 1.2 Problem Statement and Research Goal

Suppose *Life Inc.* is a corporation including a family of pharmaceutical companies active in different regions of Europe. *Life* produces dozens of specialized medicines. The corporation has several production lines for its groups of products. The operations of this corporation are highly regulated. There are many health care regulations and protocols that need to be followed. Failing to comply to any of these rules can have severe legal consequences for the company and ruin its reputation. Furthermore, the consequences of such a failure may be severe as the products of this corporation are directly connected to human

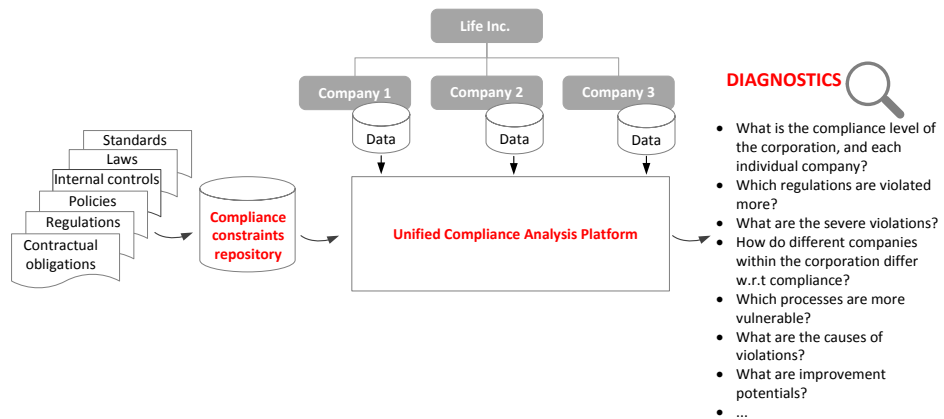


Figure 1.2: Life Inc. requires an integrated compliance solution which is generic, flexible, automated and provides detailed diagnostics.

health.

Medical guidelines and health-care protocols are not the only protocols that the corporation should adhere to. The contractual obligations set with partners and the industry specific standards are also important for the company. There are several internal policies and procedures designed by in-house experts to improve the productivity and efficiency in daily operations. Keeping track of all the rules and monitoring the compliance status against them is a tedious task for the management team. In addition, it seems that there is not common agreement among different companies of this corporation about the rules and their interpretation in daily operations.

As illustrated in Fig. 1.2, the corporation requires to detect possible violations against all the relevant compliance constraints and monitor its compliance state level as a holding and in each of its individual companies. To better understand its compliance state, the management team needs to know which regulations are often violated and which of the processes are more vulnerable? In addition to find improvement potential, the deviations need to be analyzed for their causes. Above all, it is of the utmost importance to have a common understanding, within the corporation, of all the compliance constraints that need to be followed.

Currently, the corporation lacks a unified platform for collecting and analyzing compliance data. Monitoring compliance is limited to auditing initiatives on sample data which is incomplete, manual and error prone. The management requires an automated solution that enables continuous checking of whole population of data not only a sample. In fact, the management requires a dashboard to monitor the compliance and get diagnostics on different abstraction levels and from different perspectives. The complexity and difficulty of such a comprehensive compliance solution comes from various sources as follows.

- Compliance constraints are typically written in an informal domain specific language. To enable automated compliance checking, compliance constraints need to be *specified precisely and formally*. However, compliance experts who are in charge of interpretation of compliance constraints are usually less familiar with formalization techniques.
- Compliance constraints need to consider *various perspectives of a business process* such as order of steps in a process, time between the execution of these steps or data involved in executing various steps. Each type of compliance constraint, requires a specific checking mechanism.
- Designing a compliant business process *does not guarantee* the compliant

execution of a business process. It is hard and undesirable to enforce constraints in a very strict manner and often there need to be ways to *check compliance after/during execution*.

- For improving compliance, *detailed diagnostics about violations* are required. That is, it is not only enough to know whether a process is compliant w.r.t. a rule, but also it is important to know what went wrong, what should have happened instead and why a certain violation has happened.
- To have a complete picture about the compliance of a process, it is *not enough to analyze a sample of cases*. Analyzing all the cases of a process is only possible with an *automated checking technique* that can detect violations of different types.

An option for improving compliance and transparency would be the incorporation of all the relevant compliance constraints within business processes of the companies. Following this approach will lead to adding many controlling tasks to the underlying information systems supporting the processes. This option requires costly and time consuming changes for re-engineering of business processes and re-configuration of underlying information systems. In addition, inserting controlling tasks within business processes are not entirely aligned with the business objectives of the company and will make the processes less agile and less efficient. Even if these changes are in place, the management team still can not be sure that the processes are executed as they are designed and all the rules are followed in practice. Furthermore, compliance constraints are changing and it is not feasible to change the processes and the supporting information systems every time a regulation changes. All in all, polluting business processes with compliance constraints does not seem to be a sustainable solution for the corporation. The advantage, however, will be a unified system of controls within the holding.

An alternative approach would be the establishment of a unified set of compliance rules for all the companies. Then, a generic compliance analysis technique is required for analyzing historic data. The technique should be able to detect any possible violation against all types of rules without changing the design of business processes and configuration of underlying information systems. As mentioned earlier, for this, an automated and comprehensive compliance checking technique is required for analyzing the complete population of data. This solution should consider all the complexities mentioned earlier. In this case, all the changes in regulations are managed separately from the business

processes and in a unified manner for all the companies. The result of checking is reliable, i.e., the management can trust on the result of the checking in a sense that checking is done based on the facts not assumptions about the facts.

### Research Goal

In this thesis, we will develop a comprehensive and data-driven approach for analyzing compliance and understanding non-compliance. For this, we first need a technique to transform informal constraints into precise and formalized compliance rules. Although there are various rigorous mathematical formalisms for representing compliance rules, these are often perceived to be difficult to use for business users. The main challenges here are **1) to find a way to bridge the gap between the informal description and the precise specification of all compliance constraints in all process perspectives, and 2) to provide compliance experts with techniques to create formal specifications without being exposed to the details and difficulties of a formal language.**

Our approach should enable **3) automated compliance checking of all types of process compliance constraints.** In particular, a compliance constraint may address different perspectives of a business process including the control-flow of a process, the time perspective, the flow of data, and organizational aspects. The checking technique should enable a thorough compliance analysis and be able to detect all types of violations. The challenge here is to develop an approach which is **4) generic enough so that it enables checking of different types of rules but at the same time specific enough to give us a precise result about compliance.** In addition, in case of a violation, it is necessary **5) to provide detailed diagnostics about the violations.**

Our approach should also **6) enable root-cause analysis on violations.** To improve compliance it is important to know what the common and important violations and their frequencies are and why they occur. The challenge in this part is to **7) identify key insights about violations in data that enable compliance experts to understand non-compliance and plan corrective measures accordingly.** Finally, a comprehensive compliance analysis approach should **8) aggregate all compliance data and support exploring and reporting about compliance.**

Existing techniques often address only a few of these requirements, but to the best of our knowledge there exists no solution that addresses all of the above in an integrated compliance analysis solution. The goal of this thesis is to provide an approach that addresses all 8 requirements in a comprehensive manner. Next, we will discuss state of the art w.r.t. compliance related activities.



## 1.3 State of the Art in Compliance Management

Compliance management includes a set of activities in three interrelated but distinct perspectives on compliance, namely *preventive*, *detective*, and *corrective* [144]. These techniques aim at ensuring that the operations of a company comply with regulations. Five types of compliance-related activities can be identified [49, 99]:

- *compliance elicitation*: determine the relevant constraints for a process that need to be satisfied (i.e., rules defining the boundaries of compliant behaviors),
- *compliance formalization*: formulate precisely the compliance constraints derived from laws and regulations in compliance elicitation,
- *compliance implementation*: implement and configure information systems such that they fulfil compliance constraints,
- *compliance checking*: investigate whether the constraints will be met (forward compliance checking) or have been met (backward compliance checking), and
- *compliance improvement*: modify the processes and supporting information systems based on the diagnostic information in order to improve compliance.

These activities form *five phases* in *Compliance Management (CM) life cycle* shown in Fig. 1.3. These phases are: (1) *elicit*, (2) *formalize*, (3) *implement*, (4) *analyze*, and (5) *improve*. The challenges we listed above in Sect. 1.2 are related to different phases of CM life cycle.

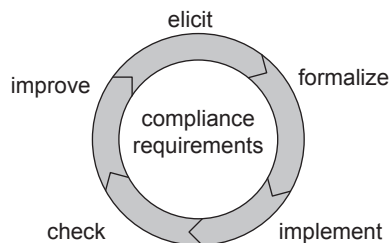


Figure 1.3: Compliance Management life cycle.

Next, we give a short overview about the literature available for different activities in CM life cycle. A more precise analysis of related works will be done in the individual chapters.

**Elicitation and formalization of compliance constraints.** Several formalization languages are proposed to capture compliance constraints precisely. For example Formal Contractual Language (FCL) proposed in [113] extends Deontic and Defeasible logic with modal operators, where a compliance constraint can be reduced to the identification of what obligations an enterprise has to fulfill to be deemed as compliant. The combination of the two logics offers a good trade off between expressive power and computational complexity. The critical concern related to FCL is the high complexity of the language which makes it difficult for compliance and business experts to use. Various techniques are developed for precisely describe compliance constraints in Linear Temporal Logic (LTL) [92]. The advantages of this language is that several verification tools have been developed for checking LTL formulas. Although LTL formulas are simpler than FCL, they are still complex to write for business users. In addition, the verification of formulas in LTL will lead to true or false result and does not give more diagnostics. The above mentioned formalization languages are just examples. In general, the problem for using different formalization languages remains to be their complexity for business users.

To facilitate the task of formalization, many approaches use pre-formulated templates and specification patterns for formulating compliance constraints [?, 40, 46, 132]. Specification patterns are extensively used in software development [4, 26, 37, 66, 123]. A common problem in most of above mentioned works is that pre-formulated patterns are limited and hard coded, hence they fail to capture subtle aspects of different compliance constraints.

**Compliance implementation and checking.** There are two basic types of compliance checking: (1) *forward compliance checking* aims to design and implement processes where conforming behavior is enforced and (2) *backward compliance checking* aims to detect and localize non-conforming behavior.

- *Forward* compliance checking aims at ensuring compliant process executions. Processes can be constructed to be compliant [113] or verified whether they are compliant [52, 53, 77, 114]. Alternatively, compliance constraints can be transformed into monitoring rules [49] or model annotations which then are used to *enforce* compliant process executions [16, 46]. Diagnostic information is obtained by pattern matching [14, 15, 15, 48].

- *Backward* compliance checking evaluates in hindsight whether process executions complied to all compliance rules or when and where a particular rule was violated. A variety of conformance checking techniques have been proposed to quantify conformance and detect deviations, based on event data and a process model (e.g., a Petri net) [8–10, 24, 27, 33, 44, 50, 86, 87, 111, 135, 145]. Also approaches based on temporal logic [31, 81, 85, 136, 149] have been proposed to check compliance.

*Forward compliance checking* techniques are based on the idea of incorporating controlling objectives within business processes, i.e., coupling CM and BPM. The drawbacks of *forward compliance checking* techniques were mentioned earlier in Sect. 1.2. **In this thesis we focus on compliance auditing (i.e., backward compliance checking) and separating CM from BPM.** If we consider the broader concept of compliance, then it does not only include compliance w.r.t. strict financial controls imposed by legislation such as SOX, BASEL II or COSO framework but also covers adherence to any desired process behavior (e.g. medical protocols in health care or even more efficient ways of executing a process). Configuring information system such that they adhere to all possible desired ways of executing a process will lead to rigid and complex business processes. We advocate the idea of keeping the design of business processes and configuration of underlying information systems simple. Limiting the inserted controls within business processes to only essential ones will increase the flexibility and agility of business processes. At the same time, we can leverage from the omnipresence of event data to do a factual analysis of compliance during and after execution of processes instead of only relying on preventive controls. We are interested to detect all possible deviations from desired process behaviors.

**Compliance improvement based on diagnostics.** Fewer works are available in the area of root-cause analysis and improving compliance based on diagnostics obtained during compliance analysis. Compliance auditing enables us to analyze the actual execution of processes rather than assumptions about the processes. In case of a deviation, we can use the contextual information of a deviation (hidden in data) to understand why a specific desired process path are not followed. The corrective measures, taken to improve business processes, then will be based on the factual diagnostics obtained during compliance checking.

The literature available on root-cause analysis of compliance violations using execution of business processes [104, 122] are mostly done in the area of Service Oriented Architecture (SOA). The authors introduce Key Compliance

Indicators (KCI) and report about compliance level of business services that computing these indicators. These studies are limited to definition of appropriate indicators and reporting on them. However, using more advanced data analysis techniques such as data mining for reasoning about root-causes of compliance violations has been explored less.

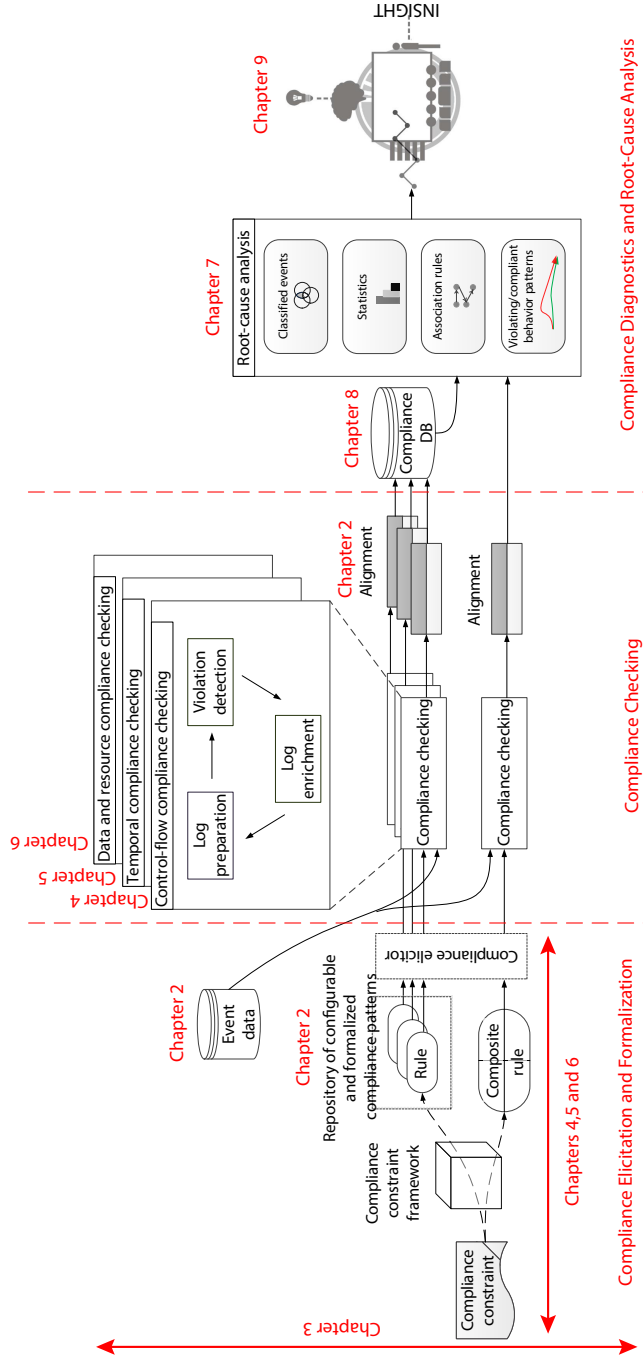


Figure 1.4: The thesis road map gives the mapping between chapters of the thesis onto our compliance analysis approach.

## 1.4 Thesis Approach and Results

This thesis contributes a novel technique for the specification of compliance constraints and data-driven analysis of process compliance. Using the research method of design science which starts with understanding requirements from literature and practice, we developed and implemented an approach and a family of techniques, each of which was evaluated and improved iteratively in various case studies. In Chap. 3, we will discuss the details of requirements for compliance analysis and sketch our ideas on the approach. The exact techniques and their implementation will be discussed in chapters 4 to 8 and Chap. 9 evaluates our approach results.

Figure 1.4) is organized in alignment with our compliance analysis approach. This approach enables compliance experts and business users to:

- Elicit and specify informal compliance rules in terms of compliance without exposing them to technicalities of any specification language. Our approach bridges the gap between informal constraints and formal rules by providing repositories of configurable patterns that capture details of rules as options for commonly-required compliance constraints. These options are configured interactively with end-users, using question trees and natural language (**addressing research goals 1 and 2**). We also explain how we can merge relevant compliance rules as a composite compliance model.
- Detect violations of different types including violations from control-flow compliance rules, temporal, data-aware, and resource aware compliance rules. We check compliance by computing *optimal alignments* between execution of a business process and the “most likely compliant path” according to a formal model of a compliance rule. Such an alignment offers detailed diagnostics compared to current approaches of compliance checking (**addressing research goals 3,4, and 5**).
- Aggregate compliance data and provide the capacity for statistical analysis. We developed a compliance database to integrate various sources of compliance data. The data structure of this database provides us the flexibility to explore compliance data from different perspectives and prepare reports based on even complex criteria (**addressing research goal 8**).
- Analyze the root-causes of violations by leveraging various data analysis techniques. We use association rule mining to detect meaningful relations

between violations and their context. The result of this analysis enables business users to identify key insights in the large amount of data. Furthermore, We use classification techniques to differentiate violating and compliant patterns in data. These patterns can be used for predicting possible future violations (**addressing research goals 6 and 7**).

As is shown in Fig. 1.4, our compliance analysis solution is organized in three main parts: *compliance elicitation and formalization*, *compliance checking*, and *compliance diagnostics*. Typically compliance constraints are decomposed into a set of compliance rules. These rules are formalized individually as atomic compliance rules or as a composite one. This choice of formulation impacts the later phases. Dedicated checking techniques allow us to detect various violation types. Different types of diagnostics about the detected violations are produced, including statistics, associations between violations and their context, and conditions when a certain violation type holds or not.

The results of this thesis have been implemented and evaluated in a large case study in collaboration with the Dutch Employee Insurance Agency (UWV). The case study was conducted using A/B testing on three business units in UWV. The result showed a remarkable improvement in compliance of business units that received feedback through our approach. In addition we received positive feedback from domain experts about the applicability of our approach and its findings.

## 1.5 Thesis Road Map

We conclude the introduction with the road map depicted in Fig. 1.4. This road map gives the mapping between the thesis chapters onto our compliance analysis approach.

**Part I.** After this introduction, Chap. 2 provides the formal background for this thesis. We introduce the general concept of *behavior* to describe *compliant behavior* specified by compliance constraints and *observed behavior* that captures executions of business processes. We introduce *alignment* to compare the observed behavior versus compliant behavior. In Chap. 3 we describe the research questions in more detail and systematically develop the requirements for each step, of, our approach based on literature and practical requirements.

**Part II.** Chapters 4, 5, 6 discuss our dedicated compliance elicitation and checking techniques for compliance constraints confining control-flow, temporal, data and organization perspectives of business processes.

**Part III.** Chapter 7 explains our techniques for doing root-cause analysis on violations and providing detailed diagnostics about violations. Chapter 8 discusses our solution for integrating various compliance data sources to enable exploring data from different views and produce various reports.

**Part IV.** In Chap. 9 we report on a case study that we conducted in collaboration with UWV (Dutch Employee Insurance Agency) to evaluate our approach and its findings. We describe the design and scope of the experiment. The procedures taken for data collection and data preparation are described. Then the execution of the experiment including the techniques used and the compliance rules that were checked will be discussed. We will elaborate on the results of the analysis and the lessons learned in the project. We summarize our contribution in Chap. 10. We conclude with open problems and ideas about future research.

The techniques developed and discussed in this thesis are implemented in the Process Mining Toolkit ProM 6.6.<sup>3</sup> The implementation of the techniques relevant for each chapter is discussed in the same chapter.

---

<sup>3</sup>available from: <http://www.promtools.org/>





# Chapter 2

## Preliminaries

As explained in Chap. 1, the main input elements to compliance checking are (1) the legal constraints to satisfy, (2) and data about past business process executions to check for compliance. Both refer to process behaviors; normative behavior that is prescribed and real behavior that has been observed. In this chapter we define “behavior” as a general concept and we use it to relate observed behavior (event logs) and prescribed behaviors (compliance constraints). We discuss particular representations for event logs and compliance constraints: the former in XES standard format and the latter as Petri nets. Both notions are well-established in literature and we will present them only to the extent necessary. We will use these concepts and definitions throughout the thesis to explain our compliance checking approach.

The general definition of *behavior* is presented in Sect. 2.1. Section 2.2 discusses event logs. Section 2.3 presents prescribed behaviors and explains how we express prescribed behaviors in terms of (*data-aware*) *Petri nets*. Section 2.4 explains the notion of *alignments* and how we compare observed behavior and prescribed behaviors. In Sect. 2.5, we describe how we can capture prescribed behavior from different perspectives ( control-flow, time, data and resource) and Sect. 2.7 concludes this chapter.

### 2.1 Behavior

An event log refers to one specific behavior describing what has happened whereas a compliance constraint describes a set of prescribed behaviors. Later,

we will align the observed behavior (event log) to the prescribed behavior (model) to identify and measure the deviations between observed and prescribed behaviors.

Figure 2.1 exemplifies an observed behavior as a collection of events in a table.

<i>event ID</i>	<i>pi</i>	<i>time</i>	<i>act</i>	<i>resource</i>
e <sub>1</sub>	p <sub>1</sub>	1	a	Nic
e <sub>2</sub>	p <sub>1</sub>	2	b	Nic
e <sub>3</sub>	p <sub>1</sub>	3	d	Ben
e <sub>4</sub>	p <sub>2</sub>	4	f	Bill
e <sub>5</sub>	p <sub>2</sub>	5	a	Sara
e <sub>6</sub>	p <sub>2</sub>	6	c	Nic
e <sub>7</sub>	p <sub>2</sub>	7	d	Ben

Figure 2.1: Example of observed behavior recorded in an event log.

Intuitively, an *event* is a record describing that a particular activity occurred in a particular context. Formally, we distinguish events by their event identifiers (the *event ID* column in Fig. 2.1). Each event refers to a particular activity: the activity name (*column 'act'*) in Fig. 2.1). An event may carry any number of attributes having particular values, for example event timestamp (indicated as *column 'time'* and resource (indicated as *column 'resource'*) in Fig. 2.1). These attributes describe the event's context. For instance, the resource (i.e., person or device) executing or initiating the activity, the timestamp of the event, or data elements recorded with the event (e.g. size of an order). Moreover, we assume that each event can be associated to a unique process instance in which the event occurred (indicated as *column 'pi'* in Fig. 2.1). One process instance (also called *case*) can be viewed as a set of *related* events. This gives rise to the following universes of identifiers, attributes, values and events.

**Definition 2.1 (Universes)**

- $\mathcal{I}$  is the set of all process instance identifiers,
- $\mathcal{Attr}$  is the set of all possible attributes,
- $\mathcal{Val}$  is the set of all possible attribute values,
- $\mathcal{E}$  is the set of all events.

As mentioned earlier, events are represented by a unique identifier. This allows us to refer to a specific event, and events with the same properties can be distinguished. We define a *behavior* as a *partial order* of events.

**Definition 2.2 (Behavior)**  $B = (E, \#, \leq)$  is a behavior when:

- $E \subseteq \mathcal{E}$  is a set of events,
- $\# : E \rightarrow (\text{Attr} \not\rightarrow \text{Val})$  maps events into a partial function assigning values to some attributes,<sup>1</sup>
- $\leq \subseteq E \times E$  defines a partial order on events ( $\leq$  is reflexive, antisymmetric, and transitive).

$BH$  is the set of all behaviors.

Figure 2.2 shows the behavior defined from the events given in Fig. 2.1. The behavior  $B = (E, \#, \leq)$  consists of seven events that are totally ordered in two process instances:

- set of events  $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ .
- $\#_{act}(e_1) = a$ ,  $\#_{act}(e_2) = b$ , etc. <sup>2</sup>
- for  $1 \leq i < 7$ ,  $\#_{time}(e_i) = t_i$  s.t.  $t_i \leq t_{i+1}$ .
- for  $i \in \{1, 2, 6\}$ ,  $\#_{resource}(e_i) = Nic$ , for  $i \in \{3, 7\}$ ,  $\#_{resource}(e_i) = Ben$ ,  $\#_{resource}(e_4) = Bill$ , and  $\#_{resource}(e_5) = Sara$ .
- $\#_{pi}(e_1) = \#_{pi}(e_2) = \#_{pi}(e_3) = p_1$  and  $\#_{pi}(e_4) = \#_{pi}(e_5) = \#_{pi}(e_6) = \#_{pi}(e_7) = p_2$ .
- $\leq$  is a partial order.

In the following, we assume any two behaviors to be disjoint on events. We write  $dom(\#(e))$  for the set of attributes having a value in  $e$ . We write  $\#_{attr}(e) = v$  if attribute  $attr \in dom(\#(e))$  has value  $v$ . We assume that each event is assigned to a process instance and formally  $pi \in dom(\#(e))$  and  $\#_{pi}(e) \in \mathcal{P}$ . For example,  $\#_{pi}(e_1) = p_1$  denotes that event  $e_1$  belongs to process instance  $p_1$ ,  $\#_{act}(e_1) = a$  denotes that  $e_1$  corresponds to the execution of activity  $a$ ,  $\#_{time}(e_1) = 1$  denotes that the event occurred at time 1, and  $\#_{resource}(e_1) = Nic$  denotes that  $Nic$  executed this activity. In general *activity name* (*act*), and *time* are also assumed to

<sup>1</sup>  $f \in X \not\rightarrow Y$  is a partial function with domain  $dom(f) \subseteq X$ .

<sup>2</sup>  $\#_a(e)$  is a shorthand for  $(\#(e))(a)$ .

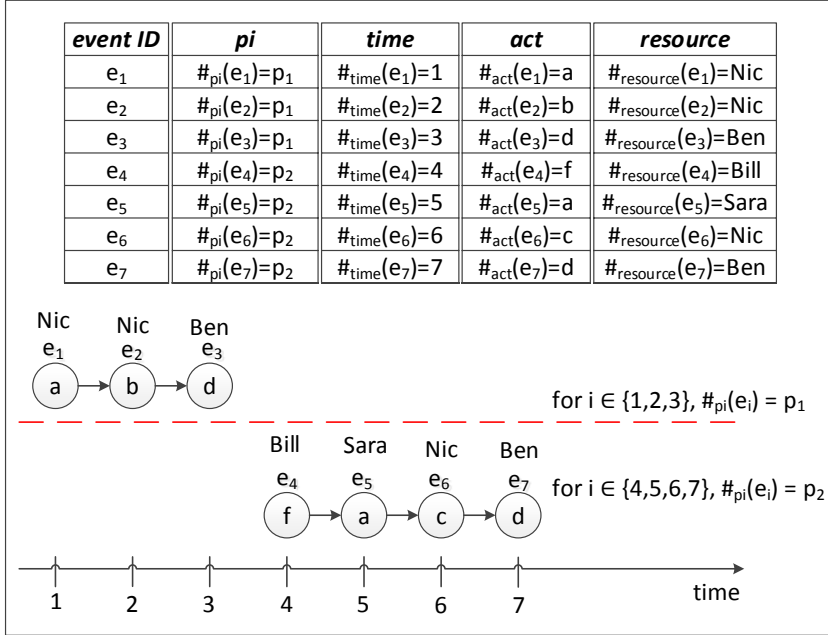


Figure 2.2: Behavior  $B = (E, \#, \leq)$  consisting of events related to two process instances  $p_1$  and  $p_2$ .

be attributes of each event. However, later in this thesis during event log enrichment with compliance diagnostics, we introduce some artificial events that may not have an *activity name* or *time* attribute.

Next, we discuss the role of the ordering relation  $\leq$  of a behavior and how it relates to other attributes of events. For events  $e_1$  and  $e_2$ , we write  $e_1 < e_2$  if and only if  $e_1 \leq e_2$  and  $e_1 \neq e_2$ .  $e_1 < e_2$  means that  $e_1$  precedes  $e_2$ . Note that,  $<$  does not necessarily define a total order based on time. Hence, even if  $e_1$  happens before  $e_2$ , it may still be the case that  $e_1 \not\prec e_2$ .

Figure 2.2 illustrates a graph of the totally ordered events in our example. The arcs shows the precedence relation between events. For example, the arc between event  $e_1$ , and  $e_2$  shows that  $e_1$  is causally related to  $e_2$ . Note that, in the Fig. 2.2, we only show the *non-transitive* pairs of the relation  $\leq$ . For example  $e_1 \leq e_3$  is not shown explicitly by an edge. According to our definition of behavior, one behavior describes many process instances together. In our

example, the events in behavior  $B$  are distributed over two process instances.

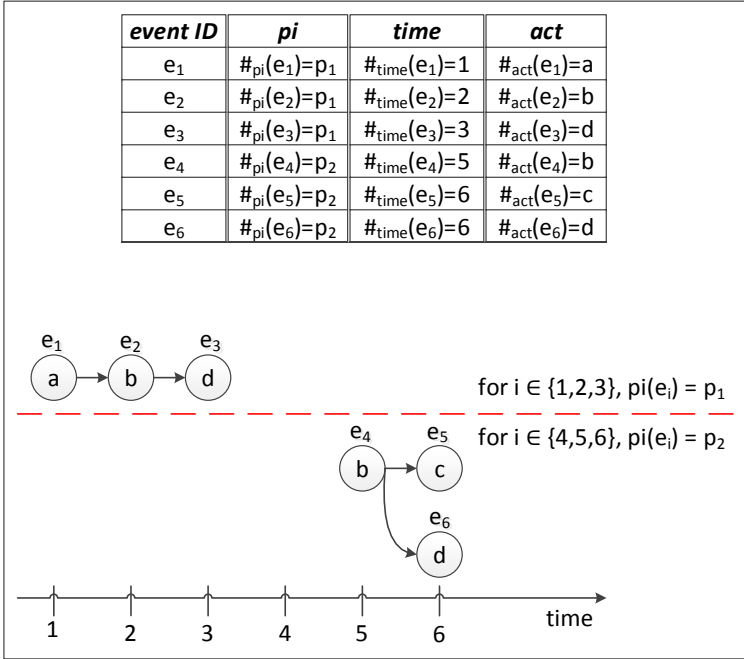


Figure 2.3: Behavior  $B' = (E', \#, \leq)$  consisting of events related to two process instances  $p_1$  and  $p_2$ .

While (Def. 2.2) allows that events in different process instances may also be ordered, in this thesis, we only consider the orderings within process instances. For example, in Fig. 2.2, although event  $e_4$  occurs time-wise strictly after event  $e_3$  (time 4 and time 3, respectively), both events are unordered as they belong to different process instances.

Another example shown in Fig. 2.3 denotes a behavior where events within the same process instance are partially ordered (interested reader is referred to [78] for detailed discussion on partial ordering of events). The behavior  $B' = (E', \#, \leq)$  shown in this figure exemplifies the general nature of Def. 2.2 with three events totally ordered in process instance  $p_1$  and three events partially ordered in process instance  $p_2$ . As shown in Fig. 2.3, the events  $e_5$  and  $e_6$  in process instance  $p_2$  are partially ordered and both occurred at time 6. In princi-

ple, one can leave out explicit timestamps and order events based on a different attribute (e.g. data dependency). If an event *writes* a value on an attribute that is *read* by another event, these two events are related such that the second event is dependent on the first event. In this case two independent events that occur at different points in time, may be partially ordered. In the scope of this thesis, we consider the ordering of events based on their timestamps only.

## 2.2 Expressing Classical Event Logs as Behaviors

Existing literature typically formalizes a process instance as a sequence of events that occurred in that instance and an event log as a set of process instances. Events are ordered based on their time of occurrence and often a total order is assumed on all events of a process instance, whereas events from different process instances are *not* ordered. We adopt this view for our work by imposing constraints on the partial ordering of events.

**Definition 2.3 (Event Logs)**  $L = (E, \#, \leq)$  is an event log if

- $L \in BH$ , i.e.,  $L$  is a behavior,
- $E$  is finite, and every event is also finite in the number of attributes,
- $\forall e_1, e_2 \in E, e_1 \leq e_2 \Rightarrow \#_{time}(e_1) \leq \#_{time}(e_2) \wedge \#_{pi}(e_1) = \#_{pi}(e_2)$ .

The behaviour  $B$  shown in Fig. 2.2 is an event log. Note that, in the classical definition of event logs the relation  $\leq$  on events is total for all events within the same process instance, i.e., for all  $e_1, e_2 \in E$  and  $\#_{pi}(e_1) = \#_{pi}(e_2)$ :  $e_1 = e_2$ ,  $e_1 < e_2$  or  $e_2 < e_1$ .

**Tree view representation of event logs.** Event logs, as they occur in practice and research, may be represented in various forms. Every system architecture that includes some sort of logging mechanism has so far developed their own, insular solution for this task. In many parts of this thesis we will use XES standard [57] for presenting event logs. XES is XML-based and is extensively used in the process mining area. XES presents event logs in a hierarchical structure with a log being the first-level object. A log then contains an arbitrary number of process instance objects and every process instance contains an arbitrary number of events. Figure 2.4 shows the behavior  $B$  of Fig. 2.2 in the hierarchical structure of an event log, subsequently called *tree-view* of an event log.

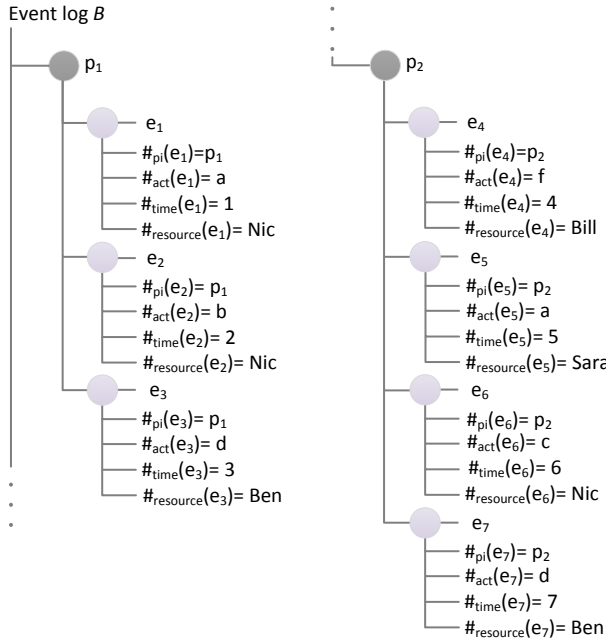


Figure 2.4: Tree-view representation of the event log  $B$  consisting of two process instances  $p_1$  and  $p_2$ .

As shown in this figure, event log  $B$  has two process instances consisting of in total seven events. Events that have  $p_1$  as the value for their  $pi$  attribute are grouped as the child nodes of this process instance and those that have  $p_2$  as the value for their  $pi$  attribute, are grouped under  $p_2$ . Each event has several attributes ( $act$ ,  $time$ , and  $resource$ ) that together form child nodes of that event.

### 2.3 Prescribed Behaviors

In the previous section we discussed how a finite “observed behavior” captures an event log. Next, we discuss how to understand a possible infinite set of behaviors as the prescribed behaviors, and how to use a finite model to specify an infinite set of behaviors.

There are various rigorous and mathematical formalisms for representing



prescribed behaviors using different specification languages, for example Formal Contractual Language (FCL) [53] from deontic logic family and Linear Temporal Logic (LTL) [91] and Computational Tree Logic (CTL) [25] from temporal logic family.

We use Petri nets to specify compliance constraints. Our choice of Petri nets as the formalization language for representation of prescribed behavior has several reasons that we will discuss in Chap. 3. No matter what formalization is chosen, it must precisely specify what is allowed and what is not allowed as a compliant behavior. It should allow for all possible compliant behaviors, while excluding each non-compliant behavior. Next, we will give a short introduction to Petri nets and we will discuss how we formalize compliant behaviors using variants of Petri nets.

### 2.3.1 Introduction to Petri nets

Petri net is a process modelling technique that allows us to describe a system behavior. The initial idea of Petri nets was introduced by Carl Adam Petri in 1962, however initial ideas were not supported by a graphical notation. This notion evolved over several years and the first actual graphical Petri nets came up in 1970's.

Petri nets offer a formal but also a graphical notation and therefore are accessible for non-expert. In this section, we present the basic concepts of Petri nets using some examples. The interested reader is referred to [141] for the formalism and technical details.

**Classical Petri nets.** Suppose the following simple loan application process in a bank. The process starts with receiving a loan application. The application is checked for credit history. Based on the credit check history, the bank decides to grant the loan to the applicant or reject the application. In case the application is accepted, after the collection of required guarantees, the payment process starts. In case of rejection, the applicant will be informed about the decision and the decision is archived as well. The two activities (archive the decision and inform client) may occur in any sequence. Finally, the case will be closed and either way, the process ends.

The Petri net shown in Fig. 2.5 models this process. A Petri net is a graph that is composed of two types of nodes: *places* and *transitions* that can be mutually connected by arcs. A *place* shows the local state of a system. Places are illustrated as *circles* in a Petri net structure. A *transition* represents a change in the local state of a system. Transitions are illustrated as *boxes* in a Petri net

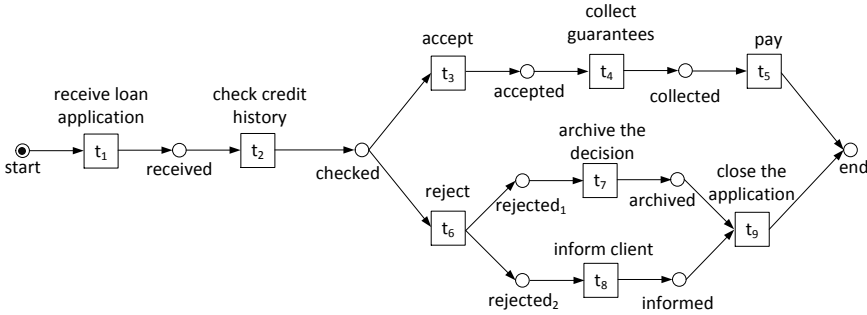


Figure 2.5: The loan process modelled as a Petri net.

structure.

The loan application is initially in the local state *start*. The execution of the transition *receive loan application*, changes the local state to *received*. By executing the *check credit history* transition, the application will be in the local state *checked*. If the application gets accepted, it reaches the local state *accepted*. After *collecting guarantees*, the application will be in the local state *collected*. After the execution of the transition *pay*, the application reaches the final state *end*. Alternatively, an application may be rejected and therefore the application will be in the local states *rejected<sub>1</sub>* and *rejected<sub>2</sub>*. At this moment, the transitions *archive decision* and *inform client* may be executed in any order. Consequently, the application will be in the local states *archived*, and *informed*. Finally, after *closing* the application, it will reach the final state *end*.

In this example, the possible local states are: *start*, *received*, *checked*, *accepted*, *rejected<sub>1</sub>*, *rejected<sub>2</sub>*, *collected*, *archived*, *informed*, or *end*. The possible state transitions are: *receive loan application* ( $t_1$ ), *check credit history* ( $t_2$ ), *accept* ( $t_3$ ), *collect guarantees* ( $t_4$ ), *pay* ( $t_5$ ), *reject* ( $t_6$ ), *archive the decision* ( $t_7$ ), *inform client* ( $t_8$ ), and *close the application* ( $t_9$ ) where  $\{t_1, \dots, t_9\}$  denotes the set of transition identifiers of the example Petri net, and  $\{\textit{receive loan application}, \textit{check credit history}, \textit{accept}, \textit{collect guarantees}, \textit{pay}, \textit{reject}, \textit{archive the decision}, \textit{inform client}, \textit{close the application}\}$  is the set of *transition labels* of the example net. Sometimes in this thesis for comprehensibility, we may refer to a transition by its label instead of its identifier. This is only in case transitions have unique labels. However, in principle different transitions with distinct identifiers may have the same label. In this case, we will refer to a transition by its identifier or simply add an index to its label to differentiate duplicate labels. For instance,

if two transitions with distinct identifiers both have the same label  $a$ , we may refer to these transitions as  $a_1$  and  $a_2$ .

A token in the place *start* models a case (an application) that was submitted to the bank. The network structure of a Petri net is fixed. The distribution of tokens over the places, however, can be changed by transitions. The *marking* of a Petri net is determined by the distribution of tokens over the places of the net. For instance, if there is a token in place *received*, the marking  $M$  of the net will be the place *received*. Note that, only places can contain tokens, not transitions. Transitions are the active components of a Petri net structure because they can change the marking when they are executed. Execution of a transition (we call it *firing* of a transition) is bound to rules. A transition may fire only if there is a token in each of its input place(s)<sup>3</sup>. When it fires, it consumes one token from each input place and produces a token in each output place. For example, the transition *receive loan application* may only be executed if there is a token in place *start* (i.e., its input place). Execution of the transition *receive loan application* consumes the token from the place *start* and produces a token in its output place *checked*. Transitions may have arbitrarily many inputs and output places or even have common input and output places. Nevertheless, each transition may fire only if there is a token in all of its input places. For example, both *reject* and *accept* have the place *checked* as their input place. That is, if there is a token in place *checked*, only one of the two transitions may fire. After a transition is fired, it produces a token in all of its output places. Transition *reject* has two output places, i.e., if it fires, it produces tokens for both of its output places and consequently both transitions *archive the decision*, and *inform client* may fire in any order.

A Petri net has an initial marking and a final marking. In the loan process example, the initial marking of the net is  $M_{initial} = [start]$ , and the final marking is  $M_{final} = [end]$ . A *firing sequence* of a Petri net is a sequence of firings (of its transitions). A **terminating firing sequence** of a Petri net with the initial marking  $M_{initial}$  is a firing sequence that starts at the *initial marking*  $M_{initial}$  and lead to its *final marking*  $M_{final}$ . There exists three distinct terminating firing sequences for the example Petri net (Fig 2.5) that will change the initial marking  $M_{initial}$  to the final marking  $M_{final}$ : (1)  $\langle receive\ loan\ application, check\ credit\ history, accept, collect\ guarantees, pay \rangle$ , (2)  $\langle receive\ loan\ application, check\ credit\ history, reject, archive\ decision, inform\ client, close\ application \rangle$ , and (3)  $\langle receive\ loan\ application, check\ credit\ history, reject, inform\ client, archive\ decision, close$

---

<sup>3</sup>A place  $p$  is an input place of a transition  $t$  if there is an arc from  $p$  to  $t$ . Likewise, a place is an output place of a transition if there is an arc from  $t$  to  $p$ .

application }.

In the example shown above, we used Petri nets to model the control-flow perspective of a process. Processes in complex information systems may have many more properties that need to be modelled as well to better describe the behavior of a process. For example the choice between firing the transition *accept* or *pay* in the above mentioned example can be made based on the result of the credit history check. In case the result is *OK*, then the application gets accepted and if *NOK*, it gets rejected. *Data-aware Petri nets* are a variant of Petri nets that allow us to adequately model such situations. A data-aware Petri net [34, 121] extends classical Petri nets with data. Using data-aware Petri nets, we can model when a transition is allowed to be executed based on some data.

Data-aware Petri nets are different from colored petri nets (CPNs) [141]. In colored Petri nets tokens in places have values. However, tokens in data-aware Petri nets do not carry values (i.e., data-aware Petri nets only have black tokens). However, the events produced as a result of firing a transition in a data-aware Petri net have variables. Values stored in these variables. Similar to any imperative programming language, values can be read from and written in variables.

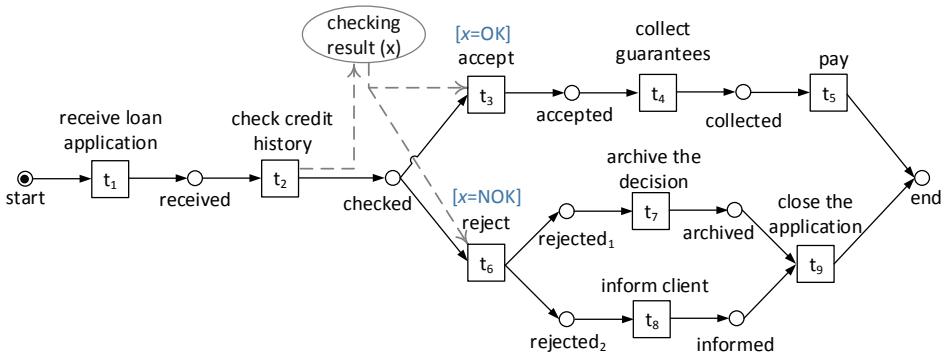


Figure 2.6: A data-aware Petri net modelling the control-flow and data perspective of the loan process.

**Petri net with data.** A Petri net with data is a Petri net in which transitions can *read* and *write* variables. A transition is allowed to write (or update) a predefined subset of process variables. A transition can have a data-dependent guard that prevents enabling of that transition if the guard evaluates to false.

Only if the guard evaluates to true and all input places have sufficiently many tokens, a transition can fire. A guard can be a boolean expression over the set of variables. In the net, boolean expressions can be combined using the standard logical operators such as conjunction ( $\wedge$ ), disjunction ( $\vee$ ), and negation ( $\neg$ ).

Figure 2.6 shows the example loan process modelled as a data-aware Petri net. In this Petri net, the variable  $x$  stores the result of credit history check. The ellipse shown in this net models this variable. The connections between this ellipse and the transitions indicate the transitions that are allowed to *read* (dashed arrow from variable to transition) or *update* (*write*) a value in variable  $x$  (dashed arrow from the transition to variable). As is shown in Fig. 2.6, the transition *check credit history* can *update* variable  $x$  and transitions *accept* and *reject*, read a value from variable  $x$ . The transitions *accept*, and *reject* are guarded and they may fire only if their guard evaluates to true. Therefore, the admissible *firing sequences* of the data-aware Petri net in Fig. 2.6 is as follows: (1)  $\langle t_1, t_2(\text{write } x = \text{OK}), t_3(\text{read } x = \text{OK}), t_4, t_5 \rangle$ , (2)  $\langle t_1, t_2(\text{write } x = \text{OK}), t_6(\text{read } x = \text{NOK}), t_6, t_7, t_8, t_9 \rangle$ , and (3)  $\langle t_1, t_2(\text{write } x = \text{OK}), t_6(\text{read } x = \text{NOK}), t_6, t_8, t_7, t_9 \rangle$

### 2.3.2 Behavioral Notions: Process Instance Run, Run, and Model

The set of all possible firing sequences of a (data-aware) Petri net can be specified as a prescribed behavior. Next, we introduce the notion of “*process instance run*” that corresponds to the notion of “*firing sequence*” in a (data-aware) Petri net. Then, we discuss how several *process instance runs* together yield one behavior and finally the set of all such behaviors gives the complete semantics of a (data-aware) Petri net.

#### Definition 2.4 (Process Instance Run)

$P$  is a process instance run if:

- $P$  is a behavior, i.e.,  $P = (E, \#, \leq)$ , and
- there exists a  $p \in \mathcal{I}$  such that  $\forall e \in E, \#_{pi}(e_i) = p$ , i.e., all the events in  $P$  have the same value for their process instance attribute.

We express each **terminating firing sequence** in a (data-aware) Petri net as a *process instance run*. Each *terminating firing sequence* of a (data-aware) Petri net  $N$  defines a *process instance run*  $P = (E, \#, \leq)$ . We write  $P(N)$  for the set of all *process instance runs* of a (data-aware) Petri net  $N$ . **Note that we only consider the terminating firing sequences of a data-aware Petri net as**

a *process instance run*. Each event in a *process instance run* is produced as a result of *firing* a transition. Later in Sect. 2.3.3, we will discuss how attributes of an event  $e \in E$  and  $P \in P(N)$  are derived from properties of a transition in  $N$ .

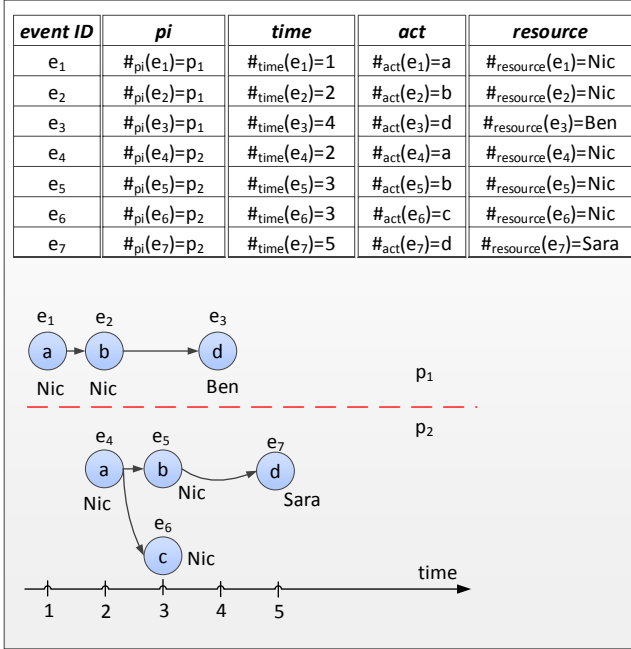


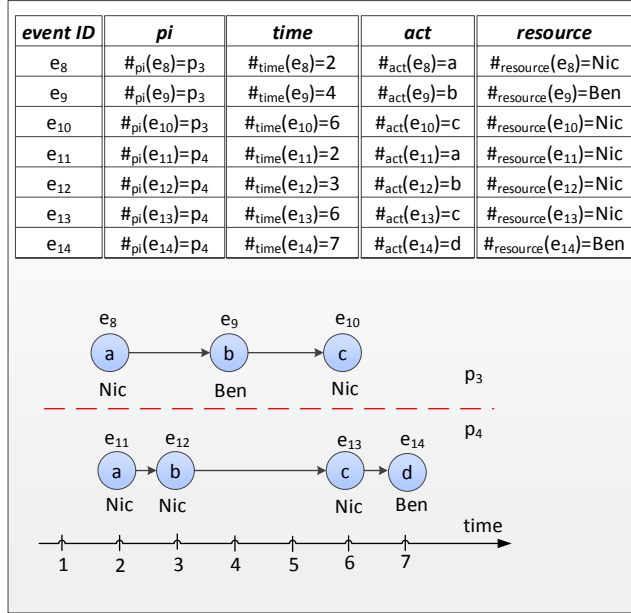
Figure 2.7: Events in run  $R_1$  are partially ordered.

**Definition 2.5 (Run)**

Let  $\{P^1, \dots, P^n\}$  be a set of process instance runs with  $P^i = (E^i, \#^i, \leq^i)$ . Then the behavior  $R = (E^1 \cup \dots \cup E^n, \#^1 \cup \dots \cup \#^n, \leq^1 \cup \dots \cup \leq^n)$  is a run (involving process instance runs  $P^1, \dots, P^n$ ). Note that,  $R$  is a behavior; i.e.,  $R \in BH$ .

A (data-aware) Petri net  $N$  may have infinitely many different *process instance runs*. In this case,  $N$  has also infinitely many different runs. We write  $R(N)$  for the set of all runs of  $N$ .

**Definition 2.6 (Model)** A Model  $Mod$  is a set of runs, i.e.,  $Mod \subseteq BH$

Figure 2.8: Events in run  $R_2$ .

A (data-aware) Petri net  $N$  defines the *Model*  $Mod(N) = R(N)$ , i.e., the set of all *runs* of  $N$  where each  $R \in Mod(N)$  is the union of several *process instance runs* of net  $N$ .

This rather unusual notion of a model will be used for aligning a log  $L$  to the most similar *run*  $R \in Mod(N)$ , i.e., the *set* of process executions specified by the model which match best the set of process executions recorded in the log.

Technically, we will identify a set  $R$  of process instance runs of  $N$  so that there is a one-to-one mapping between the process instances in  $L$  and process instance runs in  $R$ . We call  $R$  a *run* of  $N$  and formalize it as a single behavior.

Figures 2.7 and 2.8 exemplify two prescribed behaviors as two *runs*  $R_1$ , and  $R_2$ . Each contains the events of two different *process instance runs*, i.e.,  $R_1$  is a *run* obtained from  $p_1$  and  $p_2$ , and  $R_2$  is a *run* obtained from  $p_3$  and  $p_4$ . Similar to an event log a *run* is a collection of events that are partially ordered. In these examples, *runs*  $R_1$  and  $R_2$  have seven events each, distributed over four process instance runs  $p_1$ ,  $p_2$ ,  $p_3$ , and  $p_4$ . Each event has several attributes such as *time*,

and *resource*.

In essence, each of the process instance runs in  $R_1$  is a *run* (behavior) itself where all the events in this run have the same value for their *process instance* attribute  $pi$ . Recall from Definition 2.6 that a *Model* is a set of behaviors and may include many runs. Next, we will show how we can describe all possible *runs* of a *Model* as a (*data-aware*) *Petri net*.

### 2.3.3 Precise Specification of a Particular Model as a (Data-Aware) Petri net

As we discussed earlier, we can present prescribed behavior in terms of (data-aware) Petri nets. For instance, consider the data-aware Petri net  $DPN$  in Fig. 2.9.  $DPN$  models exactly the prescribed behaviors  $R'_1$  and  $R'_2$  shown in Fig. 2.10. Note that, the runs  $R'_1$  and  $R'_2$  are the same as the runs  $R_1$  and  $R_2$  shown in Fig. 2.7 and Fig. 2.8 with the difference that events in  $R'_1$  and  $R'_2$  have two additional attributes  $t_{ID}$  and  $label_N$ . Note that, attribute  $label_N$  has exactly the same value as the attribute  $act$ .



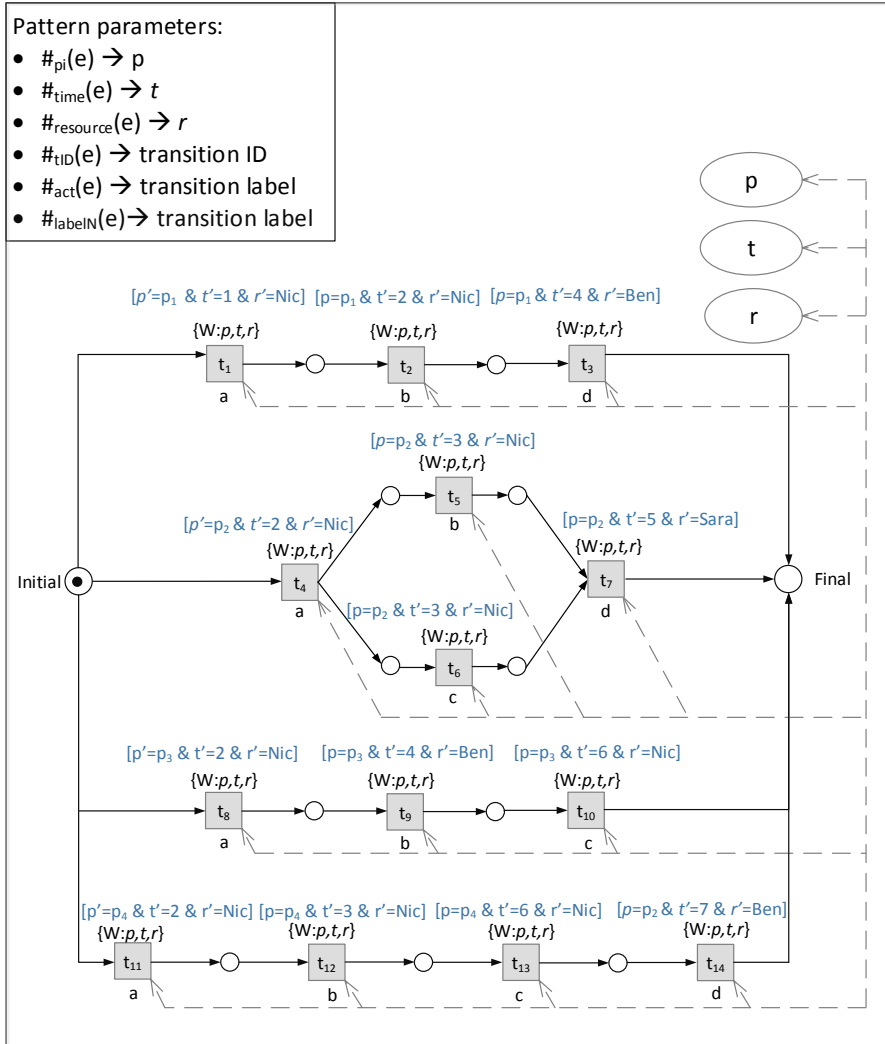


Figure 2.9: The data-aware Petri net *DPN* specifying four firing sequences  $p_1, p_2, p_3,$  and  $p_4$ .

<i>event ID</i>	<i>t<sub>ID</sub></i>	<i>label<sub>N</sub></i>	<i>pi</i>	<i>time</i>	<i>act</i>	<i>resource</i>
e <sub>1</sub>	#tID(e <sub>1</sub> )=t <sub>1</sub>	#labelN(e <sub>1</sub> )=a	#p <sub>i</sub> (e <sub>1</sub> )=p <sub>1</sub>	#time(e <sub>1</sub> )=1	#act(e <sub>1</sub> )=a	#resource(e <sub>1</sub> )=Nic
e <sub>2</sub>	#tID(e <sub>2</sub> )=t <sub>2</sub>	#labelN(e <sub>2</sub> )=b	#p <sub>i</sub> (e <sub>2</sub> )=p <sub>1</sub>	#time(e <sub>2</sub> )=2	#act(e <sub>2</sub> )=b	#resource(e <sub>2</sub> )=Nic
e <sub>3</sub>	#tID(e <sub>3</sub> )=t <sub>3</sub>	#labelN(e <sub>3</sub> )=d	#p <sub>i</sub> (e <sub>3</sub> )=p <sub>1</sub>	#time(e <sub>3</sub> )=4	#act(e <sub>3</sub> )=d	#resource(e <sub>3</sub> )=Ben
e <sub>4</sub>	#tID(e <sub>4</sub> )=t <sub>4</sub>	#labelN(e <sub>4</sub> )=a	#p <sub>i</sub> (e <sub>4</sub> )=p <sub>2</sub>	#time(e <sub>4</sub> )=2	#act(e <sub>4</sub> )=a	#resource(e <sub>4</sub> )=Nic
e <sub>5</sub>	#tID(e <sub>5</sub> )=t <sub>5</sub>	#labelN(e <sub>5</sub> )=b	#p <sub>i</sub> (e <sub>5</sub> )=p <sub>2</sub>	#time(e <sub>5</sub> )=3	#act(e <sub>5</sub> )=b	#resource(e <sub>5</sub> )=Nic
e <sub>6</sub>	#tID(e <sub>6</sub> )=t <sub>6</sub>	#labelN(e <sub>6</sub> )=c	#p <sub>i</sub> (e <sub>6</sub> )=p <sub>2</sub>	#time(e <sub>6</sub> )=3	#act(e <sub>6</sub> )=c	#resource(e <sub>6</sub> )=Nic
e <sub>7</sub>	#tID(e <sub>7</sub> )=t <sub>7</sub>	#labelN(e <sub>7</sub> )=d	#p <sub>i</sub> (e <sub>7</sub> )=p <sub>2</sub>	#time(e <sub>7</sub> )=5	#act(e <sub>7</sub> )=d	#resource(e <sub>7</sub> )=Sara
e <sub>8</sub>	#tID(e <sub>8</sub> )=t <sub>8</sub>	#labelN(e <sub>8</sub> )=a	#p <sub>i</sub> (e <sub>8</sub> )=p <sub>3</sub>	#time(e <sub>8</sub> )=2	#act(e <sub>8</sub> )=a	#resource(e <sub>8</sub> )=Nic
e <sub>9</sub>	#tID(e <sub>9</sub> )=t <sub>9</sub>	#labelN(e <sub>9</sub> )=b	#p <sub>i</sub> (e <sub>9</sub> )=p <sub>3</sub>	#time(e <sub>9</sub> )=4	#act(e <sub>9</sub> )=b	#resource(e <sub>9</sub> )=Ben
e <sub>10</sub>	#tID(e <sub>10</sub> )=t <sub>10</sub>	#labelN(e <sub>10</sub> )=c	#p <sub>i</sub> (e <sub>10</sub> )=p <sub>3</sub>	#time(e <sub>10</sub> )=6	#act(e <sub>10</sub> )=c	#resource(e <sub>10</sub> )=Nic
e <sub>11</sub>	#tID(e <sub>11</sub> )=t <sub>11</sub>	#labelN(e <sub>11</sub> )=a	#p <sub>i</sub> (e <sub>11</sub> )=p <sub>4</sub>	#time(e <sub>11</sub> )=2	#act(e <sub>11</sub> )=a	#resource(e <sub>11</sub> )=Nic
e <sub>12</sub>	#tID(e <sub>12</sub> )=t <sub>12</sub>	#labelN(e <sub>12</sub> )=b	#p <sub>i</sub> (e <sub>12</sub> )=p <sub>4</sub>	#time(e <sub>12</sub> )=3	#act(e <sub>12</sub> )=b	#resource(e <sub>12</sub> )=Nic
e <sub>13</sub>	#tID(e <sub>13</sub> )=t <sub>13</sub>	#labelN(e <sub>13</sub> )=c	#p <sub>i</sub> (e <sub>13</sub> )=p <sub>4</sub>	#time(e <sub>13</sub> )=6	#act(e <sub>13</sub> )=c	#resource(e <sub>13</sub> )=Nic
e <sub>14</sub>	#tID(e <sub>14</sub> )=t <sub>14</sub>	#labelN(e <sub>14</sub> )=d	#p <sub>i</sub> (e <sub>14</sub> )=p <sub>4</sub>	#time(e <sub>14</sub> )=7	#act(e <sub>14</sub> )=d	#resource(e <sub>14</sub> )=Ben

Figure 2.10: Prescribed behaviors specified by DPN.

Each unique event in the runs  $R'_1$ , and  $R'_2$  is presented as a transition in DPN. The ellipses connected to each transition represent three variables  $p$ ,  $t$ , and  $r$ . Each of these variables store the values of attributes of the events in  $R'_1$ , and  $R'_2$ . Three variables  $p$ ,  $t$ , and  $r$  in DPN respectively capture the values for attributes  $pi$ ,  $time$ , and  $resource$ . This is denoted by the *write statement*  $\{W : p, t, r\}$  on each transition. The additional attributes of events in Fig. 2.10 ( $tID$ , and  $act$ ) correspond to *transition ID* and *transition label* in DPN. Since the  $label_{NT}$  has exactly the same values of attribute  $act$ , the values for attribute  $label_{NT}$  is also captured by the *transition label* in DPN.

As can be seen, the control-flow of DPN precisely models the events and their sequences according to  $R'_1$  and  $R'_2$ . Each process instance run in  $R'_1$  and  $R'_2$  corresponds to a *terminating firing sequence* in DPN. As soon as any of the  $a$ -labelled transitions in DPN fires, it sets a value for process instance attribute ( $pi$ ), which is stored in variable  $p$ . Similarly the values for attributes  $time$ , and  $resource$  are written and captured in variables  $t$  and  $r$ . The variables get updated as other transitions fire in a terminating firing sequence. However, the value of variable  $p$  is only read by these other transitions. Note that,  $p$ ,  $t$ , and  $r$  denote respectively the *read* values for attributes  $pi$ ,  $time$ , and  $resource$  at each transition, and  $p'$ ,  $t'$ , and  $r'$  denote the *written* value for these attributes. The terminating firing sequences in DPN are limited to the process instance runs shown in Fig. 2.10. The guards at transitions in DPN exclude other events with

other attribute values than events and attribute values specified in Fig. 2.10.

**Silent transitions and duplicate labels in Petri nets.** As we mentioned in Sect. 2.3.1, we can refer to transitions by their label instead of their identifiers as long as the labels are unique. In case distinct transitions in a net have the same label, we differentiate transitions by adding an index to their label. For example, in  $DPN'$  shown in Fig. 2.11 transition identifiers are replaced by the transition labels.

Compared to  $DPN$  in Fig. 2.9, two transition with label  $\tau$  have been added in  $DPN'$ . Firing  $\tau$ -labelled transitions also results in an event in the corresponding run with a value for the “*act*” attribute.  $DPN'$  models the runs  $R_1''$  and  $R_2''$  (shown in Fig. 2.12).

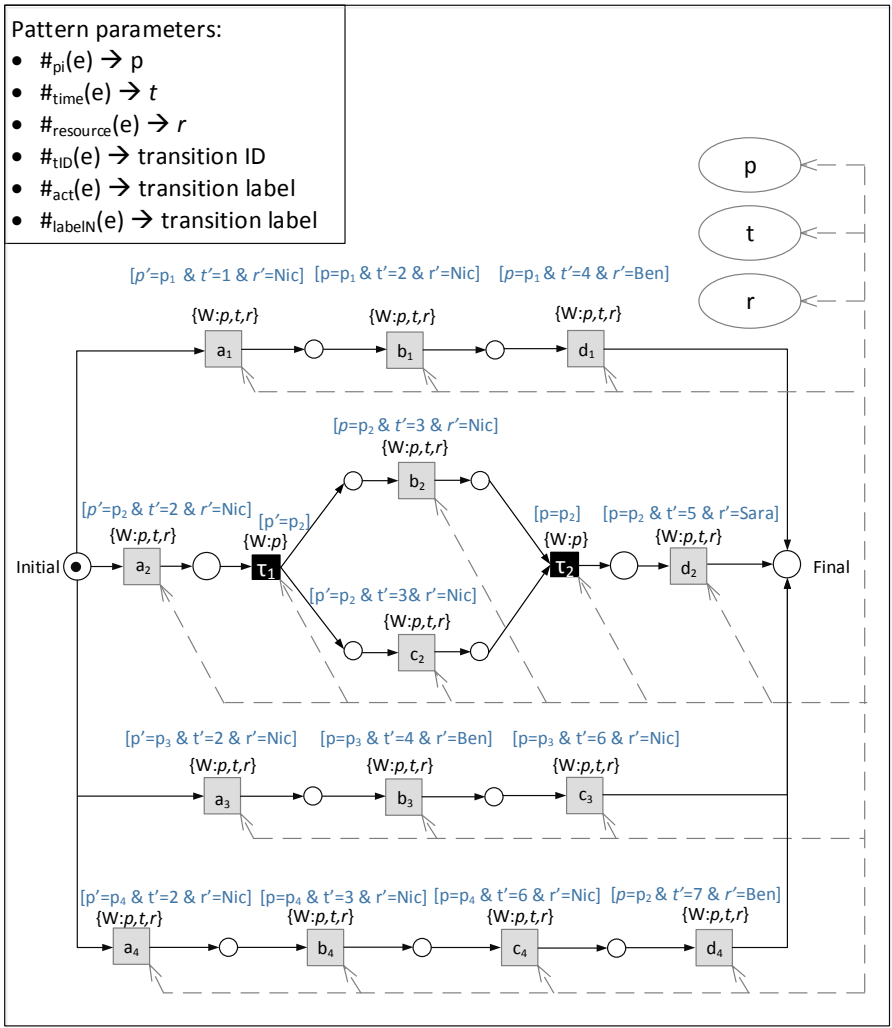


Figure 2.11: The data-aware Petri net  $DPN'$  specifying  $R_1''$ , and  $R_2''$ .

The run  $R''_1$  has extended  $R'_1$  (Fig. 2.10) with two events. These events and their attributes are hachured in Fig. 2.12. These events are referring to the two  $\tau$ -labelled (so called *silent* or *invisible*) transitions in  $DPN'$ . These transition do not represent any business activity and are usually used for modelling purposes in Petri nets (e.g. for modelling concurrency or similar structures). Note that when ignoring the technical attribute “ $tID$ ” and projecting only on events with visible activity names (non- $\tau$ ), then  $R_1$ ,  $R'_1$ , and  $R''_1$  express the same behavior.

<i>event ID</i>	<i>t<sub>ID</sub></i>	<i>label<sub>N</sub></i>	<i>pi</i>	<i>time</i>	<i>act</i>	<i>resource</i>
e <sub>1</sub>	# <sub>tID</sub> (e <sub>1</sub> )=t <sub>1</sub>	# <sub>labelN</sub> (e <sub>1</sub> )=a	# <sub>pi</sub> (e <sub>1</sub> )=p <sub>1</sub>	# <sub>time</sub> (e <sub>1</sub> )=1	# <sub>act</sub> (e <sub>1</sub> )=a	# <sub>resource</sub> (e <sub>1</sub> )=Nic
e <sub>2</sub>	# <sub>tID</sub> (e <sub>2</sub> )=t <sub>2</sub>	# <sub>labelN</sub> (e <sub>2</sub> )=b	# <sub>pi</sub> (e <sub>2</sub> )=p <sub>1</sub>	# <sub>time</sub> (e <sub>2</sub> )=2	# <sub>act</sub> (e <sub>2</sub> )=b	# <sub>resource</sub> (e <sub>2</sub> )=Nic
e <sub>3</sub>	# <sub>tID</sub> (e <sub>3</sub> )=t <sub>3</sub>	# <sub>labelN</sub> (e <sub>3</sub> )=d	# <sub>pi</sub> (e <sub>3</sub> )=p <sub>1</sub>	# <sub>time</sub> (e <sub>3</sub> )=4	# <sub>act</sub> (e <sub>3</sub> )=d	# <sub>resource</sub> (e <sub>3</sub> )=Ben
e <sub>4</sub>	# <sub>tID</sub> (e <sub>4</sub> )=t <sub>4</sub>	# <sub>labelN</sub> (e <sub>4</sub> )=a	# <sub>pi</sub> (e <sub>4</sub> )=p <sub>2</sub>	# <sub>time</sub> (e <sub>4</sub> )=2	# <sub>act</sub> (e <sub>4</sub> )=a	# <sub>resource</sub> (e <sub>4</sub> )=Nic
e <sub>5</sub>	# <sub>tID</sub> (e <sub>5</sub> )=t <sub>5</sub>	# <sub>labelN</sub> (e <sub>5</sub> )=b	# <sub>pi</sub> (e <sub>5</sub> )=p <sub>2</sub>	# <sub>time</sub> (e <sub>5</sub> )=3	# <sub>act</sub> (e <sub>5</sub> )=b	# <sub>resource</sub> (e <sub>5</sub> )=Nic
e <sub>6</sub>	# <sub>tID</sub> (e <sub>6</sub> )=t <sub>6</sub>	# <sub>labelN</sub> (e <sub>6</sub> )=c	# <sub>pi</sub> (e <sub>6</sub> )=p <sub>2</sub>	# <sub>time</sub> (e <sub>6</sub> )=3	# <sub>act</sub> (e <sub>6</sub> )=c	# <sub>resource</sub> (e <sub>6</sub> )=Nic
e <sub>7</sub>	# <sub>tID</sub> (e <sub>7</sub> )=t <sub>7</sub>	# <sub>labelN</sub> (e <sub>7</sub> )=d	# <sub>pi</sub> (e <sub>7</sub> )=p <sub>2</sub>	# <sub>time</sub> (e <sub>7</sub> )=5	# <sub>act</sub> (e <sub>7</sub> )=d	# <sub>resource</sub> (e <sub>7</sub> )=Sara
e <sub>15</sub>	# <sub>tID</sub> (e <sub>15</sub> )=t <sub>15</sub>	-	# <sub>pi</sub> (e <sub>15</sub> )=p <sub>2</sub>	-	-	-
e <sub>16</sub>	# <sub>tID</sub> (e <sub>16</sub> )=t <sub>16</sub>	-	# <sub>pi</sub> (e <sub>16</sub> )=p <sub>2</sub>	-	-	-
e <sub>8</sub>	# <sub>tID</sub> (e <sub>8</sub> )=t <sub>8</sub>	# <sub>labelN</sub> (e <sub>8</sub> )=a	# <sub>pi</sub> (e <sub>8</sub> )=p <sub>3</sub>	# <sub>time</sub> (e <sub>8</sub> )=2	# <sub>act</sub> (e <sub>8</sub> )=a	# <sub>resource</sub> (e <sub>8</sub> )=Nic
e <sub>9</sub>	# <sub>tID</sub> (e <sub>9</sub> )=t <sub>9</sub>	# <sub>labelN</sub> (e <sub>9</sub> )=b	# <sub>pi</sub> (e <sub>9</sub> )=p <sub>3</sub>	# <sub>time</sub> (e <sub>9</sub> )=4	# <sub>act</sub> (e <sub>9</sub> )=b	# <sub>resource</sub> (e <sub>9</sub> )=Ben
e <sub>10</sub>	# <sub>tID</sub> (e <sub>10</sub> )=t <sub>10</sub>	# <sub>labelN</sub> (e <sub>10</sub> )=c	# <sub>pi</sub> (e <sub>10</sub> )=p <sub>3</sub>	# <sub>time</sub> (e <sub>10</sub> )=6	# <sub>act</sub> (e <sub>10</sub> )=c	# <sub>resource</sub> (e <sub>10</sub> )=Nic
e <sub>11</sub>	# <sub>tID</sub> (e <sub>11</sub> )=t <sub>11</sub>	# <sub>labelN</sub> (e <sub>11</sub> )=a	# <sub>pi</sub> (e <sub>11</sub> )=p <sub>4</sub>	# <sub>time</sub> (e <sub>11</sub> )=2	# <sub>act</sub> (e <sub>11</sub> )=a	# <sub>resource</sub> (e <sub>11</sub> )=Nic
e <sub>12</sub>	# <sub>tID</sub> (e <sub>12</sub> )=t <sub>12</sub>	# <sub>labelN</sub> (e <sub>12</sub> )=b	# <sub>pi</sub> (e <sub>12</sub> )=p <sub>4</sub>	# <sub>time</sub> (e <sub>12</sub> )=3	# <sub>act</sub> (e <sub>12</sub> )=b	# <sub>resource</sub> (e <sub>12</sub> )=Nic
e <sub>13</sub>	# <sub>tID</sub> (e <sub>13</sub> )=t <sub>13</sub>	# <sub>labelN</sub> (e <sub>13</sub> )=c	# <sub>pi</sub> (e <sub>13</sub> )=p <sub>4</sub>	# <sub>time</sub> (e <sub>13</sub> )=6	# <sub>act</sub> (e <sub>13</sub> )=c	# <sub>resource</sub> (e <sub>13</sub> )=Nic
e <sub>14</sub>	# <sub>tID</sub> (e <sub>14</sub> )=t <sub>14</sub>	# <sub>labelN</sub> (e <sub>14</sub> )=d	# <sub>pi</sub> (e <sub>14</sub> )=p <sub>4</sub>	# <sub>time</sub> (e <sub>14</sub> )=7	# <sub>act</sub> (e <sub>14</sub> )=d	# <sub>resource</sub> (e <sub>14</sub> )=Ben

Figure 2.12: Prescribed behaviors specified by  $DPN'$ .

Normally, event logs do not contain events referring to invisible ( $\tau$ -labelled) steps, however, we will make extensive use of invisible steps in order to precisely capture compliance constraints in a DPN.

## 2.4 Aligning an Event Log to a Model

An observed behavior (event log) may deviate from a prescribed behavior (run). Deviations are detected using alignments [135]. An alignment pairs events of a log to events of a run. Each pair ( $e^L, e^R$ ) is called a *move* expressing that event  $e^L$  of the log aligns to event  $e^R$  of the run. In case the log deviates from the run,

not all events can be paired. For this, we introduce the symbol  $\gg$  denoting a “no move”, i.e.,  $\gg \notin \mathcal{E}$ . For any set  $E \subseteq \mathcal{E}$ :  $E^{\gg} = E \cup \{\gg\}$ .

Let  $E^L$  be a set of log events and  $E^R$  be a set of run events.  $(e^L, e^R) \in (E^L)^{\gg} \times (E^R)^{\gg}$  is a move. There are four types of moves:

- $(e^L, e^R)$  is a *synchronous move* if  $e^L \neq \gg$  and  $e^R \neq \gg$ ,
- $(e^L, \gg)$  with  $e^L \neq \gg$  is a *log-only move*,
- $(\gg, e^R)$  with  $e^R \neq \gg$  is a *model-only move*, and
- $(\gg, \gg)$  is an *invalid move*.

There are two kinds of *synchronous moves*. A *synchronous move* is *correct* (w.r.t. its attributes) if:

- $\#(e^L) = \#(e^R) \upharpoonright (\text{dom}(\#(e^R)) \setminus \{tID\})$ <sup>4</sup>.

Otherwise the *synchronous move* is *incorrect*.

An *alignment* of a log to a model pairs each event in the log to an event of some *run of the model* up to deviations. Thereby, two events paired in a move have to be consistent with the ordering of events both in the log and in the runs of the model respectively. In the original definition of alignments [135], two events of a move had to be consistent w.r.t. their *activity name* attribute ( $act \in Attr$ ). In the following definition, we introduce an attribute *checking*  $\in Attr$  as the attribute for which two events in a move have to be consistent. We allow the user to pick any attribute from the log as the *checking* attribute as long as all the events in the  $L$  have this attribute, i.e., for any  $e \in E^L$ :  $checking \in \text{dom}(\#(e))$ . Recall from Sect. 2.3.3 that each (data-aware) Petri net  $N$  maps its transition label to an attribute  $label_N$  of run events. We may pair log events with run events in a synchronous move whenever the checking attribute of the log event is consistent with the  $label_N$  attribute of the run event.

**Definition 2.7 (Alignment)**  $A = (L, R, M, \leq^M)$  is an alignment of log  $L = (E^L, \#^L, \leq^L)$  to run  $R = (E^R, \#^R, \leq^R)$  of some (data-aware) Petri net w.r.t. the consistency attributes  $checking, label_N \in Attr$  if and only if:

- $M \subseteq ((E^L)^{\gg} \times (E^R)^{\gg}) \setminus \{(\gg, \gg)\}$  is a set of moves, and
- $\leq^M$  is a partial order, such that:

<sup>4</sup>Given a function  $f$  with domain  $\text{dom}(f)$  and  $X \subseteq \text{dom}(f)$ :  $f \upharpoonright X$  is the function projected on  $X$ , i.e.,  $\text{dom}(f \upharpoonright X) = X$  and for all  $x \in X$ :  $f \upharpoonright X(x) = f(x)$

1.  $E^L = \{e^L \mid (e^L, e^R) \in M \wedge e^L \neq \gg\}$ , i.e., the left hand-sides of all moves are all log events,
2.  $E^R = \{e^R \mid (e^L, e^R) \in M \wedge e^R \neq \gg\}$ , i.e., the right hand-sides of all moves are all run events,
3.  $\forall_{e \in E^L} |\{(e^L, e^R) \in M \mid e^L = e\}| = 1$ , i.e., for each event in the log there is precisely one corresponding move,
4.  $\forall_{e \in E^R} |\{(e^L, e^R) \in M \mid e^R = e\}| = 1$ , i.e., for each event in the model run there is precisely one corresponding move,
5.  $\forall_{(e^L, e^R) \in M \cap (E^L \times E^R)} \#_{pi}(e^L) = \#_{pi}(e^R)$ , i.e., model events involved in a synchronous move belong to the same process instance,
6.  $\forall_{(e^L, e^R) \in M \cap (E^L \times E^R)} \#_{checking}(e^L) = \#_{label_N}(e^R)$ , i.e., events involved in a synchronous move hold in their checking attribute the value of the transition label,
7.  $\forall_{(e_1^L, e_1^R), (e_2^L, e_2^R) \in M \cap (E^L \times E^R)} e_1^R \leq e_2^R \Rightarrow e_1^L \leq e_2^L$ , i.e., the event log respects the ordering in the model run, and
8.  $\forall_{(e_1^L, e_1^R), (e_2^L, e_2^R) \in M} ((e_1^L, e_1^R) \leq^M (e_2^L, e_2^R)) \iff ((\{e_1^R, e_2^R\} \subseteq E^R) \wedge (e_1^R \leq^R e_2^R)) \vee (\{e_1^L, e_2^L\} \subseteq E^L) \wedge (e_1^L \leq^L e_2^L)$ , i.e., ordering of the moves in the alignment respects ordering of the log events in the event log or ordering of run events in the run.

Here we use the notion of alignments different from literature [135]. First of all, we do not align individual cases in the log, but align the log (a behavior) as a whole to a run (of multiple process instance runs). Second, we do not align activities but unique events. Third, we check consistency (i.e., enforce synchronous moves) w.r.t. user-chosen attributes (*checking*, and *label<sub>N</sub>*). Note that, *label<sub>N</sub>* can be simply a copy of the another attribute e.g. *activity name* (*act*) or it can be combination of several other attributes. Finally, we use partial orders rather than a total ordering of moves.

Given a log and a model, infinitely many alignments are possible. We are interested in alignments which minimize the number of “non-perfect matches” between log events and model events.

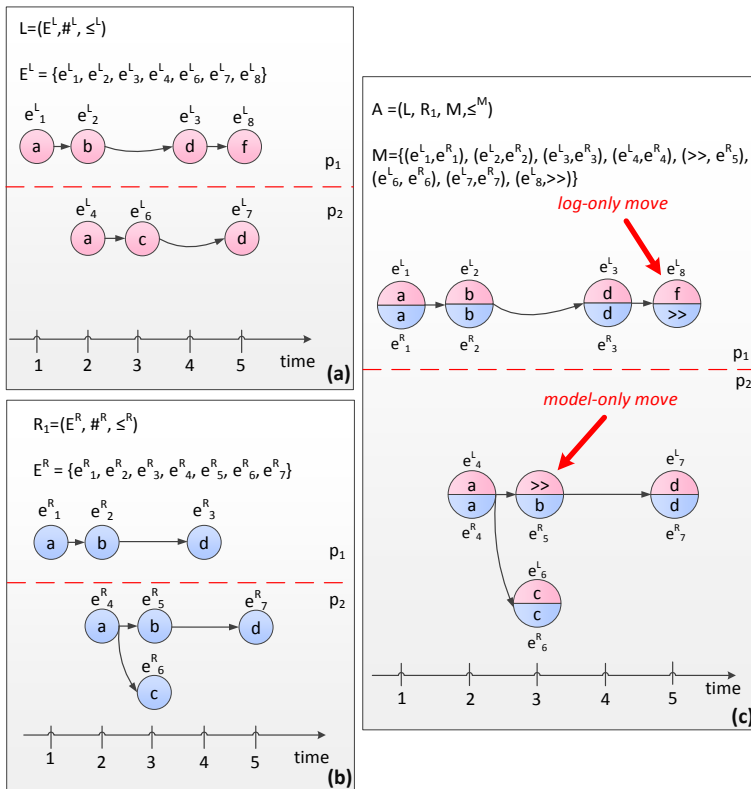


Figure 2.13: The partially ordered alignment  $A$  (c) between the event log  $L$  (a) and the process model run  $R_1$  (b).



**Deviations related to events.** Therefore, we typically define a notion of optimal alignment using a cost function to penalize moves where model and log deviate. Obvious deviations are log-only moves  $(e^L, \gg)$  and model-only moves  $(\gg, e^R)$ . However, incorrect synchronous moves are also deviations which we will discuss later in detail. Note that, different deviations may have different costs (e.g. model-only moves, log-only moves and incorrect moves may have different costs). Figure 2.13(c) shows an alignment between the event log  $L$  shown in Fig. 2.13(a) and the run  $R_1$  in Fig. 2.13(b). For this alignment we choose *checking* = *act*, i.e., we align log event to run event w.r.t. the *activity name*. The alignment  $A$  shown in Fig. 2.13(c) has a log-only move, a model-only move, and six synchronous moves that are partially ordered in every process instance. The log-only move in the alignment  $A$  indicates that event log  $L$  has an event without a corresponding event in  $R_1$  and the model-only move shows that an event was missing in event log  $L$  that was supposed to happen according to  $R_1$ . The arcs shown in the graph<sup>5</sup> of the alignment  $A$  illustrate the ordering of the moves.

**Deviations related to attributes.** A log and a model can also disagree with respect to attributes and attribute values. There can be synchronous moves disagreeing on attributes, i.e.,  $(e^L, e^R)$  with  $dom(\#(e^L)) \neq dom(\#(e^R)) \setminus \{tID\}$ , or disagreeing on an attribute value, i.e.,  $\#(e^L) \neq \#(e^R)$ , for example  $\#_{age}(e^L) \neq \#_{age}(e^R)$ . Synchronous moves never disagree on the consistency attributes (see Def. 2.7, point 6).

These kinds of deviations are illustrated in Fig. 2.14 which extends the events of Fig. 2.13 with their *resource* attribute. Besides the log-only move and the model-only move, alignment  $A'$  in Fig. 2.14(c) indicates an *incorrect synchronous move* (a synchronous move with a deviating resource attribute), i.e., event  $e_7^L$  was executed by *Nic* whereas the corresponding resource in the prescribed behavior ( $R_1$ ) is *Sara* ( $\#_{resource}(e_7^R) = Sara$ ).

When aligning behaviors with many different attributes, we will present an alignment in the form of a table. For example, Fig. 2.15 shows the alignment between the event log  $L$  and the run  $R_1''$  (Fig. 2.12) in a tabular form. Note that,  $R_1''$  is a run of the data-aware Petri net  $DPN'$  (Fig. 2.11). Log events are shown in the top row of the table and run events are shown in the bottom row of the table. Sub-rows refer to different event attributes and their values. The consistency attributes are highlighted. The alignment shown in Fig. 2.15 denotes five violations: three *model-only moves*, a *log-only move*, and an *incorrect syn-*

<sup>5</sup>Note, the graph of partially ordered alignment  $A$  includes only non-transitive arcs, i.e., we only show the minimal set of arcs whose transitive closure defines a partial order.

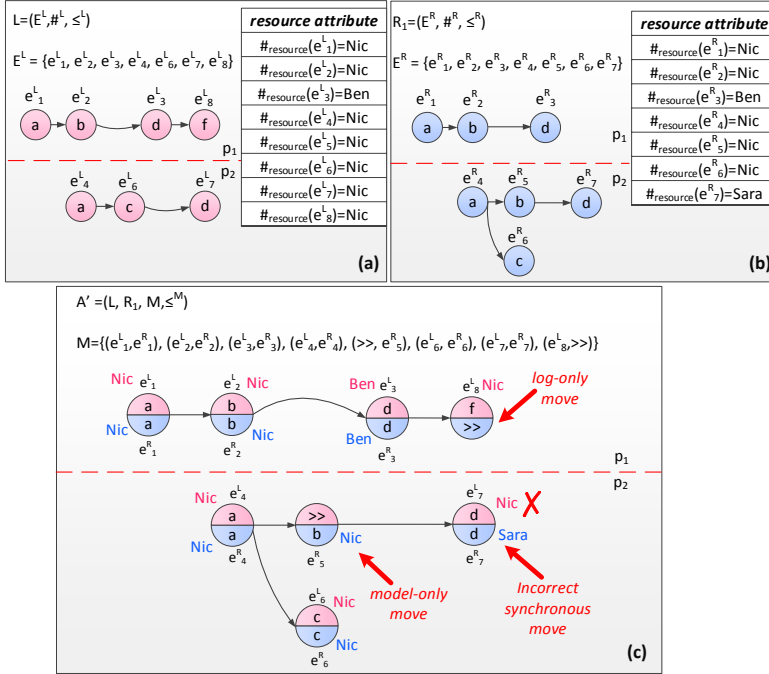


Figure 2.14: The partially ordered alignment  $A'$  (c) between the event log  $L$  (a) and the process model run  $R_1$  (b).

*chronous move*. Columns with a “>>” in the top row denote model-only moves, the column with a “>>” in the bottom row denotes a log-only move, the column with different attribute values between top and bottom row indicates an incorrect synchronous move (highlighted red). Note that two of these model-only moves are due to  $\tau$ -labelled (columns with gray background). Later, these will mostly be considered as a “technical” artifact and not as a relevant deviation between observed and specified behavior. We also typically abstract from *transition ID* ( $tID$ ) attribute since we pair run events, and log events on their consistency attributes. Note that if we abstract from moves related to invisible transitions and the attribute *transition Id*, in essence, we get the alignment  $A'$  shown between  $R'_1$ , and  $L$  (Fig. 2.12).

Finding optimal alignments requires solving an optimization problem mini-

L	resource	Nic	Nic	Ben	Nic	Nic			Nic		Nic	
	event timestamp	1	2	4	5	2			3		5	
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>2</sub>	>>	>>	p <sub>2</sub>	>>	p <sub>2</sub>	
	event ID	e <sup>L</sup> <sub>1</sub>	e <sup>L</sup> <sub>2</sub>	e <sup>L</sup> <sub>3</sub>	e <sup>L</sup> <sub>8</sub>	e <sup>L</sup> <sub>4</sub>			e <sup>L</sup> <sub>6</sub>		e <sup>L</sup> <sub>7</sub>	
	<b>activity name (checking)</b>	a	b	d	f	a			c		d	
R <sub>1</sub>	<b>activity name (label<sub>N</sub>)</b>	a	b	d			a	τ	b	c	τ	d
	event ID	e <sup>R</sup> <sub>1</sub>	e <sup>R</sup> <sub>2</sub>	e <sup>R</sup> <sub>3</sub>			e <sup>R</sup> <sub>4</sub>	e <sup>R</sup> <sub>15</sub>	e <sup>R</sup> <sub>5</sub>	e <sup>R</sup> <sub>6</sub>	e <sup>R</sup> <sub>16</sub>	e <sup>R</sup> <sub>7</sub>
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	>>		p <sub>2</sub>	p <sub>2</sub>	p <sub>2</sub>	p <sub>2</sub>	p <sub>2</sub>	p <sub>2</sub>
	event timestamp	1	2	4			2	-	3	3	-	5
	resource	Nic	Nic	Ben			Nic	-	Nic	Nic	-	Sara
	transition ID	a <sub>1</sub>	b <sub>1</sub>	d <sub>1</sub>			a <sub>2</sub>	τ <sub>1</sub>	b <sub>2</sub>	c <sub>2</sub>	τ <sub>2</sub>	d <sub>2</sub>

Figure 2.15: Alignment between the event log  $L$  and run  $R'_1$  shown in tabular form.

mizing the differences between a model run and a log using a cost function. As mentioned earlier, we can assign different costs to different violations (e.g. violations of types ‘model-only move’, ‘log-only move’ and ‘incorrect synchronous move’ may have various costs). Here, we do not consider a specific algorithm to solve the optimization problem and assume an ‘oracle’. A concrete oracle for finding optimal alignments has been defined in [135]. Given a log and a model, the oracle produces an optimal alignment. There could be multiple optimal alignments having the same deviation costs. However, in the following, we limit ourselves to the setting where the oracle provides one of these optimal alignments in a deterministic manner.

**Definition 2.8 (Oracle)**  $\alpha$  is the oracle function, i.e., it is a deterministic function such that for any event log  $L \in BH$  and model  $Mod \subseteq BH$ :  $\alpha(L, Mod) = (L, R, M, \leq^M)$  is an alignment such that  $R \in Mod$ .

Let  $\beta$  be a cost function that assigns each move  $m$  a cost  $\beta(m) \geq 0$ . The cost of an alignment  $(L, R, M, \leq^M)$  is then  $\sum_{m \in M} \beta(m)$ . An oracle  $\alpha$  is a deterministic function such that for any event log  $L \in BH$  and any model  $Mod \subseteq BH$  holds:  $\alpha(L, Mod) = (L, R, M, \leq^M)$  is an alignment such that  $R \in Mod$  and for any other alignment  $(L, R', M', \leq^{M'})$  of  $L$  and  $Mod$  holds:  $\beta(M') \geq \beta(M)$ .

It is easy to see that there always exists an alignment between a log  $L$  and a model  $Mod$ : we can pick a run  $R \in Mod$  and construct) the alignment  $A = (L, R, M, \leq^M)$  that has just log-only moves for the events in  $L$  and just model-only moves for the events in  $R$  (of course the ordering need to be respected),

i.e.,  $M = \{(e^L, \gg) \mid e^L \in E^L\} \cup \{(\gg, e^R) \mid e^R \in E^R\}$ . Such an alignment  $A$ , given also the right  $\leq^M$ , satisfies all the requirements of Definition 2.7. Oracle  $\alpha$  always returns an alignment with the least cost. It can be realized in a way similar to [44, 135].

## 2.5 Specifying Prescribed Behaviors from Different Perspectives

The transition labels in DPN or DPN' (Fig 2.9 and Fig. 2.11) capture the admissible values of attribute *act* (activity name) for each event. Next, we discuss how we can capture the admissible behaviors from a different perspective, for example the *resource* attribute.  $DPN^{resource}$  shown in Fig. 2.16-(top) explicitly models the *resource* values as the transition labels.

This net has an initial place, a final place, and five transitions including two invisible transitions. The net has three variables capturing values of *process instance*, *activity name*, and *time* attributes. For the ease of representation, the ellipses related to these variables are not shown in the net. The first invisible  $\tau$ -labelled transition sets the initial values for attributes *process instance*, and *time*. Other transitions will not update the value of *process instance* attribute. The attributes *time*, and *activity name* get updated by other transitions except the second invisible  $\tau$ -labelled transition that will only change the resource value.

The runs generated from this net are partially shown in Fig. 2.16-(bottom). If we abstract from *transition ID* attribute and events related to invisible transitions (these events are shaded in gray), the behaviors described in these runs are the same behaviors described in  $R_1$ , and  $R_2$ . Therefore, in essence DPN, DPN', and  $DPN^{resource}$  all specify the same behaviors. However, they differ significantly in the perspective they give on the specified behavior (i.e., order of activities versus order of resources in a process). In principle, we can choose any attribute of a run as the consistency attribute  $label_N$  in run and model it as transition labels of a (data-aware) Petri net as long as the attribute is global in the run, i.e., all the events in the run share the chosen attribute. Note that, for all the patterns always the mapping  $\#_{label_N}(e) \rightarrow transition\ label$  always hold. Therefore, in the remaining part of the thesis, we will not show this information in the patterns.

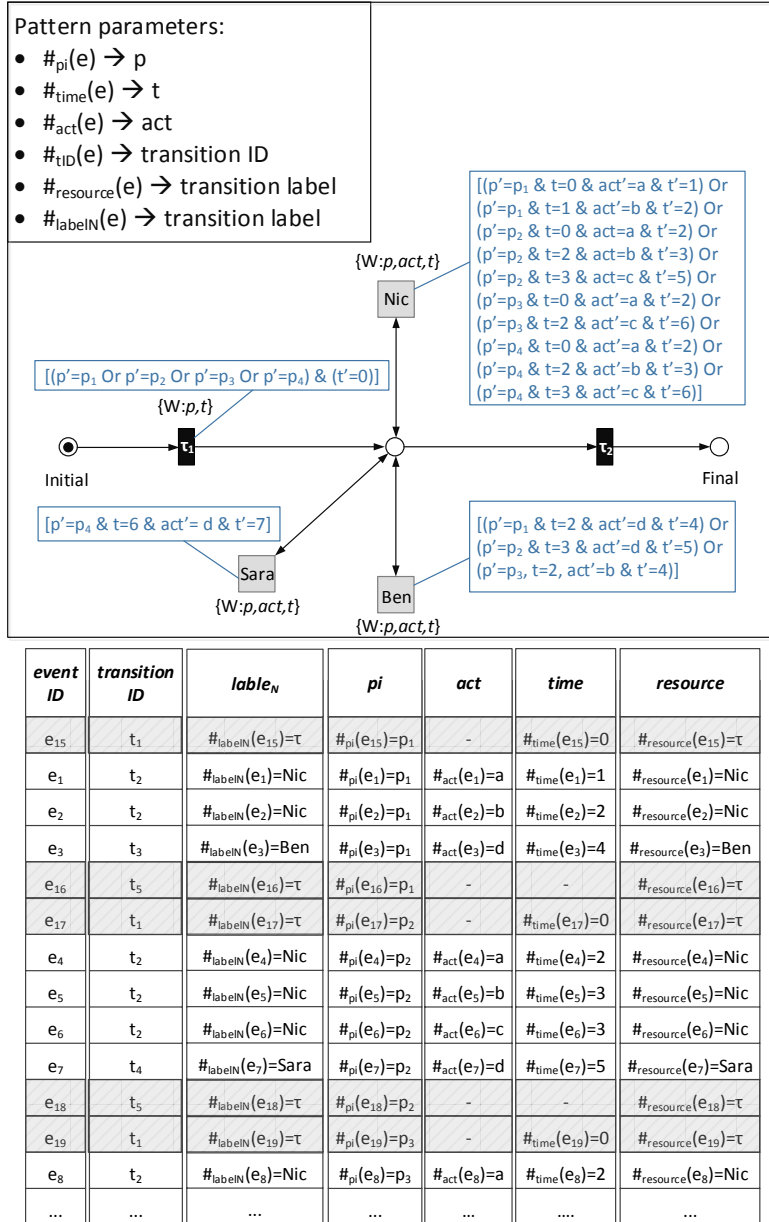


Figure 2.16:  $DPN^{resource}$ : The data-aware Petri net modeling values of attribute *resource* as transition labels.

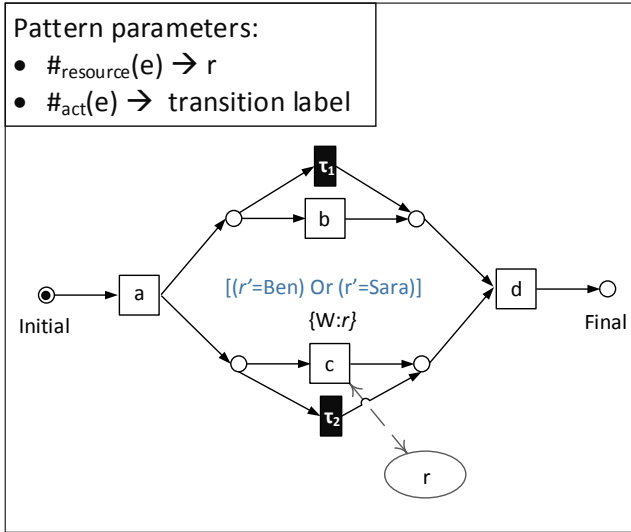


Figure 2.17:  $DPN^{generic}$ : The data-aware Petri net can replay  $R_1$ ,  $R_2$  and many more runs as long as they adhere to the constraints.

## 2.6 Specifying behaviors with (Data-Aware) Petri nets Using an Open World Assumption

Although  $DPN$ ,  $DPN'$ , and  $DPN^{resource}$  precisely describe the behaviors  $R_1$  and  $R_2$ , the admissible firing sequences in these nets are limited to *process instance runs* in  $R_1$  and  $R_2$  (abstracted from the *transition ID* and events related to invisible transitions).

In this sense, the nets only allow for the behaviors exactly specified and exclude any other behavior, which corresponds to *closed world assumption*. However, compliance constraints have an *open world assumption*: everything is allowed unless it violates the given constraint. Next, we discuss how we can use (data-aware) Petri nets to specify behaviors in an open-world assumption.

Assume we would allow for any occurrence of activities  $a, b, c$ , and  $d$  as long as (1) the sequence  $\langle a, b \rangle$  is present and (2) activity  $d$  may only be executed by *Ben* or *Sara*. In this case, we can abstract from the strict guards annotating the transitions in  $DPN$ ,  $DPN'$  or  $DPN^{resource}$  and allow for any occurrence of

these activities as long as they adhere to the constraints.  $\text{DPN}^{\text{generic}}$  (shown in Fig. 2.17) describes the two runs  $R_1$  and  $R_2$  and many more, while adhering to the constraints we mentioned.

In this pattern, we capture the compliant sequence of activities by ordering the transitions. As can be seen, similar to  $\text{DPN}$  and  $\text{DPN}'$ , transition labels in  $\text{DPN}^{\text{generic}}$  make activity names explicit, allowing us to abstract from attribute  $act$  in the guards. In addition, we can also abstract from the attribute  $time$  as long as activities follow the specified sequence. Note that the above specification does not constrain time delays between two activities. We also do not need the attribute  $pi$  since the start and completion of a firing sequence, specified by  $\text{DPN}^{\text{generic}}$ , is interpreted as exactly one process instance run. The precise value of the process instance is irrelevant to us and it can take any value. The alignment will then pick a suitable process instance for a given execution in a log. The only attribute we need to explicitly model in  $\text{DPN}^{\text{generic}}$  is the  $resource$ . Further, because this attribute is only constrained on the occurrences of activity  $d$ , we can locally connect it to  $d$  and restrict it with the guard  $[(r = Ben) \text{ Or } (r = Sara)]$ .

As discussed before, in a data-aware Petri net some attributes may be connected to all the transitions and some may be used only locally for specific transitions. Accordingly, the events produced as the result of executing these transitions may or may not have an attribute. We can constrain the admissible values of attributes by annotating each transition with guards. For all attributes not specified in  $\text{DPN}^{\text{generic}}$ , their value can change arbitrarily except:

- $pi$  (process instance attribute): for  $pi$ , we assume events related to each firing sequence to have the same  $pi$  value (as required by the Def. 2.4.)
- $time$ : We require that time values agree with the ordering of events. That is,  $e_1 \leq e_2 \Rightarrow \#_{time}(e_1) \leq \#_{time}(e_2) \wedge \#_{pi}(e_1) = \#_{pi}(e_2)$ .

Note that if we only have constraints on the presence of activities and their sequence, we can abstract from all other attributes and model the behavior as a classical Petri net. In chapters 4, 5, and 6, we will explain in detail how to model prescribed behaviors as (data-aware) Petri nets.

## 2.7 Concluding Remarks

This chapter introduced the general definition of behavior as a collection of partially ordered events. Intuitively each event refers to a set of attributes and

value pairs that records an execution of a particular activity in a particular context. Each event is represented by a unique identifier. Hence, we can distinguish events even if they have identical properties.

Both event log and model are behaviors. We express prescribed behaviors as a (data-aware) Petri net. We can constrain the presence of events, their ordering of occurrence and their properties by using transitions, their sequence, their attributes, and transition guards.

We use alignments to relate events in a prescribed behavior to events in an observed behavior and detect deviations. An alignment is a collection of partially ordered moves where each move refers to a particular event in the observed behavior and a particular event in the prescribed behavior. A move can be compliant or violating. Violating moves are of types: log-only move, model-only move, or incorrect synchronous move. Since a move pairs two unique events, moves with identical properties can be distinguished.

We introduced the notion of *consistency* attributes to the alignment which allows us to pair observed and specified events based on a user-chosen attribute from the log and the model. We will make extensive use of these consistency attributes when checking for compliance w.r.t. various perspectives of a process.





# Chapter 3

## Requirements for Analyzing Compliance

To improve compliance in a sustainable manner, it is necessary to understand non-compliance. To understand non-compliance, we need diagnostics that provide insights needed. These diagnostics impose some requirements on different phases of a compliance analysis approach.

In this chapter, we discuss different requirements on various phases of a compliance analysis approach including compliance rule elicitation and formalization requirements, requirements on compliance checking techniques and requirements on diagnostics.

Figure 3.1 illustrates how this chapter is organized. In Sect. 3.1, we discuss different dimensions of compliance constraints that a process must adhere to. In Sect. 3.2, we propose a comprehensive compliance analysis approach and its required steps. In Sect. 3.3, we sketch the type of diagnostics that the compliance analysis approach should deliver us. We discuss the requirements of a compliance checking technique both regarding the desired diagnostic information and the kind of inputs required for such diagnostics. Based on these requirements, we then discuss a number of technical and conceptual choices for realizing compliance analysis.

In Sect. 3.4, we discuss various aspects of formalizing compliance constraints. In the same section, we also present three collections of compliance constraints. Section 3.5 discusses the selected compliance checking technique. We will sketch our ideas for providing diagnostics in Sect. 3.6.

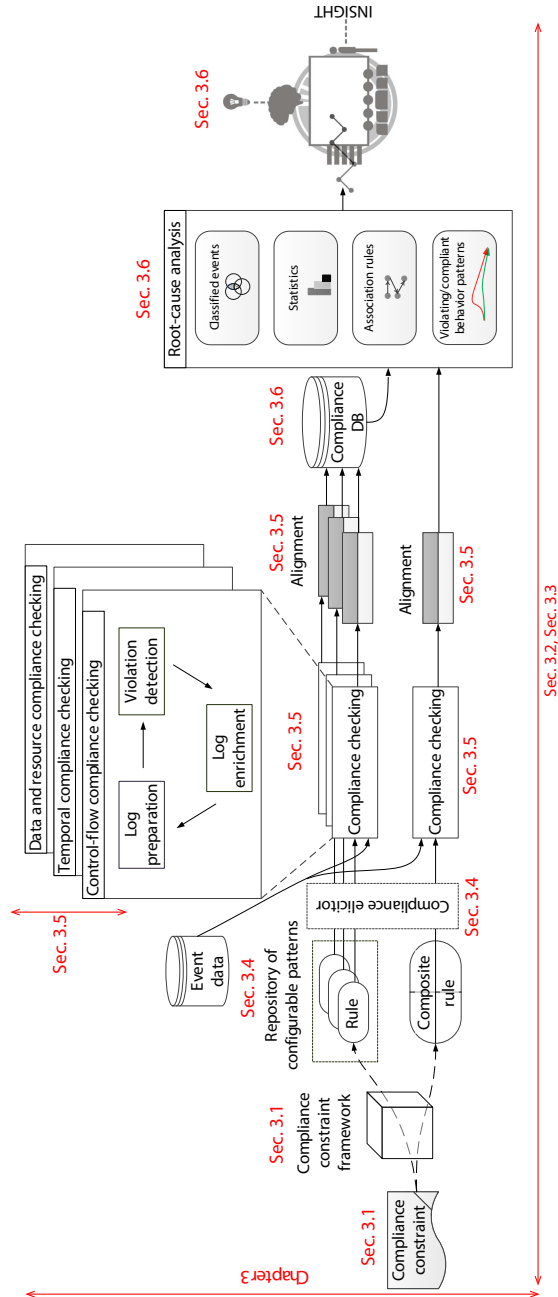


Figure 3.1: Thesis road map gives the mapping of the sections in Chap. 3 on to our compliance analysis approach.

### 3.1 Dimensions of Compliance Constraints

Compliance constraints prescribe how an internal or cross-organizational business process has to be designed or executed. Usually a business process is governed by several constraints. Compliance constraints may limit individual perspectives of a business process (control flow, data flow, organizational aspects, and process time) or a combination of several perspectives. For example, consider the compliance constraint: “After a claim of more than 3000 euros has been filed, two different employees need to check the validity of the claim independently. Each claim must be handled at most 6 months after the placement.” This constraint refers to (1) *control flow* (“After a claim has been filed, validity must be checked”), (2) *data flow* (“A claim of over 3000 euros requires two validity checks”), (3) *the organization* (“Multiple validity checks are carried out by different employees”), and (4) *the temporal dimension* (“Within 6 months the claim must be processed”).

Furthermore, a compliance constraint can (5) prescribe properties of a single

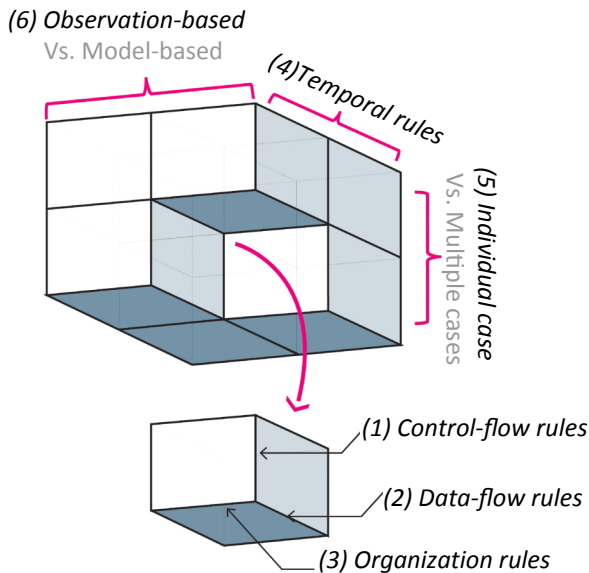


Figure 3.2: Compliance constraints framework.

case or of multiple cases e.g. “20% of all claims require a detailed check.”, and (6) prescribe properties of the process design e.g. “The claim process must have a time-out event handler.” or properties of the process executions, which can be observed (i.e., recorded in an event log).

A constraint usually covers several perspectives of a process. Our studies [97, 98, 130, 131] analyzing the existing body of literature in compliance and our experience on different case studies show that even small compliance constraints are concerned with multiple perspectives.

Based on these observations, we identified six orthogonal dimensions of compliance constraints. These different aspects of compliance give rise to the framework shown in Fig. 3.2.

To enable precise compliance checking of a constraint, we decompose it into several related *compliance rules*. Each rule is concerned with one process dimension only. As discussed in Chap. 1, in this thesis we discuss techniques for *compliance auditing* of a business process, i.e., checking after the execution of a business process whether it complies to the relevant compliance constraints or not (thereby we focus on rules that are observation-based). In addition, we detect possible violations in the entire log for each case individually and we do not focus on compliance rules constraining multiple cases. We define the scope of this thesis to cover fully the remaining four dimensions of the framework, i.e., we analyze compliance of processes against compliance constraints that confine control-flow of a process, process time, process data, and organization (resource).

## 3.2 Compliance Analysis Overview

In order to comply with a set of compliance constraints confining a business operation, the *Compliance Management (CM) life cycle* 1.3 discussed in Chap. 1, introduces five types of compliance related activities: (1) elicit compliance constraints that have to be satisfied, (2) formalize precisely the elicited compliance constraints, (3) implement and configure information systems such that they fulfill the compliance constraints, (4) check whether the compliance constraints will be met (*forward compliance checking*) or have been met (*backward compliance checking*), and (5) improve the processes and systems based on diagnostic information.

Figure 3.3 illustrates an overview of our approach for compliance analysis which reflects different phases of the CM life cycle. Our approach starts with elicitation and formalization of compliance constraints. As discussed pre-

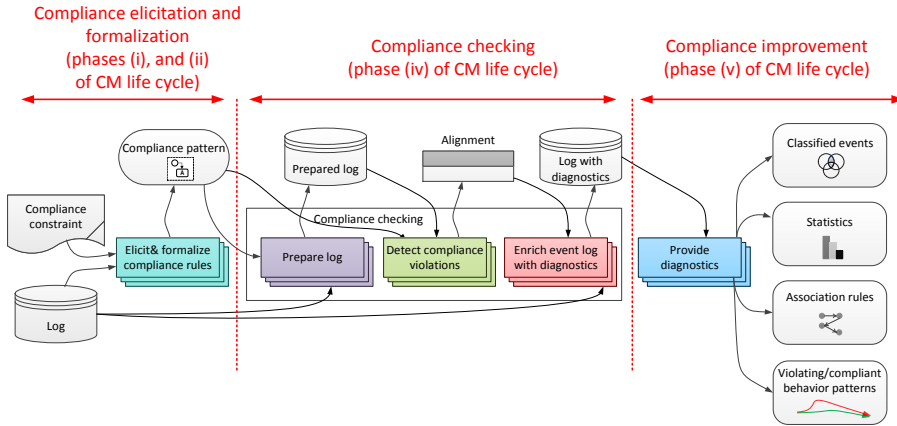


Figure 3.3: Compliance analysis overview.

viously in Chap. 1, we advocate the idea of separating *compliance management* from *business process management*. Therefore, we deliberately avoided adding a component in our approach for implementing and configuring compliance constraints within business processes. The next phase of the CM life cycle supported by the proposed approach is compliance checking and the approach ends by providing diagnostics to improve a business operation.

In the following, we discuss how and under which conditions this proposed approach can support compliance analysis in practice.

### 3.3 Requirements for Compliance Analysis

First, we discuss how our approach, depicted in Fig. 3.3, can realize compliance analysis by discussing requirements from practice, starting from the desired diagnostics.

#### 3.3.1 Requirements on Compliance Diagnostics

The aim of analyzing compliance of a process is to get an overview of how compliant the process is. Yet a more important goal is to obtain insights about *non-compliance* and possibly some insights about the root causes of violations that would guide us to take corrective measures and improve compliance. Hence,

there are several groups of questions that should be answered by the diagnostic information we seek to produce:

- How many times each compliance constraint is violated? What are the violating activities?
- What kind of violations occurred? How frequent they are?
- What exactly went wrong and what should have happened instead?
- Which violations are more important and require a closer look first?
- What are the causes of a specific violation?

These questions form the prerequisites for different steps of this approach. Next, we will discuss these questions and illustrate the kind of answers a compliance analyst might expect from our approach.

### **First group of questions: Violating constraints, violating activities and their frequencies**

We would like to know: *how many times each compliance constraint is violated?* and *what the violating activities are?*

A business process usually is confined by several compliance constraints (e.g. in Chap. 9, we analyze compliance of one single business process against eight distinct compliance rules). There are many sources of compliance rules: (1) *regulatory compliance* (e.g. national law), (2) *commercial compliance* (e.g. service level agreements), (3) *organizational compliance* (e.g. internal policies of organizations). Violation of a rule incurs cost or worse. In order to improve business operations in the right way (i.e., such that violations no longer occur), it is important to know for each rule whether it is violated and in which business activity.

To answer these questions, we first need to have a clear definition of a **violation**. A violation can be defined as a deviation observed in reality from the prescribed behavior specified by a compliance constraint. For example, consider a simple compliance constraint  $C_1$  stating: “**Every execution of activity A must be followed immediately by the sequence of activities  $\langle B,C \rangle$** ”. Assume we have the process instance  $p$  as shown in Fig. 3.4. We can see that this process instance is violating the compliance constraint because at least three occurrences of  $A$  were *not* followed by the sequence  $\langle B,C \rangle$ . In this process instance, activity  $A$  occurred five times, that means the compliance constraint was

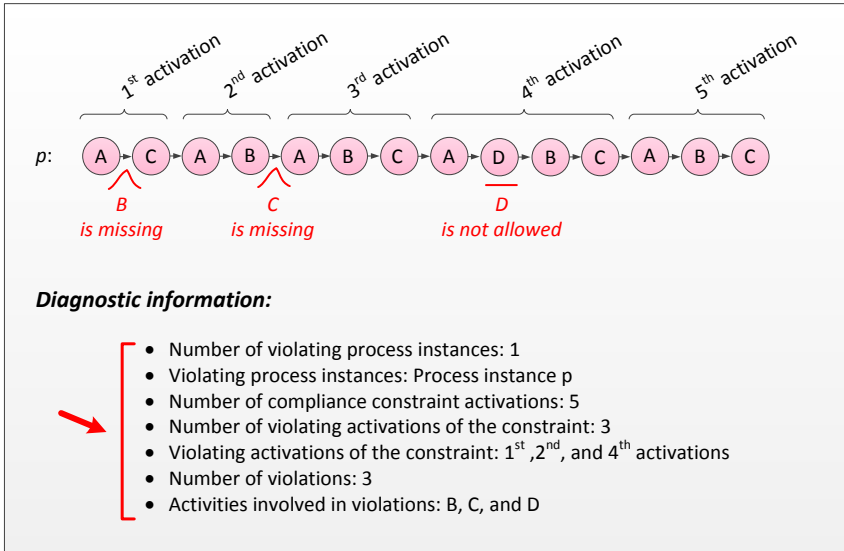


Figure 3.4: Diagnostics about violations of the constraint  $C_1$  and the violating activities.

activated five times (i.e., each occurrence of A demands that the rule is satisfied (again)), but it was completed in a compliant manner only twice and three times not, resulting in a violation. We are not only interested to know if a process instance is compliant or violating but also we would like to know exactly how many times a constraint is violated.

In addition, we would like to know what the violating activities are, because we would like to spot where a process is more vulnerable and prone to violation. The first activation of the constraint is non-compliant (B is missing). In the second activation of the constraint, occurrence of C is missing. So, the violating activity is C. The third activation is compliant. In the fourth activation of the constraint, D occurred that was not allowed. Hence, D is the violating activity and the fifth activation is compliant. Figure 3.4 (bottom) illustrates possible diagnostics that could be reported for this example.

In general, suppose we have several constraints that a process must adhere to, like the one of our example. We would like to know:

- What are the violating process instances per constraint?



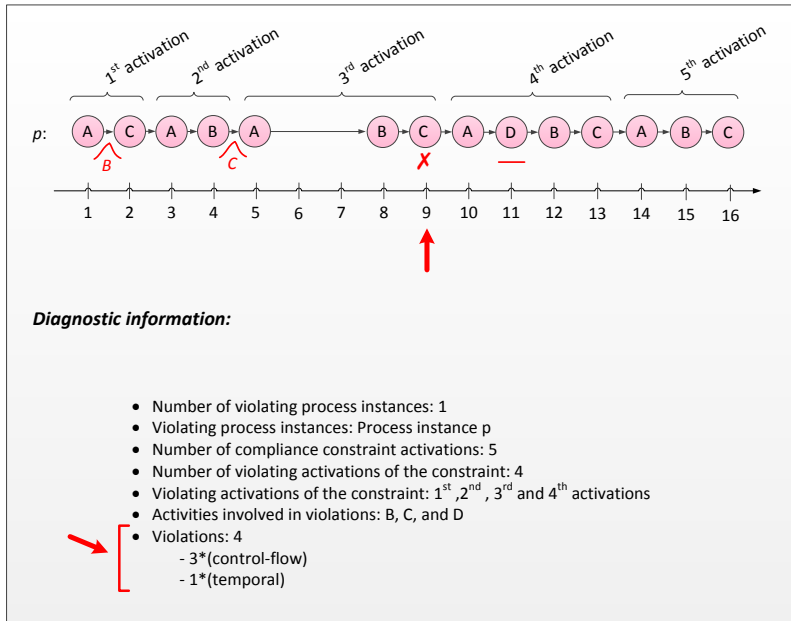


Figure 3.5: Diagnostics about violations of the constraint  $C_1$  extended with information about the violation types and their frequency.

- *How many times each of the constraints are activated in total and how many times they are violated? What activities are violating and how many violations are observed per activity? In total? Per process instance? and per constraint?*

The technique developed in this thesis should allow for producing such (or even more comprehensive) diagnostic information. These diagnostics will help us to get an overview about the compliance level of a business process.

### Second group of questions: Violation types and their frequencies.

The second group of questions is concerned with the type of violations and the frequency of each type.

Recall from Sect. 3.1 that a compliance constraint may restrict different perspectives of a business process. Hence, when we gather diagnostics about vio-

lations, it is useful to know what perspective of the process has been violated (*type of violation*) and how violations are distributed over the different types.

Consider the following compliance constraint which extends the previous example with a restriction on the temporal perspective: “**Activity A must be followed within three time units by the sequence of activities  $\langle B,C \rangle$** ”. Figure 3.5 shows events of a process instance  $p$  along a time axis.

As illustrated in Fig. 3.5, first and second activations of the compliance constraint have violations of type *skipped activity*. The third activation of the constraint is compliant with the control-flow restriction (i.e., occurrence of the sequence  $\langle B,C \rangle$  after  $A$ ) but violated the temporal restriction (i.e., activity  $C$  occurred later than expected). The fourth activation of the constraint violates the control-flow (occurrence of  $D$  is not allowed). The fifth activation of the compliance constraint is compliant. Both activities  $B$ , and  $C$  occurred within the time specified by the constraint. Figure 3.5 (bottom) illustrates what kind of diagnostics could be given.

Such diagnostics help understand the violations. That is, it helps understanding:

- *What type of violations occurred in total (i.e., how vulnerable are the processes w.r.t. to time, control-flow or data and resource perspectives)? In a process instance? and per activity?*
- *How often did they occur?*
- *Which type of violations occurred more than others?*

In order to produce diagnostics of this kind, one needs a compliance analysis approach that can separate compliance constraints w.r.t. their different perspectives, analyze compliance in each perspective, and integrate the results.

### **Third group of questions: Violation compensation.**

These questions are related to identifying exactly what should change in a process to make it compliant. We would like to know exactly:

- *What went wrong? (e.g. which activity, which data, which resource with which amount, ... of deviation)?*
- *What should have happened instead?, i.e., How the violation can be compensated*

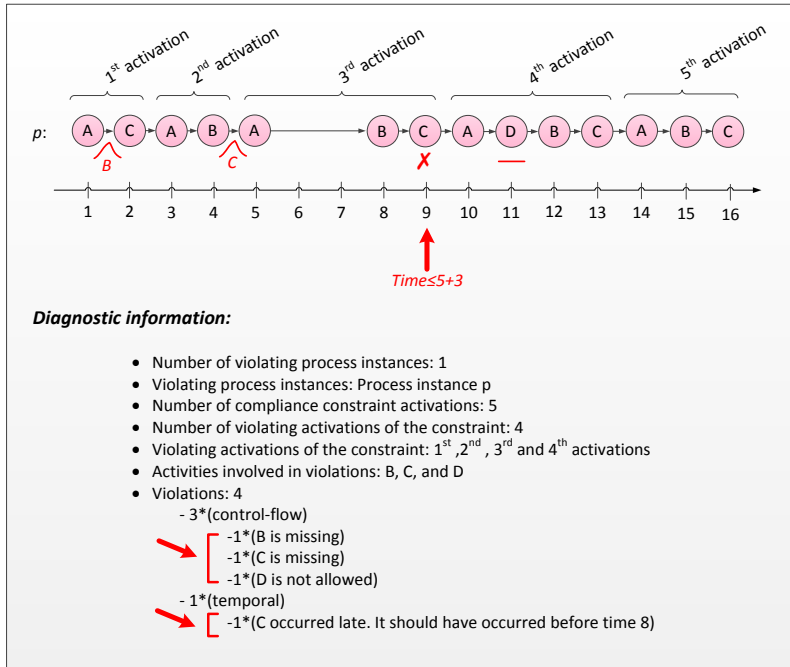


Figure 3.6: Information on how the process should have been executed differently to be compliant.

To understand a specific violation, it is not enough to know the type of violation and its frequency. It is also important to know exactly where in a process instance a violation occurred and how the process should have been executed to be compliant.

For instance in case of our previous example, Fig. 3.6 shows in the first, and second activation of the constraint which activities should have happened to make process instance *p* compliant. It is also shown which activity was executed while it was not allowed in the fourth activation of the constraint. Similarly it is useful information to know that activity *C* in the third activation should have happened before time 8. Such diagnostics are required to guide business users to take corrective actions and change the process in order to prevent future violations.

Thus, the compliance checking technique should not only report the violations, but also be able to show in detail the compliant behavior that should have occurred instead.

#### **Fourth group of questions: Important violations.**

Compliance constraints in an organization may have different priorities. They can be prioritized for instance based on the negative consequences that their violation may impose, or based on the source that a compliance constraint originates from. Violating compliance constraints that originate from text of law may have severe legal consequences for an organization compared to constraints that originate from internal policies of a company and are set to improve business operations. Another example would be compliance constraints that are industry specific and usually are demanded by clients and business partners; violating such constraints may lead to losing a client or high costs.

As a result of compliance checking, many violations may be detected. However, not all of these violations have the same priority for a company or are of the same severity. The severity of a violation is not limited to its frequency but is also influenced by the negative consequences that it may have for an organization. We would like to know: *What are the important violations that require a closer look?*

For instance, if violations of a specific rule are related to a higher monetary value or have far reaching legal consequences for a company, they would be more important compared to others. Hence, it is required to rank violations such that the important ones are highlighted and will not be hidden among minor violations. We can compute severity of violations based on their frequency and importance.

Suppose in our example, violations related to activity *B* have priority 1, temporal violations have priority 2, and other types of violations have a lower priority. Since each type of violation observed occurred once, then the ranked list of violations will be:

- Activity *B* is missing.
- Activity *C* occurred late.
- Activity *C* is missing.
- Activity *D* is not allowed.

In this thesis, we will investigate how to determine the relevance of violations based on event data.

**Fifth group of questions: Root causes of violations.**

Reporting statistics about violations, their type, frequency and importance provides a view on the compliance level of a process but it is also important to have diagnostics about the causes of violations. We would like to know if we can use the contextual data available for the process to understand:

- *Who are the main actors involved in violations of a particular type?*
- *Is there an indicator in the process that allows us to predict future violations?*
- *Can we find a pattern by comparing violating and non-violating instances of activities and processes?*

A compliance analysis technique needs to relate violations to the context in which they occur in order to be able to detect the factors that influence compliance or non-compliance.

**Presenting diagnostics.**

In addition to the requirements for diagnostics we listed above (described by the five groups of questions to be answered), it is required that this information should be presented at different abstraction levels to not only give an overview about the compliance but also to provide detailed analysis of a specific violation.

These diagnostics should address the needs of the auditors and process experts who are in charge of assessing and understanding compliance. Therefore, diagnostic information should be presented in concise and intuitive visualizations. Such a visualization must be complete and expressive in non-technical terms to intuitively and interactively guide business users through the wealth of information. Its structure must meaningfully group and summarize the available diagnostics and its navigation structure should effectively convey the necessary information.

### **3.3.2 Requirements for Compliance Checking Techniques**

In the following, we will discuss key requirements that compliance checking techniques must have to provide the kind of diagnostics described in Sect. 3.3.1.

Existing techniques from process and data mining on event data can help finding root causes and contexts of violations. Thus, projecting checking results back to event logs, provides us the possibility to analyze the violations further by preparing different statistical reports and leveraging various data mining techniques.

To obtain diagnostics about violating events and activities (first and second group of the questions listed in the previous section), we need to have dedicated techniques to check compliance of a process from different perspectives including control-flow, time, data, and resource. After checking compliance from different perspectives and for all relevant compliance constraints, we need to combine all obtained checking results to get an overall view on compliance. These checking techniques should not only identify violating and non-violating process instances. They should also detect all the violations in the entire process, specify exactly the type of each violation, specify to which compliance constraint it is referring, specify to which instance of a constraint is violated, and specify what the violating activity is? Therefore, the checking technique employed should be able to precisely locate each violation in a process instance.

The checking technique should state for each violation what should change in a process to make it compliant (third group of questions). The proposed change should not consider the impact of this change only locally per constraint activation but globally in a process instance. This is of particular importance when a violation related to one activity may influence the compliance of other activities.

To elaborate on this, suppose we have a compliance constraint stating: (1) **“Activity A may only occur with *amount* values less than 20.”** and another constraint requires: (2) **“If *amount* value is 10 or more then activity C must be executed.”**

Suppose we have observed the sequence of activities in a process instance  $p$  as shown in Fig. 3.7<sup>1</sup>. Activity  $A$  is violating because its *amount* should have had a value less than 20 according to (1). If the checking technique recommends a value between 10 and 20 for *amount*, then the process instance  $p$  would be compliant w.r.t. the first compliance constraint. Suppose the modified value is 15. The new value for *amount* satisfies the second constraint because activity  $C$  is not executed in the process instance  $p$ . Hence, the only acceptable *amount* value that will not violate any of the constraints is between 0 and 10. A compliance checking technique therefore requires to consider the impact of recommended compliant values globally in a process instance.

---

<sup>1</sup>Note that we use an informal representation of the notions we defined in Chap. 2.

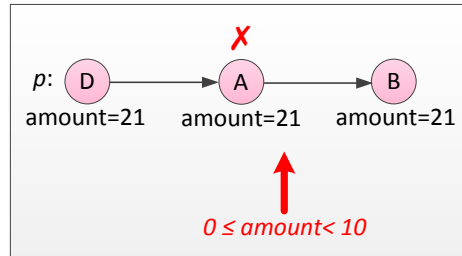


Figure 3.7: The compliance checking technique should consider the impact of proposed compliant values globally in a process instance.

In summary, the checking technique should:

- detect violations of different types including control-flow related, temporal, data and resource related violations,
- detect all violations not only violating instances of a process,
- locate violations per process instance, constraint, constraint instance, and activity, and
- suggest compensation for detected violations globally in a process instance.

The (*data-aware*) *alignment* technique discussed in Chap. 2 satisfies all the requirements for a checking technique we listed above. Using data-aware alignment we can detect violating constraints and violations of different types, locate the process instance and constraint activation where a violation occurred, and the suggests the compensation values that would make an execution of the process compliant. All other diagnostics we listed above (questions in groups four and five ) can be built using other data analysis techniques such as association rule mining and data classification based on the detailed results that alignment produces.

By selecting alignments as a basic mechanism, we also make constrain the kind of inputs our compliance analysis technique can accept: the ‘observed behavior’ has to be an event log (for example in XES format). We chose to formalize compliance constraints as (data-aware) Petri nets to express the ‘prescribed behavior’. There exists a powerful set of tooling for Petri nets in conjunction

with alignments. Next, we discuss requirements for inputs of our compliance analysis approach.

### 3.3.3 Requirements for Event Logs as an Input for the Compliance Checking Technique

To leverage the selected (data-aware) alignment checking technique, we need to map the attributes of events being checked in an event log to transitions in a (data-aware) Petri net. Recall from Sect. 2.4 that the use of alignments implies that the user has to select a particular checking attribute to relate observed events to transitions of the formal specification. Sometimes the checking attribute can be simply the activity name (*act*), for instance when we are checking the existence or sequence of occurrence of activities. However, in many cases, we may have to consider several attributes together to decide whether a particular event is subject to a compliance constraint. For example, if a rule states that “**payments over 5000 have to be approved by a manager**”, then the rule only concerns those events with activity name “payment” where also attribute “amount” is larger than 5000. The choice of the checking attribute depends on the compliance constraint to be checked and may vary from one to another. The compliance checking techniques must support checking attributes in a generic way to be able to cover different compliance constraints. Therefore, prior to the detection of violations, we need to prepare the event log for checking by choosing the right attribute or combination of attributes.

In summary:

- we need to be able to elicit an appropriate checking attribute from the information recorded in an event log,
- the attribute in the event log, chosen as the checking attribute to map events to transitions of a Petri net, must be global. That is, all the events in the event log must have a value recorded for the checking attribute. If an attribute only has a value in some events of the log, it cannot be chosen as the checking attribute.

### 3.3.4 Requirements for Elicitation and Formalization of Compliance Constraints

In order to enable automated compliance checking, the user has to choose which compliance constraints shall be checked.



During the elicitation phase of the CM life cycle, the user identifies the compliance constraints relevant for the organization by analyzing the profile of the organization including information such as company size, industry, region, and products or services.

After choosing suitable compliance constraints that an organization must comply to, the user has to identify the restrictions imposed by the constraint for each of the perspectives of the business process. As we discussed earlier in Sect. 3.1, complex compliance constraints may confine control-flow, data, resource, and time of a process. Detecting violations related to different constraints asks for a dedicated checking technique for each violation. Consequently, we need to capture constraints on each perspective separately. To specify compliance constraints precisely, we decompose them to smaller *compliance rules*. The formalization technique needs to be able to specify each compliance rule precisely and only restrict the behavior of a process where necessary, i.e., it must allow for all possible compliant behaviors.

Precise formalization of informal compliance rules, regardless of which formalization is used, is a difficult and time-consuming task that requires business knowledge together with knowledge of an appropriate formalization language. That particular combination is sometimes hard to find. Therefore, we require tool support that enables business users to formalize the intended compliance rule without being exposed to all the technicalities and inner workings of a formalization language.

In summary the formalization should:

- enable us to precisely determine boundaries of a compliant behavior and exclude violating behavior,
- have formal semantics to enable automated compliance checking,
- allow for specification of all types of compliance rules including control-flow, data-aware, resource-aware, and temporal rules,
- allow for all possible compliant behavior,
- allow the user to focus only on confined behavior.

In summary the elicitation and specification approach should:

- be easy for non-technical users to employ for specifying an intended behavior of a compliance rule.

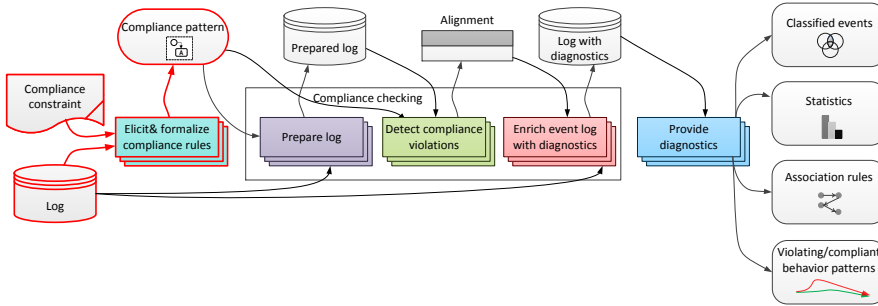


Figure 3.8: Compliance analysis overview: Elicitation and formalization of compliance rules.

After summarizing, the requirements that our compliance analysis approach (Fig. 3.3) should fulfill in each step, we now sketch our ideas to fulfill these requirements.

## 3.4 Elicit and Formalize Compliance Rules

As we discussed earlier during the elicitation and formalization step of our compliance analysis approach (shown in Fig. 3.8), we may decompose a compliance constraint into several compliance rules each focusing on a different dimension of a business process. In this section we will discuss the decomposition of compliance constraints into control-flow, data-aware, resource-aware, and temporal compliance rules. First, we describe a collection of compliance constraints covering all four dimensions identified in existing literature [3, 5, 7, 14, 19, 36, 38, 41, 42, 49, 55, 68, 69, 73, 88, 116–119, 132, 142, 149, 153]. Then, we show how (data-aware) Petri nets (see Chap. 2) can be used to capture the compliance constraints of each dimension by examples from our literature survey.

### 3.4.1 Atomic Compliance Constraints Found in Literature

We have collected compliance constraints that we found in the literature and categorized them along the dimensions of this framework into three different collections of compliance rules including *control-flow compliance rule repository*, *temporal compliance rule repository*, and *resource-aware and data-aware compli-*

Category (Rules)	Description
Existence (2)	Limits the occurrence or absence of an activity. [142], [14, 38], [55], [117], [118]
Bounded Existence (6)	Limits the number of times an activity must or must not occur. [38], [42]
Dependent Existence (6)	Limits the presence or absence of an activity with respect to existence or absence of another activity. [42]
Bounded Sequence (3)	Limits the number of times a sequence of activities must or must not occur. [38], [42]
Parallel (2)	Limits the occurrence of a specific set of activities in parallel. [117]
Precedence (10)	Limits the occurrence of an activity in precedence over another activity. [42], [38], [117], [118], [14], [49], [55]
Chain Precedence (4)	Limits the occurrence of a sequence of activities in precedence over another sequence of activities. [42], [38], [55]
Response (10)	Limits occurrence of an activity in response to another activity. [117], [42], [55], [38], [119], [14], [49]
Chain Response (4)	Limits the occurrence of a sequence of activities in response to another sequence of activities. [42]
Between (7)	Limits the occurrence of an activity within (between) a sequence of activities. [38]

Table 3.1: Categorization of control flow compliance rules.

*ance rule collection.* Tables 3.1, 3.3, and 3.2 show our collections of compliance rules.

Table 3.1 describes our collection of over 54 control-flow compliance rules. These rules have been classified into 10 categories. Each category includes several compliance rules. Note, the term “compliance rule” is reserved for atomic level of compliance constraints, i.e., we do not decompose a compliance rule further. Each compliance rule is parameterized over activities (e.g. activity *A*) or numeric parameters (e.g. governing bounds for repetitions).

Table 3.2 lists our collection of temporal compliance rules that constrain the time perspective of business processes. We collected about 15 temporal compliance rules distributed over 7 categories.

Table 3.3 lists our collection of over 10 compliance rules that constrain the data and the organization perspective of a business process.

Category (Rules)	Description
Instance Duration (2)	Limits the duration in which a control-flow rule instance must hold. [149]
Delay Between Instances (1)	Limits the delay between two subsequent instances of a control-flow rule [5, 68, 69, 88]
Validity (3)	Limits the time length in which an activity can be executed. [68, 69, 88, 149]
Time Restricted Existence (2)	Limits the execution time of an activity based on some calendar. [68, 69, 88]
Repetition (2)	Limits the delay between execution of two subsequent activities. [68, 69, 73, 88, 149, 153]
Time Dependent variability(1)	Limits choice of a process path among several ones with respect to temporal aspects. [68, 69, 88, 149]
Overlap (4)	Limits the start and completion of an activity w.r.t. the start and the completion of another activity. [68, 69, 88, 149]

Table 3.2: Categorization of the 15 temporal compliance rules.

Rule Description	Example
Four-eye principle: The principle that requires segregating the execution of critical tasks and associated privileges among multiple users. [3, 7, 19, 36, 41, 116, 119]	The person requesting purchase of goods should not be the one who approves it. A purchase order approval requires two signatures.
Authorization (Access control): A security principle that limits execution of activities or accessing a data object to authorized individuals [3, 7, 19, 36, 41, 116, 119].	Only a financial manager can approve a loan.
Two (three) (four)-way match: An accounting rule that requires the value of two different data objects to match [3].	All vendor invoices that are based on purchase orders should be matched with purchase orders (two-way matching).
Activity $T$ may/must (not) be executed if attribute $X$ has the value $v$ ; ( $X$ may be local to the activity $T$ or may appear anywhere in a trace) [3, 41, 119].	An account must not be opened in case risk is high. During ventilation, patient must receive "propofol" with dosage of (5mg).
Activity $T_1$ may/must (not) be executed if attribute $X$ has value $v$ at activity $T_2$ . (attribute $X$ is local to activity $T_2$ ) [3, 41, 119]	In case the respondent bank rating review is rejected during evaluation, an account must never be opened.
Activity $T$ must not change value of attribute $X$ [3].	Bank account data must not change during payment.
Value of attribute $X$ must not change after activity $T$ is executed. [3]	All invoices must be archived and no change must be made to the document.
Activity $T_1$ may occur only if the value of attribute $X$ is increased/decreased by activity $T_2$ with $d$ .	If gastric tube feeding cannot be increased by (1,20 kcal/ml), then use 'Erythromycin' (ICU medical guideline in a Dutch hospital (internal policy)).
If attribute $X$ has value $v$ , then resource $R$ must execute the activity. [3, 7, 19, 36, 41, 116, 119]	Loans with value more than 1000000 Euro must only be approved by CFO.
If activity $T_1$ is done by resource $A$ , then activity $T_2$ must be done by the same resource [36].	A customer complain must be handled with the same agent registered the customer request.

Table 3.3: Collection of data-aware and resource-aware compliance rules.

To enable automated techniques for compliance checking, it is important that informal text of compliance rules are specified correctly and precisely, describing exactly the behavior intended. Next, we will discuss how we formalize compliance rules.

### 3.4.2 Basic Considerations about Formal Compliance Rules

The formalization of compliance rules should enable automated compliance checking, it should precisely specify what is allowed and what is not allowed by the compliance rule to be checked, it should allow for all possible compliant behaviors, and it should enable us to focus only on the specified activities (i.e., it should be declarative).

As we discussed earlier in this chapter and Chap. 2, we use Petri nets to formalize compliance rules. The choice of Petri nets meets all the requirements we listed and allows us to build on past experience and tooling (ProM). Petri nets allow us to leverage the (data-aware) alignment technique for compliance checking. Consequently, we are able to check compliance of a process from different process perspectives including control-flow, time, data and resource and obtain the detailed diagnostics we need.

Leaving the available machinery for alignment technique aside, the choice for Petri nets allows us to describe declarative constraints in a precise manner.

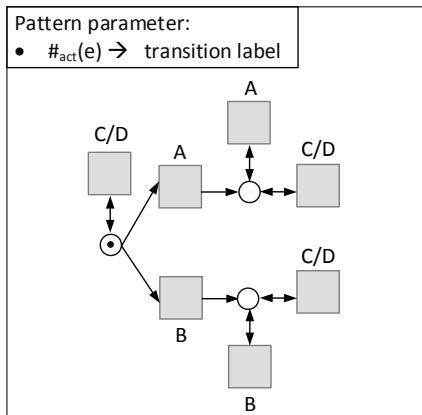


Figure 3.9: A Petri net modeling an exclusive relation between activities A and B.

For instance, for a process consisting of activities  $A$ ,  $B$ ,  $C$ , and  $D$  we would like to enforce a mutual exclusive relation between two activities  $A$  and  $B$ : “Either activity  $A$  or  $B$  must happen but not both”. The Petri net shown in Fig. 3.9 models this rule. As specified by the model, any activity may occur at the beginning. The transition  $C/D$  represents occurrences of activities  $C$  and  $D$ . As soon as  $A$  or  $B$  occurs the occurrence of the other one is not allowed. Other activities may occur an arbitrary number of times within a process.

In our approach, we use Petri nets in a “declarative style”. This allows us to specify only essential characteristics of process executions that need to be checked against a compliance rule (e.g. see the  $C/D$  transition in Fig. 3.9 representing all activities in a process other than  $A$  and  $B$ ). In this declarative style, we can abstract information in a process irrelevant to any specific rule. In addition, we are able to express all possible compliant behaviors (Note how the Petri net model in Fig. 3.9 captures all possible compliant behaviors and excludes only the violating behavior). Therefore, we overcome the hurdle of using imperative models (so called “over specification” [101, 133]) for formalizing compliance rules. In addition using Petri nets, we are able to capture subtle aspects of compliance rules that are necessary for precise checking of them and usually will be ignored using declarative techniques.<sup>2</sup>

### 3.4.3 Control-Flow Compliance Rules

In the remainder of this chapter, we explain the basic principles of formalizing compliance rules using Petri nets in a declarative style.

Suppose that we have a control-flow compliance rule stating: *response*: “Activity  $A$  must be followed directly by activity  $B$ ”. The Petri net shown in Fig. 3.10, formalizes this rule. As can be seen, we can replay the sequence of activities  $t_1 : \langle A, B \rangle$  over  $N_1$  without any problem, hence we say  $t_1$  is compliant with the rule. Note that by using the  $\Omega$ -labeled transitions, one can seamlessly navigate between the open world and closed world assumption.

<sup>2</sup>Note that by formalizing compliance rules as Petri nets, we do not face the difficulties of declarative techniques such as Declare [138], DCR Graphs [60], and SCIFF [83] being “too academic and less intuitive and less convincing for practitioners” [102]. Regardless of the notion, procedural modeling is perceived easier to read [102]. Worth to mention that we encountered the same observation during the evaluation of our approach in the case study in collaboration with UWV (presented in Chap. 9). We did not design a specific experiment whether declarative modeling or procedural modeling is perceived easier among domain experts. However, we received the feedback from domain experts that “it is easier to read and understand a process modeled in a procedural notion than declarative notion. We can understand the flow of a process more easily following the sequence of activities”.

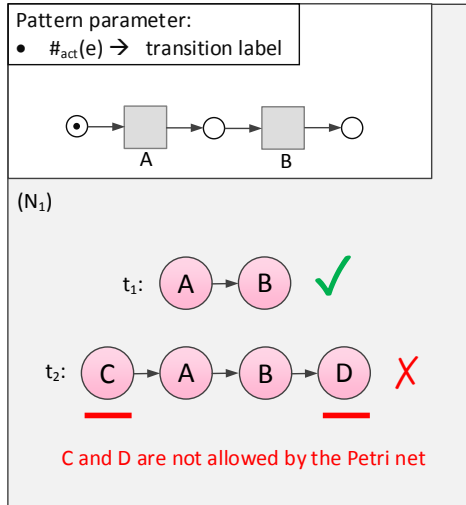


Figure 3.10: Petri net  $N_1$  formalizing the sequence  $\langle A, B \rangle$ .

**Abstract from irrelevant activities.**

Compliance rules usually focus on specific activities in a process and specify how an activity or sequence of activities must be executed. A business process may include many more activities than the specified ones. Some of these activities may be irrelevant for a given compliance rule. For example, the Petri net  $N_1$  shown in Fig. 3.10 is not a good representative for all possible compliant behaviors w.r.t. activity A and B (recall the open world assumption versus the closed world assumption discussed in Sect. 2.6). The sequence of activities  $t_2 : \langle C, A, B, D \rangle$  (shown in Fig. 3.10) is not violating the stated compliance rule, but is not allowed by the Petri net  $N_1$ .

In our “declarative style”, we overcome this problem by using an  $\Omega$ -labelled transition in the Petri net specification that represents occurrence of any activity in a process except the activities that are confined by the compliance rule in question. The Petri net  $N_2$  shown in Fig. 3.11, describes how the  $\Omega$ -labelled transitions adjacent to the *Initial* and *Final* places of the net allow for an arbitrary number of occurrences of other activities (excluding A and B). These activities may occur before and after the specified sequence  $\langle A, B \rangle$ . As shown in Fig. 3.11,  $N_2$  allows for the sequence of activities  $t_2 : \langle C, A, B, D \rangle$  without any problem.



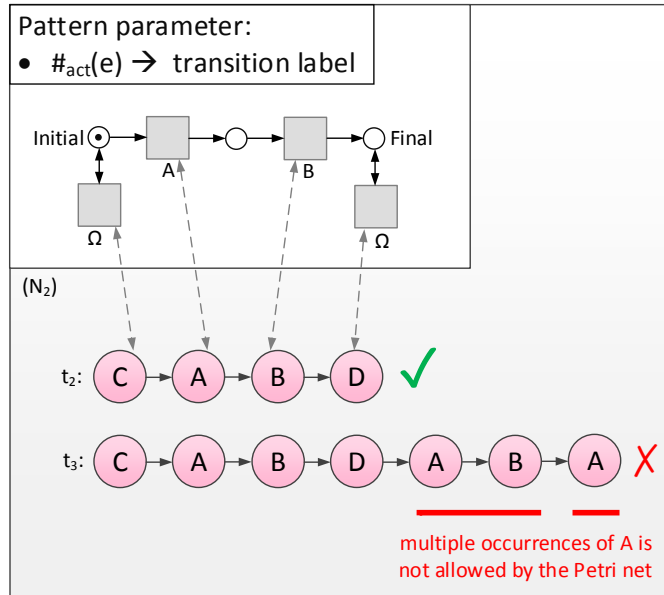


Figure 3.11: *Declarative style Petri net  $N_2$*  allowing for sequences where  $A$  is directly followed by  $B$  including  $\langle C, A, B, D \rangle$ .

### Several occurrences of a rule in the same execution.

Activity  $A$  may be executed several times throughout the life time of a business process instance. In principle, the *response* rule does not restrict the number of occurrences of activity  $A$ . For example, consider the trace  $t_3$  shown in Fig. 3.11. This trace does not violate the rule. However, the Petri net  $N_2$  does not allow for the second occurrence of  $A$  and its following activity  $B$  or the third occurrence of activity  $A$ . In addition in case of a violation, we would like to know which occurrence of activity  $A$  violated the rule. In  $t_3$ , the second occurrence of  $A$  is compliant with the *response* rule whereas the third one violates the rule. The third occurrence of activity  $A$  is not followed by  $B$ . Note that each occurrence of activity  $A$  activates the compliance rule (again) because the condition required for the compliance rule to hold is present.

To check the compliance rule for all occurrences of activity  $A$  and capture each occurrence as a separate activation of the compliance rule, we introduce

the *rule instance* concept. A *compliance rule instance* captures and marks activations of a compliance rule within a business process instance.

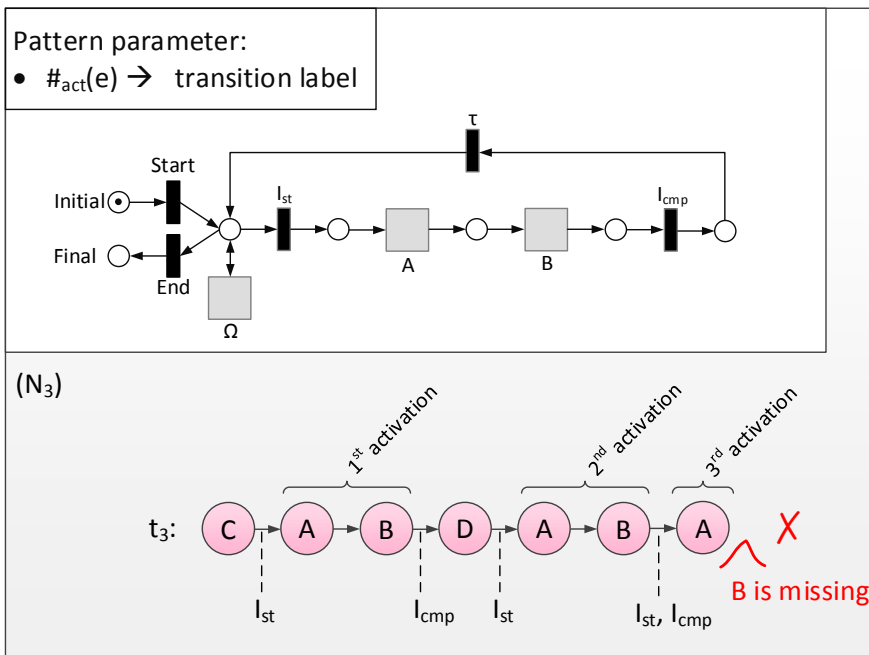


Figure 3.12: Declarative style Petri net  $N_3$  formalizing different instances of the compliance rule.

Figure 3.12 formalizes a version of the *response* compliance rule which allows us to model different instances of the rule and capture more diagnostic information about possible violations. This net ( $N_3$ ) starts by firing transition *Start*. A token in the place *Final* represents a completed process instance. The core part of the compliance rule is between transitions  $I_{st}$  and  $I_{cmp}$  which represents an instance of the compliance rule. When the condition of a compliance rule is present, an *instance* of the compliance rule is activated. In this example, each occurrence of A must be followed by B. Hence, each occurrence of A requires a new completion of the rule, i.e., a new rule instance. Firing transition  $I_{st}$  denotes activation of the *rule instance*. When the compliance rule is satisfied, i.e., the constraint of the compliance rule holds,  $I_{cmp}$  fires which denotes

the completion of the *rule instance*. The solid black transitions *Start*, *I<sub>st</sub>*, *I<sub>cmp</sub>*,  $\tau$ , and *End* are invisible. The  $\Omega$ -labelled activity after *Start* represents any other activity in a process apart from *A* and *B* that may occur before or after occurrence of  $\langle A, B \rangle$ . The invisible  $\tau$  transition allows for occurrence of several instances of the specified sequence.

By distinguishing different instances of a compliance rule in  $N_3$ , we are able to mark to which rule instance a violation belongs. As can be seen in the trace  $t_3$  in Fig. 3.12, the third instance of the compliance rule has a violation.

**Specifying subtle aspects of a rule as different variants of that rule.**

We can create variations of the *response* rule by adding some more elements to  $N_3$ . Consider for instance, the pink shadowed fragment of the net  $N_4$  (Fig. 3.13(a)). Adding an  $\Omega$ -labelled transition to the place *p*, allows for a relaxed version of the rule, i.e., an indirect sequence of *A* and *B*. Likewise, if we add the *B*-labelled transition shown in the blue shadowed fragment of the net (Fig. 3.13 (b)), we allow for occurrence of activity *B* independent from occurrences of *A*.

As we discussed above, the core behavior of a compliance rule is captured in a rule instance: any other elements added to or removed from the core behavior will change what is allowed or not allowed by the Petri net and it will lead to different variations of a compliance rule. Therefore, a detailed interpretation

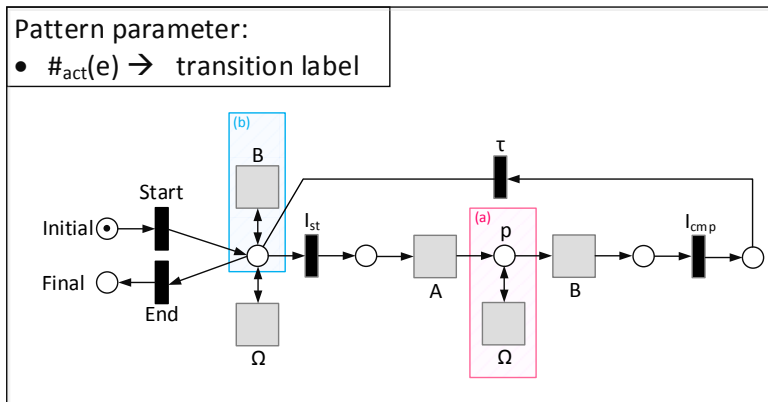


Figure 3.13:  $N_4$  formalizes a relaxed version of the response compliance rule by adding or removing some elements to  $N_3$ .  $N_4$  also allows for  $\langle B, A, C, B, A, B, B \rangle$ .

of a compliance rule is required to choose a Petri net that formalizes a compliance rule precisely for instance by answering following questions: *Are multiple instances of a rule allowed or not? Can activity B occur independently from A or not? Can other activities occur in between the sequence A and B or not?*

Formalizing a compliance rule that specifies precisely the behavior intended in the rule, can be difficult and time consuming especially if one is less familiar with different formalisms.

Therefore, we need an approach to let the business user find and formalize the right variant of a rule. In addition, the approach should allow us to express all variations of a rule in a concise form. We introduce an approach based on “disciplined natural language” that enables business users to formalize a compliance rule without being exposed to the technicalities of modeling in Petri nets. We will explain the details of this approach in Chap. 4.

### 3.4.4 Formalizing Temporal Compliance Rules.

Suppose the response compliance rule is extended with a temporal restriction stating, *response with time*: **“Activity A must be followed by activity B within 2 time units.”** To specify that activity B must occur within 2 time units after A, we first need to specify the occurrence of the sequence  $\langle A, B \rangle$ . Hence, we can use the control-flow pattern we discussed previously and extend it with restrictions on time. For this, we use *data-aware Petri nets* (see Chap. 2 for explanations about data-aware Petri nets). For instance, the  $DPN_{temporal}$  shown in Fig. 3.14 specifies the *response with time* compliance rule.

Variable  $t_A$  captures the time stamp of the events created as a result of firing transition A and  $t_B$  captures the time stamp of the events created as a result of firing transition B. This is specified by the two statements  $\{W : t_A\}$ , and  $\{W : t_B\}$  annotating transitions A and B. The *B-labelled* transition inside the *rule instance* is guarded. This guard confines the value of the time attribute for events created when this transition fires to be at most *two time units* later than the previous occurrence of the *A-labelled* transition.

As discussed in Sect. 3.4.1, we collected temporal compliance rules that we found in literature and practice in Table 3.2. In Chap. 5 we will introduce a generic approach to specify all these temporal compliance rules without exposing business users to the technicalities of data-aware Petri nets.

### 3.4.5 Formalizing Data-Aware and Resource-Aware Compliance Rules

Similar to temporal compliance rules, we use data-aware Petri nets to formalize data-aware and resource-aware compliance rules with the difference that instead of constraints on time attribute, constraints will be on data and resource related attributes. Suppose we extend the restriction on activity *A* in our example to be executed only by **an agent carrying management role when amount is more than 500**. The data-aware Petri net shown in Fig. 3.15 formalizes this compliance rule. Please note, this rule restricts only executions of activity *A*. Hence, an instance of this compliance rule starts and completes with an occurrence of activity *A*. Two variables *a* and *r* capture the values for attributes *amount* and *role* at executions of activity *A*. This is specified by the statements  $W : a, r$  annotating activity *A*. In addition, activity *A* is guarded. This guard has two parts. the statement  $(a \geq 500 \ \& \ r = \text{management})$  specifies: if an amount value at executions of activity *A* is more than 500, the activity must be executed by an agent carrying *management* role. The second part of the guard  $(a < 500)$  does not confine the agents executing activity *A* when *amount* value is less than 500.

Data-aware and resource-aware compliance rules fall in two categories. The first category of these rules assume the underlying control-flow of the compliance rule is correct and the restriction is put on the data or resource attributes of events. Our example is of this type; this rule does not put any restriction

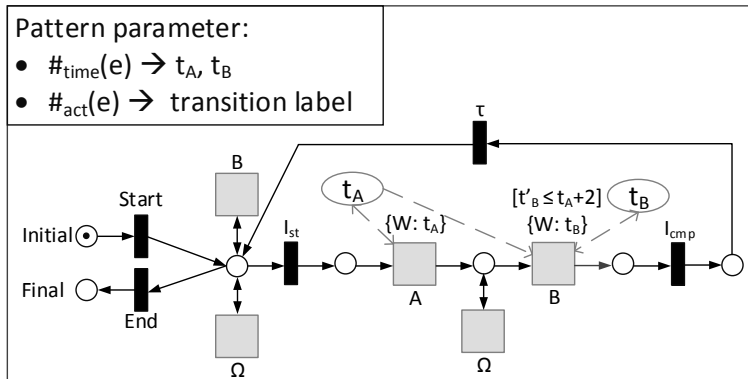


Figure 3.14: A data-aware Petri net formalizing the temporal compliance rule.

on the occurrence of activity *A* but on the role of the agent who executes this activity with a specific amount.

The second category of data-aware and resource-aware compliance rules, puts restrictions on the occurrences of an activity or sequence of activities when a resource or data condition holds. Suppose a data-aware compliance rule stating: “Activity *A* must never be executed for *Gold* customers”. In this rule, the *customer type* triggers the compliance rule and not occurrence of activity *A*. Therefore, we need to first capture all situations where the value of attribute *customer type* is *Gold* and then check whether activity *A* was executed or not in any of these situations.

We specify compliance rules of the second category with two different Petri nets: we classify the situations where the data or resource condition of the rule holds, using a data-aware Petri net and we specify the control-flow condition of the rule using a classical Petri net. We explain the specification and checking of these compliance rules in detail in Chap.6.

In this section, we sketched the idea of how we specify a single compliance rule individually using Petri nets. In the following we discuss the question about how to handle more complex (sets of) compliance rules.

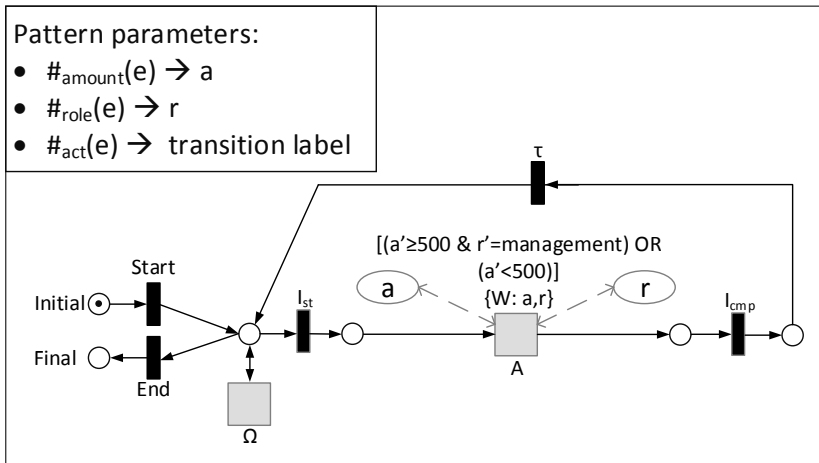


Figure 3.15: A data-aware Petri net confining the execution of activity *A* w.r.t. role and amount attributes.

### 3.4.6 Atomic Compliance Patterns Vs. a Composite Compliance Model

As we discussed above, a complex compliance constraint can be decomposed to several compliance rules. A compliance rule is the atomic level of compliance constraints that cannot be decomposed further. Each compliance rule can be formalized separately. On the other hand, each business process may be subject of several compliance constraints. Consequently, activities can also be constrained with multiple compliance rules. The question is: how we handle multiple compliance rules then?

A set of compliance rules that a business process must adhere to can be captured as a set of individual Petri net models called “*atomic compliance patterns*” or they can be specified as one large compliance model that we refer to as “*composite compliance model*”. Both essentially follow the ideas of Sect. 3.4 but differ in size and the diagnostics that we can obtain from them.

The choice to formalize a set of compliance rules for a business process as a set of atomic compliance patterns or as a composite compliance model, depends mostly on the possible mutual impact the rules may have on each other and on the process compliance. Consider two compliance rules stating:

- *Rule 1*: Activity *A* must be followed directly by activity *B*.
- *Rule 2*: Activity *D* must be preceded directly by activity *B* or activity *A*.

In the following, we show how these two rules can be modeled as two separate atomic compliance patterns or as a composite compliance model and discuss the differences in diagnostic information that can be obtained choosing either approaches.

#### **Modeling compliance rules as atomic versus a composite compliance model.**

The atomic Petri net patterns  $N_5$  and  $N_6$  shown in Fig. 3.16 specify the above mentioned rules separately. The Petri net pattern  $N_5$  describes Rule 1. If there is no *A* or just *B* occurred, any activity may occur. This is shown by activity  $\Omega$  that maps to any activity in a trace that is not *A* or *B*.

Rule 2 is described by the pattern  $N_6$ . This Petri net describes a cycle of *D* preceded by *A* or *B*. At first, any activity including *A* or *B* but *D* may occur, because the rule allows for more than one *A* or *B* before *D*. As soon as an activity *A* or *B* occurs, activity *D* may occur. After that, the cycle may repeat or other activities but *D* may occur.

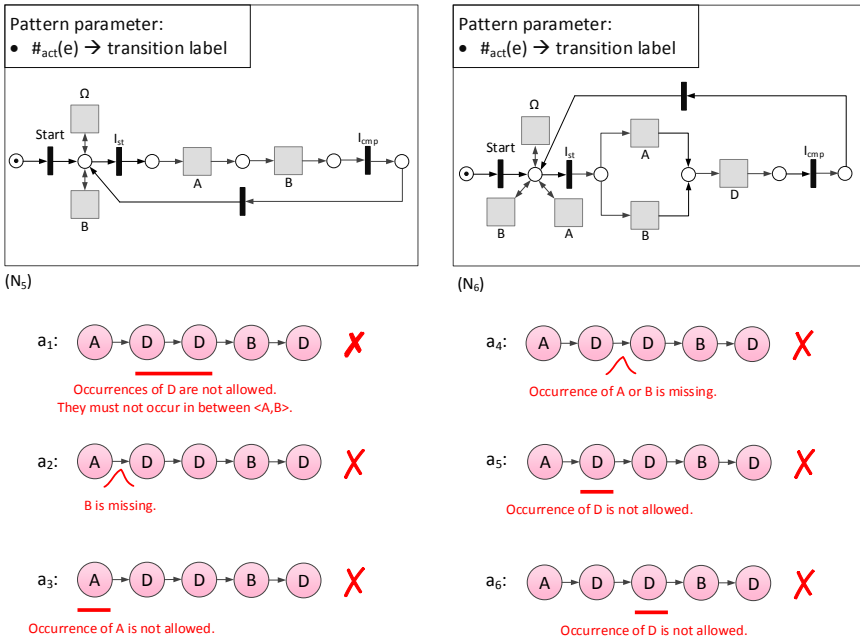


Figure 3.16: Violations of log trace  $t:\langle A,D,B,D \rangle$  against two micro compliance patterns.

The Petri net  $N_7$  (shown in Fig. 3.17) specifies both rules combined. If  $A$  occurs, it must be followed by  $B$  but  $B$  may occur an arbitrary number of times anywhere in the process. By combining two rules, it becomes explicit that activity  $D$  may only be directly preceded by  $B$ . As illustrated, combining compliance rules in a composite compliance model yields a model that is more specific than modelling the rules separately. Nevertheless, no matter if a constraint is modelled as several atomic patterns or as a composite compliance model, the conclusion whether a trace is violating the constraint or not would be the same.

**Diagnostics obtained from atomic compliance patterns versus a composite compliance model.**

Atomic compliance patterns and a composite compliance model do not only differ in size and the behavior they specify but they may differ in the diagnostics they provide as well. Consider the process instance  $t:\langle A,D,D,B,D \rangle$ . This process



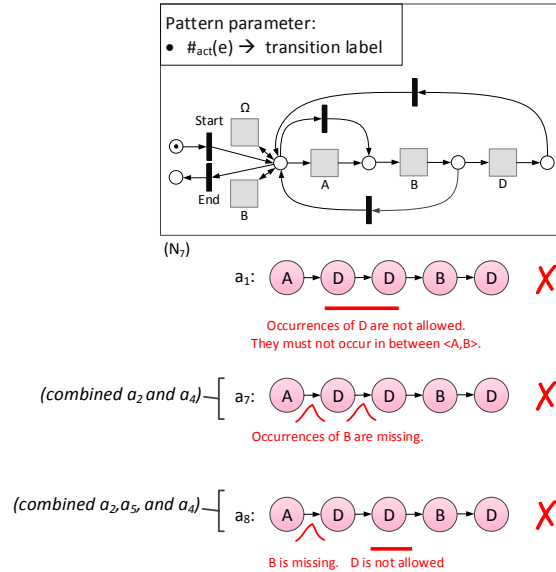


Figure 3.17: Violations of log trace  $t$  against the composite compliance model.

instance violates Rule 1 formalized as  $N_5$  (shown in Fig. 3.16). Three possible explanations can be given for the violating process instance  $t$ : either  $a_1$ ) activity  $D$  should not have occurred between  $A$  and  $B$ , but occurred twice or  $a_2$ ) an occurrence of  $B$  is missing after  $A$ , or  $a_3$ ) activity  $A$  should not have occurred. As we can see in Fig. 3.16 (bottom right), trace  $t$  violates Rule 2 as well. The diagnostics produced for  $N_6$  are  $a_4$ ) occurrence of  $A$  or  $B$  is missing between two occurrences of  $D$ , or in  $a_5$ ) and  $a_6$ ) one of the occurrences of  $D$  is not allowed. This example shows that each compliance pattern provides diagnostic information specific for the atomic patterns.

Figure 3.17 combines the two rules together as the composite model  $N_7$ . If we consider the combination of the two rules in  $N_7$ , then the diagnostics we obtain are globally optimized. Trace  $t$  is violating the rules specified in this model as well. However, the composite model  $N_7$  returns three different explanations for the violations as shown in Fig. 3.17:  $a_1$ ) the two occurrences of activity  $D$  is not allowed, or  $a_7$ ) two occurrences of activity  $B$  are missing, or  $a_8$ ) the occurrence of  $B$  directly after  $A$  is missing, and one of the occurrences of  $D$  is not allowed. Explanations  $a_7$  and  $a_8$  cannot be returned as diagnostics

information from  $N_5$  and  $N_6$  as they combine information about violations from both rules. Here, the composite rule  $N_7$  gives more insights than  $N_5$  and  $N_6$  together.

Modeling compliance rules as a set of atomic rules or as a composite model does not change the semantics, i.e., it will not impact *whether* a rule is violated, but it may impact the *quality* of diagnostic information. This statement holds only if the compliance rules are not conflicting. For example, suppose three rules given as “**A has to be followed by B**”, “**B has to be followed by C**”, and “**C has to be followed by A**”. It is possible to pick any of the rules and satisfy them, but there is no trace that can satisfy all three together. For one of the rules, the “last activity” will not come in a finite trace, i.e., the finite trace  $\langle A, B, C, A, B, C, A, B, C \rangle$  violates the third rule. Note that it is not useful to check a process against conflicting rules, however, when combining such rules we can set limitations to solve the confliction. For example allowing for termination of the process after three execution of the sequence  $\langle A, B, C \rangle$ .

#### **Further differences between atomic compliance patterns versus a composite compliance model.**

We discussed some of the differences between atomic patterns and a composite compliance model above. Note that, apart from these differences, the concept of “compliance rule instance” becomes blurred in a composite compliance model compared to atomic patterns. When rules are modeled as a composite model, we cannot assign a violation to a specific rule if rules overlap in activities. We will elaborate on this using an example. Suppose a trace  $t' : \langle A, D, B, D \rangle$  given. This trace violates the Rule 1 modelled in  $N_5$  but is compliant with Rule 2 modelled in  $N_6$  and violates the composite model of the two rules as  $N_7$ . When we combine rules together that have common activities, sometimes it is not possible to assign the occurrence of activities to one of the rules. Hence, if we are interested to know the compliance state of a business process w.r.t. compliance rules separately, it is better to formalize rules as individual atomic compliance patterns. If we combine compliance rules together as a composite compliance model, in many situations when rules overlap, we cannot pinpoint to a specific rule that is violated.

As we discussed the requirements of diagnostics in Sect. 3.2, detecting exactly the compliance rules that are violated and those that are not can become important if compliance rules in an organization are prioritized. Violating some compliance rules may have more severe consequences compared to others.

Formalizing compliance rules together as a composite compliance model

may lead to a very complex model as well. Especially if process must comply to many rules. Suppose we have the following three compliance rules that a business process must comply to:

- Activity *D* must occur at least 2 times.
- Activity *A* must be followed directly by activity *B* or activity *C*.
- Activity *G* must be preceded by activity *E* and *F*.

If the compliance rules are unrelated (i.e., violations of one will not influence the compliance of others), it would be better to consider them separately. The compliance rules mentioned above do not overlap in any activity or any condition. Formalizing these rules together will lead to a rather complex composite compliance model. A composite model for above mentioned rules is the “cartesian product” of their atomic compliance patterns that will be quite complex without adding any value.

Modeling compliance rules that are related, as a composite compliance model, will enable us to consider the global impact of these rules together on compliance. At the same time when rules are unrelated, we get more precise diagnostics if we formalize and check compliance rules separately.

In Chap. 5 and Chap. 6, we discuss some compliance rules that cannot be specified by a composite compliance model as precise as is possible by atomic compliance patterns. Furthermore, we have developed an automated approach to formalize compliance rules as atomic compliance patterns, whereas composite compliance models are unique in context of each process and there is less support for the automated elicitation of these models.

Apart from what we discussed above, any of these design choices have some consequences on detection of compliance violations and later on combining the results. Formalizing several compliance rules together as a composite compliance model, allows us to check all the rules with only one single check and the results will already include violations of different rules. Whereas if we formalize compliance rules as atomic compliance patterns, we need to check each rule in isolation and then we need to combine the checking results. We will discuss this further in Chap. 8.

In this section, we sketched the idea of how we specify a set of compliance rules for a business process. Chapters 4, 5, and 6 will present the details of control-flow, temporal, and data-aware rules. Next, we will discuss our ideas on how to check compliance of an event log against the specified compliance rules.

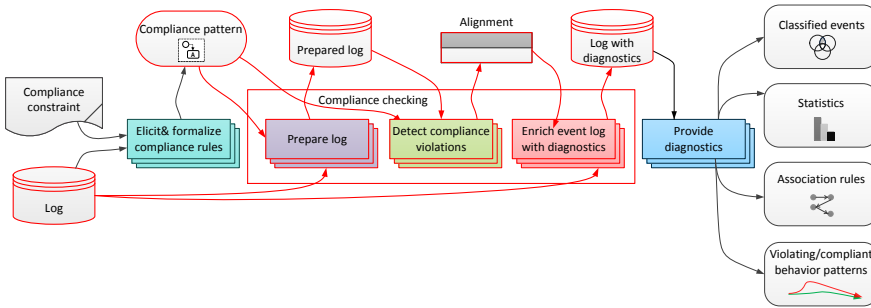


Figure 3.18: Compliance analysis overview: Compliance checking.

## 3.5 Compliance Checking

Checking compliance of an event log against a prescribed behavior, modeled in terms of a Petri net (either as an atomic compliance pattern or a composite compliance model), includes the three main steps as shown in Fig. 3.18. These steps may be repeated during the checking. The steps are: log preparation, detection of compliance violations, and combining violation detection results. Next, we will discuss different steps in compliance checking.

### 3.5.1 Log Preparation

Previously in Chap. 2 and Sect. 3.4, we discussed that compliance rules have underlying conditions that specify when a certain compliance rule must hold and we use checking attribute to be able to precisely identify the situations under which the condition of the rule holds. In this section with the help of an example we will explain how we prepare event logs to capture the condition of a rule as a checking attribute.

For example, consider the compliance rule taken from a medical guideline related to Intensive Care Unit (ICU) procedures stating: “If nutrition with multi-fiber cannot be increased in tube feeding at least by (2 kcal/ml), then Demperidone must be administered to the patient”. According to the rule, it is expected that execution of activity “tube feeding” increases the “nutrition with multi-fiber” by 2 kcal/ml. If this increase does not happen, then the rule states that the medicine “Demperidone” must be administered to the patient. Suppose the dosage of “nutrition with multifiber” is captured in an attribute *X*. This attribute

is accessed by the activity “tube feeding (T)”. Rephrasing the rule with a less domain specific jargon, it states: “If value of attribute  $X$  is not increased by  $2 \text{ kcal/ml}$  at activity  $T$ , then activity  $A$  must be executed”.

This compliance rule requires that activity  $A$  must be executed when value of attribute  $X$  was not increased by  $2 \text{ kcal/ml}$ . Therefore, we only should check the rule for situations that attribute  $X$  was not increased or increased but less than  $2 \text{ kcal/ml}$ .

Consider the simplified trace related to a patient shown in Fig. 3.19. Three executions of activity  $T$  are shown. At the third execution of  $T$ , the expected increase in the value of attribute  $X$  is not observed. According to the rule, the execution of  $A$  after the third  $T$  is expected. The log is prepared with a new attribute as the checking attribute which indicates where the condition of the rule holds.

In our example, the checking attribute captures the occurrence of activity  $T$  together with the change in the value of attribute  $X$ . Later we check whether for those events that had no increase or an increase of less than  $2 \text{ kcal/ml}$ , activity  $A$  was executed or not.

The checking attribute captures a specific value at each event when the condition of the rule in question holds. Then we can detect whether the compliance rule is followed for the events having that specific value at their checking attribute. Values of a checking attribute can be: simply a copy of one of the event attributes, the combination of several event attribute values, or a calculated

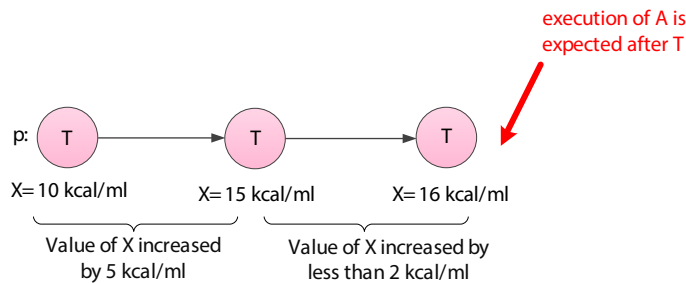


Figure 3.19: Execution of activity *Demperidone administration (A)* is required after the activity *tube feeding (T)* when the value of *nutrition with multifiber (X)* was not increased by  $2 \text{ kcal/ml}$ .

value based on one of the event attributes or combination of them.

**Choice of checking attribute for checking control-flow compliance rules.**

When we check a control-flow compliance rule, the checking attribute is the *activity name* itself, however the checking attribute gets a different value at events that are not restricted by the compliance rule. Recall the  $\Omega$ -labelled transitions in our “Petri nets in declarative style”, whenever an event is represented with an  $\Omega$ -labelled, it gets the value  $\Omega$  for its checking attribute during log preparation. This allows us to focus only on specific events that are restricted by the compliance rule and abstract from all other events that are not relevant to the compliance rule.

**Choice of checking attribute for checking temporal compliance rules.** Similar to control-flow compliance checking, when we check a temporal compliance rule, the checking attribute is the *activity name* of particular activities. We check whether those activities occurred at a right time. For instance, if a compliance rule states that “activity *A* must be followed directly by *B* not later than 2 time units”, The checking attribute should capture the executions of *A*, and the first *B* after it but not other occurrences of activity *B* (*B* events not following an *A*).

**Choice of checking attribute for checking data-aware and resource-aware compliance rules.** When checking a data-aware or a resource-aware compliance rule, the checking attribute captures the execution of specified activities or combination of the execution of the specified activity together with the data/resource that they have been executed with. If the rule is of the first category of data-aware and resource-aware rules, the checking attribute will be the *activity name* of the specified activities. For the rules of the second category, the checking attribute captures the combination of *activity name* and the data/resource attributes. For instance instance, consider a compliance rule stating: “Activity *C* must be executed by a resource carrying a *management* role”, here the checking attribute should capture the execution of activity *C*. For the rule stating: Loans with value more than 500 must be approved by *management* role, the checking attribute should capture executions of activity *approval* and *value*.

We will discuss in chapters 4, 5, and 6 dedicated, yet generic, techniques for log preparation for checking control-flow, temporal, data-aware, and resource-aware compliance rules. Enriching events in an event log with suitable checking attributes is essential for correct and precise detection of compliance violations.

**Log preparation for checking atomic compliance rules versus a composite compliance model.** When we formalize compliance constraints that a process must adhere to as a set of atomic compliance patterns, we introduce an ap-

appropriate checking attribute separately for checking each rule. This allows for a precise detection of compliance violation tailored for each compliance rule. Introducing a checking attribute for a composite compliance model, is much more difficult because each rule needs to be checked under its specific condition. Consequently, the checking attribute needs to have many values which is the result of combining different underlying conditions for different compliance rules.

Therefore, we use activity names as the checking attribute to basically map events to transitions and create multiple variables in the composite compliance model and map other attributes of events to them. Consequently, not all the compliance rules that can be checked using atomic compliance rules can be checked in a composite model. We will elaborate on this further in chapters 4, 5, and 6.

### Shortening event logs

Apart from the log abstraction (i.e., enriching events with suitable checking attribute), we have another log preparation step that helps us abstract from information in an event log that are not required for detecting compliance violations. This log preparation step is not essential for the compliance checking, however, it optimizes the performance of detecting compliance violations by making the event log smaller. We will explain the details of this step in Chap. 4.

This step of log preparation is in principle the same in case of both atomic compliance patterns and a composite compliance model. With the difference that in case of a composite compliance model, the amount of information that we can abstract from is less compared to considering the same set of compliance rules separately as a set of atomic compliance patterns.

### 3.5.2 Detection of Compliance Violations Using Alignments

As discussed earlier, compliance rules put restrictions on different process perspectives (e.g. control-flow, data, resource, and time). However, regardless of the compliance rule type, the detection of compliance violations should provide the diagnostics we listed in Sect. 3.2. The core technique we use for detecting compliance violations is the conformance checking technique based on alignments explained in Chap. 2. This technique constructs an alignment between the events in the event log (observed behaviour) and the Petri net model that specifies the compliance rule to be checked (prescribed behavior). Depending on the rule to be checked we construct control-flow alignment or data-aware

alignment. Next, we will explain how we leverage conformance checking for detecting compliance violations w.r.t. different rules.

We first discuss the detection of control-flow compliance violations and later we explain the detection of violations for temporal, data-aware, and resource-aware compliance rules.

### 3.5.3 Detecting Control-Flow Compliance Violations Using Alignments

Recall from Sect. 3.3 that we aim at producing detailed diagnostic information about which compliance violations happened and which compliant behavior should have happened instead. In Chap. 2, we discussed how alignments give detailed diagnostics about differences between specified and observed behavior. In the following, we show how to align an event log to a Petri net model that formalizes a control-flow compliance rule (as explained in Sec. 3.4) allows us to detect detailed control-flow violations.

Suppose we have an event log with the events all related to one process instance  $P_L$  as shown in Fig. 3.20. We would like to check compliance of this log against a compliance rule stating: “**Activity A must be followed directly by the sequence of activities  $\langle B, D \rangle$** ”. As can be seen in the log, the occurrence of activity  $D$  is missing and log is violating the rule.

By choosing the activity name as the checking attribute, we can restrict the order of occurrences of activities by a compliance pattern. Next we illustrate how alignments using the activity name as the checking attribute detect control-flow violations.

We prepare the log by adding to each event the checking attribute with value  $A$ ,  $B$ , or  $D$  if the value of activity name is  $A$ ,  $B$  or  $D$  respectively and  $\Omega$  otherwise.

event ID	process instance	events timestamp	activity names	resource	amount
$e^L_1$	$\#_{pi}(e^L_1)=p$	$\#_{time}(e^L_1)=1$	$\#_{act}(e^L_1)=G$	$\#_{resource}(e^L_1)=John$	$\#_{amount}(e^L_1)=100$
$e^L_2$	$\#_{pi}(e^L_2)=p$	$\#_{time}(e^L_2)=2$	$\#_{act}(e^L_2)=A$	$\#_{resource}(e^L_2)=Sara$	$\#_{amount}(e^L_2)=100$
$e^L_3$	$\#_{pi}(e^L_3)=p$	$\#_{time}(e^L_3)=6$	$\#_{act}(e^L_3)=B$	$\#_{resource}(e^L_3)=Sara$	$\#_{amount}(e^L_3)=100$
$e^L_4$	$\#_{pi}(e^L_4)=p$	$\#_{time}(e^L_4)=7$	$\#_{act}(e^L_4)=C$	$\#_{resource}(e^L_4)=Sara$	$\#_{amount}(e^L_4)=100$

$L=(E^L, \#, \leq)$

$E^L = \{e^L_1, e^L_2, e^L_3, e^L_4, e^L_5\}$

Figure 3.20: Events in the log  $L$ .



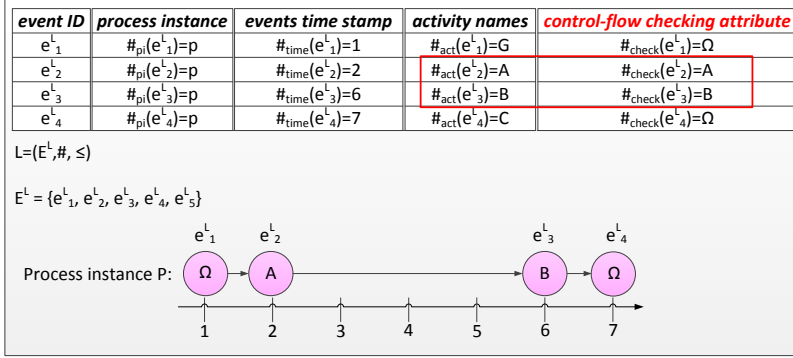


Figure 3.21: Preprocessed event log after adding control-flow checking attribute and shortening the event log.

Note that  $e_1^L$  and  $e_4^L$  are now  $\Omega$  events. Figure 3.21 shows some of the attributes of the events in  $L$ , and *checking attribute* for the rule in question and its values (Note that activity  $D$  is not executed). Using this checking attribute, the events can be directly mapped to the transitions of the Petri net model shown in Fig. 3.22 that formalizes the above compliance rule.

Recall the principles of atomic Petri net patterns discussed earlier in Sect. 3.4.3. An instance of this rule includes the sequence of activities  $\langle A, B, D \rangle$ . In this figure three process instance runs ( $P_{R1}$ ,  $P_{R2}$ , and  $P_{R3}$ ) are shown from the infinite set of possible process instance runs of this Petri net pattern.

Figures 3.23, 3.24, and 3.25 show three alignments between the events in process instance  $P$  and these process instance runs.

The alignment between  $P_{R1}$  and the net of Fig. 3.22 shows a model-only move (see Sec. 2.4); the model-only move indicates that activity  $D$  was expected according the compliance rule specified by the net, but did not occur. The alignment between  $P_{R2}$  and the net of Fig. 3.22 shows a log-only move, indicating that activity  $A$  occurred where it was not allowed. The third alignment shown in Fig. 3.25 indicates two violations: a model-only move related to the skipped activity  $D$  and one log-only move related to execution of activity  $A$  that was not allowed. This way, alignments and their model-only moves and log-only moves produce exactly the diagnostics about violations and alternative compliant behavior we described in Sec. 3.2 for control-flow compliance rules.

The cost function of an alignment allows us also to fine-tune which speci-

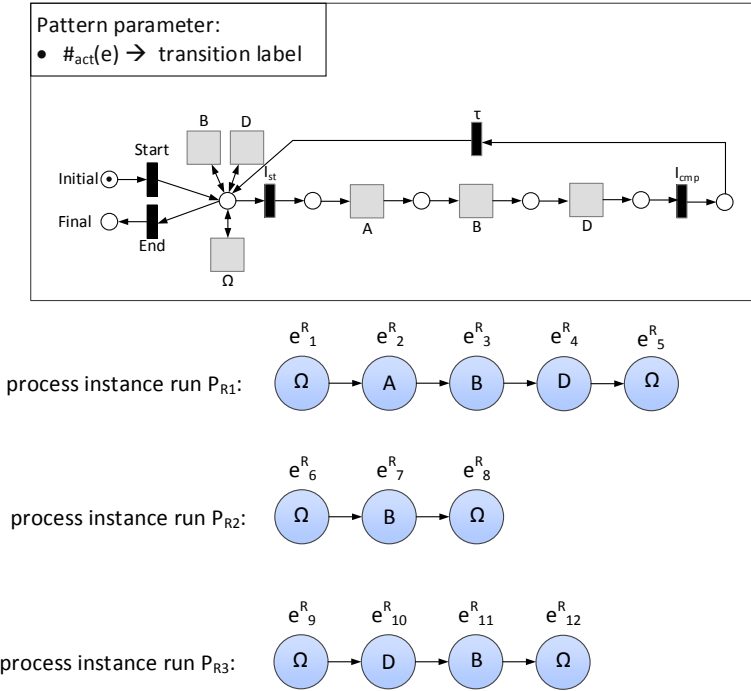


Figure 3.22: Three process instance runs  $P_{R1}$ ,  $P_{R2}$ , and  $P_{R3}$  of the Petri net pattern.

fied behavior is returned as the compliant behavior that should have happened. Assuming that the cost of compliant moves is *zero*, and the cost of any violation is *one*, the total cost of violations for the first second and third alignment respectively are *one*, *one*, and *two*. In this case both the first and the second alignment can be considered as the *optimal (best) alignment*. However by assigning a higher cost for skipping activity *A* we can make sure that the alignment technique returns a run in the model that is the closest to the behavior observed in the log. Therefore, by adjusting the cost function based on the compliance rule, we can guide the selection of a suitable alignment. In this thesis and all the experiments we did, we used the cost function implemented in [10, 134]. We will discuss the role of the cost function for producing particular kinds of diagnostic information in detail in later chapters.

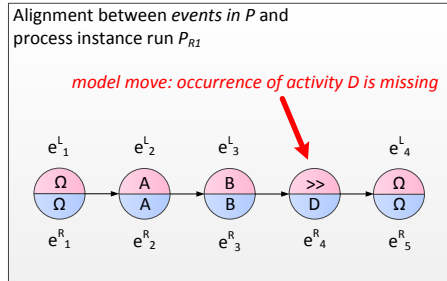


Figure 3.23: An alignment between pre-processed event log of Fig. 3.21 and the process instance run  $P_{R1}$  (in Fig. 3.22).

### 3.5.4 Detection of Temporal Compliance Violations Using Data-Aware Alignments

Temporal compliance rules restrict *when* activities may occur. Suppose there is a time restriction on the specified sequence of  $\langle A, B \rangle$  of our example as well stating: “The delay between occurrence of A and B must be within 2 time units”.

To check compliance of an event log against a temporal rule, we need to first check if the activities in question occurred at all. And if they occurred, whether they occurred at the right time. If it did not occur, depending on the rule to be checked either it is considered as a control-flow violation or it will be ignored (i.e., the temporal rule is not triggered). The data-aware Petri net

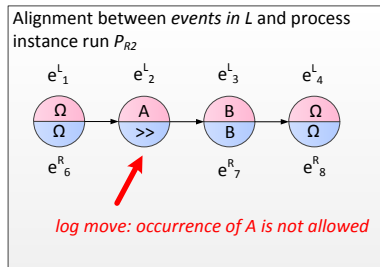


Figure 3.24: An alignment between between pre-processed event log of Fig. 3.21 and the run  $P_{R2}$  (in Fig. 3.22).

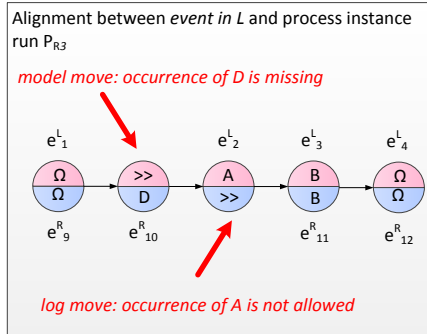


Figure 3.25: An alignment between pre-processed event log of Fig. 3.21 and the run  $P_{R3}$  (in Fig. 3.22).

pattern formalizing this rule is shown in Fig. 3.26. We discussed the details of this pattern earlier in Sect. 3.4.2.

To check if a specific event occurred at all, we first align log  $L$  with the compliance pattern  $DPN_{temporal}$  without considering time value of events. Note that compared to the previous example, in the temporal rule we only restrict the delay between the two activities  $A$  and  $B$ . Hence, we will not model activity  $D$  in  $DPN_{temporal}$ . The result of this step is an alignment highlighting control-flow deviations of the rule to the log. In our example of Fig 3.27 no control-flow deviations are identified.

Next, we enrich the original event log with information from the control-flow alignment of Fig. 3.27. This allows us to check adherence of temporal constraints for all those events that are relevant to the rule and occurred in the correct order. The temporal restriction itself is expressed through guards that constrain the *time* attribute of events in  $DPN_{temporal}$  (as explained in Sec. 3.4.4). We check whether activity  $B$  was executed within 2 *time units* after execution of  $A$ . Figure 3.28 shows the resulting data-aware alignment. The data-aware alignment detects the temporal violations and returns a compliant value for violations based on the specification in the data-aware Petri net  $DPN_{temporal}$ .

Since there is no temporal restriction on the first  $A$ -labelled transition, the event  $e_2^S$  that describes the occurrence of  $A$  in the run, gets the same value as its corresponding log event  $e_2^L$ , i.e.,  $\#_{time}(e_2^R) = 2$ . Consequently variable  $t_A$  will get the same value. Hence, the event  $e_3^R$  is only allowed to get a value between 2 and 4 as its time value, the guard on the  $B$ -labelled transition in  $N$

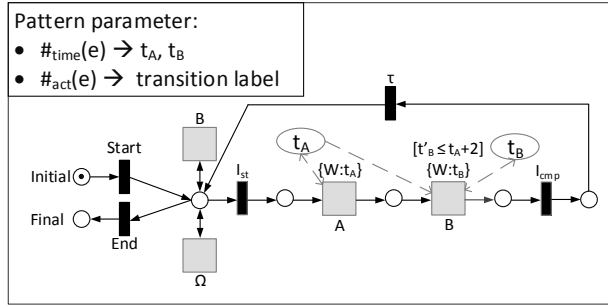


Figure 3.26: The temporal compliance rule formalized as the data-aware Petri net  $DPN_{temporal}$ .

specifies that. However, as event  $e_3^L$  has time value 6, the alignment reports a *temporal violation*. The compliant value returned by the data-aware alignment is a value in the interval  $[2, 4]$  which is taken from the time interval allowed by the prescribed behavior in  $DPN_{temporal}$ . In Sect. 3.5.7, we show how to enrich the event log with the diagnostics we obtained during checking.

### 3.5.5 Detection of Data-Aware and Resource-Aware Compliance Violations Using Data-Aware Alignments

Checking data and resource-aware compliance rules follows the same principle as temporal compliance checking, i.e., by constructing data-aware alignments, with the difference that instead of constraints on time attribute, constraints will

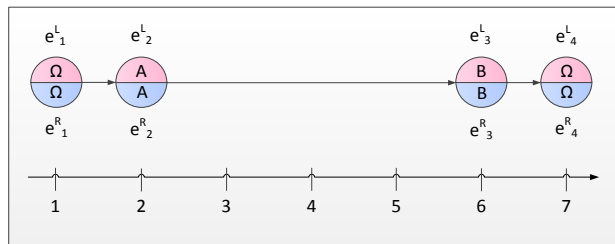


Figure 3.27: The control-flow alignment resulting from the alignment of log  $L$  (in Fig. 3.20) and  $DPN_{temporal}$  (in Fig. 3.26)

be on data and resource attributes. However, as explained in Sect. 3.4.5, there are two types of data-aware compliance rules that require different approaches. We explain both next.

**Data on top of control-flow.**

We discussed earlier that the first category of data-aware and resource-aware compliance rules assumes that the underlying control-flow of the compliance rule is correct, and the restriction is put on the data or resource attributes of events. Suppose a resource-aware compliance rule stating “Activity *C* must be executed by resource *John*”. This rule does not put any restriction on the occurrence of activity *C* but on the resource who executed this activity. Checking this compliance rule is very similar to temporal compliance checking described in Sect. 3.5.4. We first detect the occurrences of activity *C* and then we check if this activity was executed by resource *John* or not. The data-aware Petri net in  $DPN_{resource}$  in Fig. 3.29 specifies this compliance rule. An instance of this rule includes an occurrence of activity *C*. The value of attribute *resource* executed this activity is captured in the variable *r*. The guard annotating transition *C* specifies that activity *C* is only allowed to be executed by “*John*”.

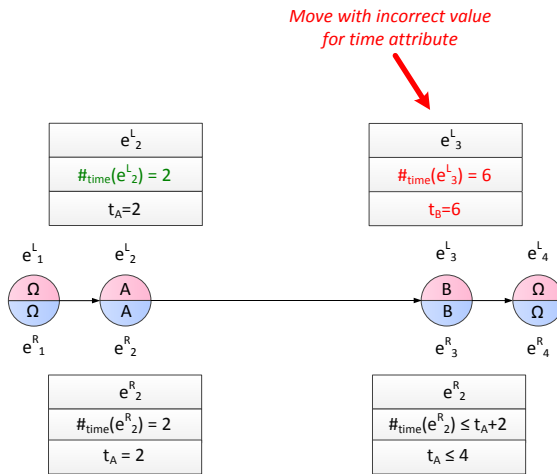


Figure 3.28: The data-aware alignment resulting from the alignment of log *L* and data-aware Petri net  $DPN_{temporal}$ .

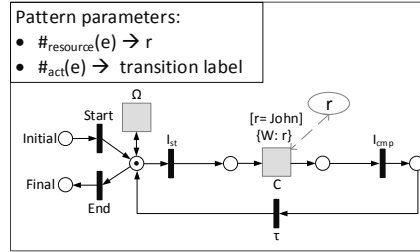


Figure 3.29: The resource compliance rule formalized as a data-aware Petri net  $DPN_{resource}$ .

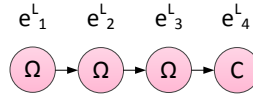


Figure 3.30: The event log  $L$  of Fig. 3.20 is prepared for checking the resource compliance rule.

We prepare the log  $L$  as we did for previous examples with the checking attribute *activity name* to check this rule. The prepared log is shown in Fig. 3.30. In the prepared log,  $e_4^L$  has for the checking attribute *activity name* value “C” whereas all other events have value  $\Omega$ . According to the original event log shown in Fig. 3.20, event  $e_4^L$  has “Sara” for its resource attribute. Figure 3.31 shows the alignment of the log  $L$  with  $DPN_{resource}$ . As can be seen, the log is violating the rule and activity  $C$  should have been executed by “John”.

**Control-flow on top of data.**

The second category of data-aware and resource-aware compliance rules restricts the occurrence of an activity or sequence of activities when a resource or data condition holds. Suppose a data-aware compliance rule stating: “**Whenever value of attribute amount is higher than 100, activity C must not be executed**”. In this rule, the value of attribute *amount* triggers the compliance rule but not occurrence of activity  $C$ . Therefore, we first need to identify all situations where the value of attribute *amount* is higher than 100 and then check if in any of these situations activity  $C$  was executed or not. For this type of data-aware and resource-aware compliance rules, we first need to construct a

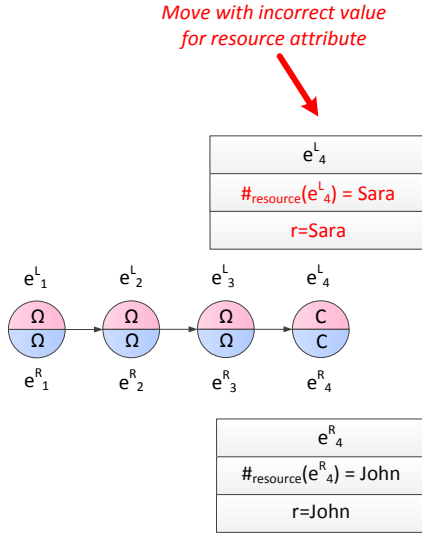


Figure 3.31: Data-aware alignment resulted from aligning the prepared log of Fig. 3.30 and  $DPN_{resource}$  of Fig. 3.30.

data-aware alignment to detect situations where the data condition of the rule holds and only then construct a control-flow alignment to detect if under the specified data condition a certain activity was executed or not. We will elaborate more on the details of this approach in Chap. 6.

### 3.5.6 Summary: Detecting Violations in all Dimensions

Figure 3.32 summarizes the steps we take for checking each type of compliance rule including control-flow, temporal, data-aware, and resource-aware compliance rules. As can be seen, we have dedicated checking techniques for each type of compliance rule. However, in principle all of these techniques combine log preparation, control-flow alignment, data-aware alignment, and log enrichment.

The input for control-flow compliance checking is a control-flow compliance rule formalized as a Petri net and an event log. After log preparation which includes log abstraction and shortening the event log, the prepared log is aligned with the Petri net pattern using control-flow checking technique. The resulting



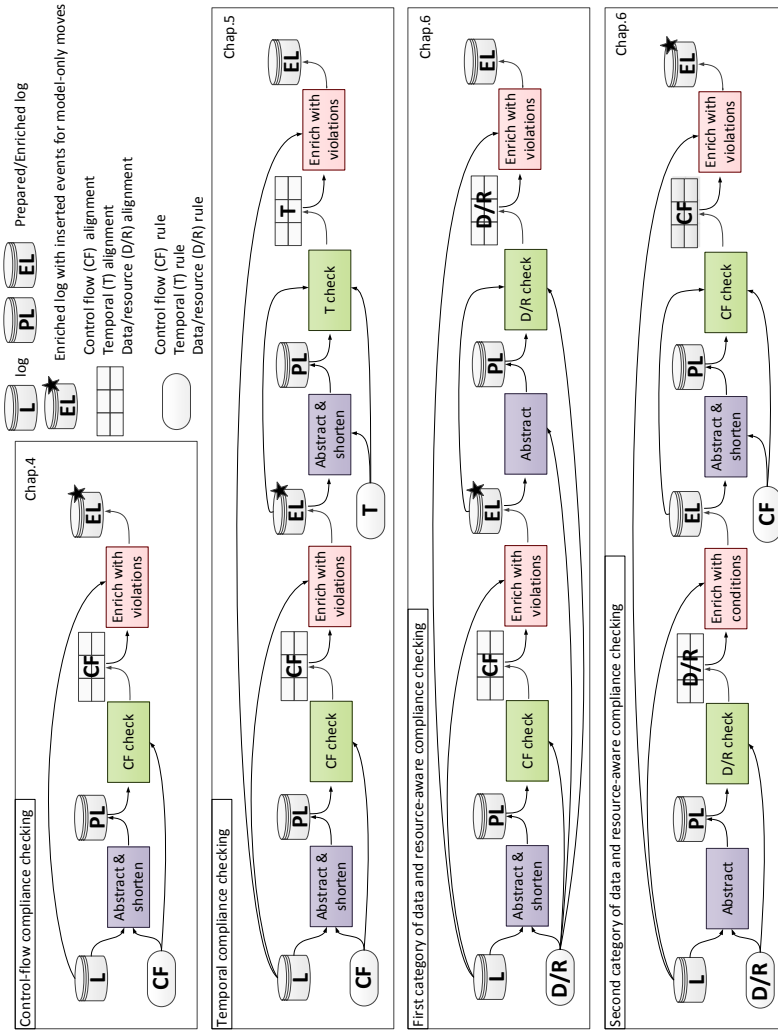


Figure 3.32: Steps taken in each compliance checking techniques. Each dedicated compliance checking technique includes iterations of three main steps: log preparation, checking and log enrichment.

alignment with diagnostics about violations are used to enrich the original event log.

Temporal compliance checking follows similar steps as control-flow checking. After enriching the event log with diagnostics obtained from control-flow checking, the enriched log needs to be prepared for temporal checking. This preparation also includes log abstraction and shortening of the event log. The prepared log and a temporal rule are used to build a data-aware alignment indicating all temporal violations. These diagnostics are then used to enrich the original event log with temporal violations.

Data-aware and resource-aware compliance checking may follow two different procedures according to the type of the rule to be checked. The first category of data and resource-aware compliance rules follows almost the same procedure as sketched for temporal compliance checking. In both approaches, first we check whether the activities in question occurred or not and we build data-aware alignment to check the temporal or data/resource condition of the rule. There is a difference, however, in the inputs of the approach. In temporal compliance checking the temporal rule and its underlying control-flow condition are formalized as two different Petri nets, while in data and resource-aware checking (first category) both control-flow condition and the data or resource restriction are modelled as one pattern.

In the second category of data and resource-aware compliance rules, the order of building control-flow alignment and then data-aware alignment is reversed. We first detect the conditions where the rule must hold by building a data-aware alignment and then we check whether on those situations the underlying control-flow restriction holds or not. Note that in this approach two separate patterns are used. Details of each technique will be discussed in the following chapters 4, 5, and 6.

Finally, it is worth noting that the actual checking of a composite compliance model does not differ from checking an atomic compliance rule. One can simply apply any of the compliance techniques for control-flow/temporal and data-aware checking also on composite compliance models. The advantages and disadvantages for using atomic rules or composite models as discussed in detail in Sect. 3.4.6 are entirely determined by the scope of the specification and not by the checking technique. Note that one can even specify both data-aware and temporal constraints in one composite model and check them together as the underlying checking techniques are identical. One should be aware that checking composite models of some data-aware constraints and some temporal constraints requires a more specific approach. The details will be discussed in chapters 5 and 6.

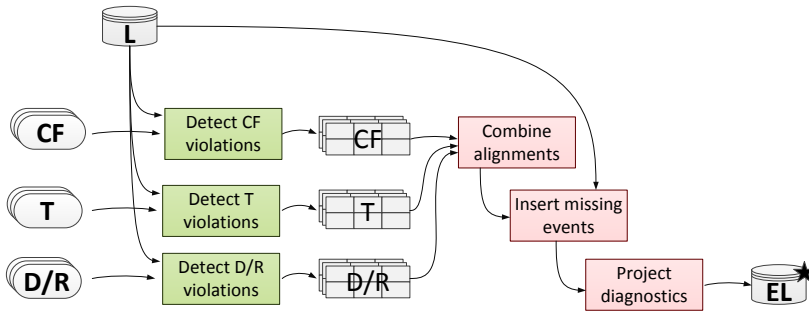


Figure 3.33: Generating enriched event log with diagnostics obtained from several alignments.

### 3.5.7 Combining Compliance Violation Detection Results

The compliance checking techniques described up to now return diagnostic information as an alignment between a log and a compliance rule. Some of the paths described in Fig. 3.32 produce multiple alignments (i.e., one for control-flow violations and one for temporal violations). Recall that our aim is to provide integrated insights into compliance violations by answering questions four and five of Sect. 3.3.1. Thus, we have to integrate the diagnostic information from different alignments. Technically, we choose to do this integration by enriching the original event log with diagnostic information from the different alignments. This will allow us to leverage existing process mining techniques and data mining techniques that operate on event logs to answer questions four and five of Sect. 3.3.1. We enrich the event log as follows:

We map moves to events. Synchronous moves and log-only moves have existing events that are enriched with further attributes. Artificial events need to be created for model-only moves with new attributes containing diagnostic information. In general enriching the original log with the diagnostics obtained during checking includes three main steps shown in Fig. 3.33:

1. Combining detected violations during the checking.
2. Inserting artificial events for *missing events*, i.e., where a model-only move was detected during compliance checking.
3. Projecting diagnostics on the original event log with inserted events by adding a *rule attribute* for each compliance rule that is checked. This

attribute is a nested attribute that is added to all the events and captures the *compliance state* of every event w.r.t. that rule, the *rule instance*, and in case it is a violating event, its *violation type*, and *compliant value* of every event.

Recall that using the approach described in sections 3.5.2 and 3.5.3, we may check the log  $L$  of Fig. 3.20 against three different compliance rules, yielding three alignments as shown in figures 3.23, 3.28, and 3.31). Applying the above steps yield the log shown in Fig. 3.34 (using the tree representation explained in Sect. 2.2); the diagnostic information added to the log is hachured.

As can be seen each event in the original log gets three nested attributes representing the rules for which its compliance has been checked. Each of these attributes have child-attributes that store the diagnostics related to that specific rule. For instance, we can see that the second event  $e_2^L$  has been compliant to all the three rules (see the attribute '*rule1. compliance state*', '*rule2. compliance state*', and '*rule3. compliance state* of event  $e_2^L$ ). It has been part of the first activation of the control-flow rule instance and also the first activation of the temporal rule instance, therefore it has been outside a rule instance when checking the resource-aware compliance rule. The resource-aware rule is only concerned with the executions of activity  $C$ .

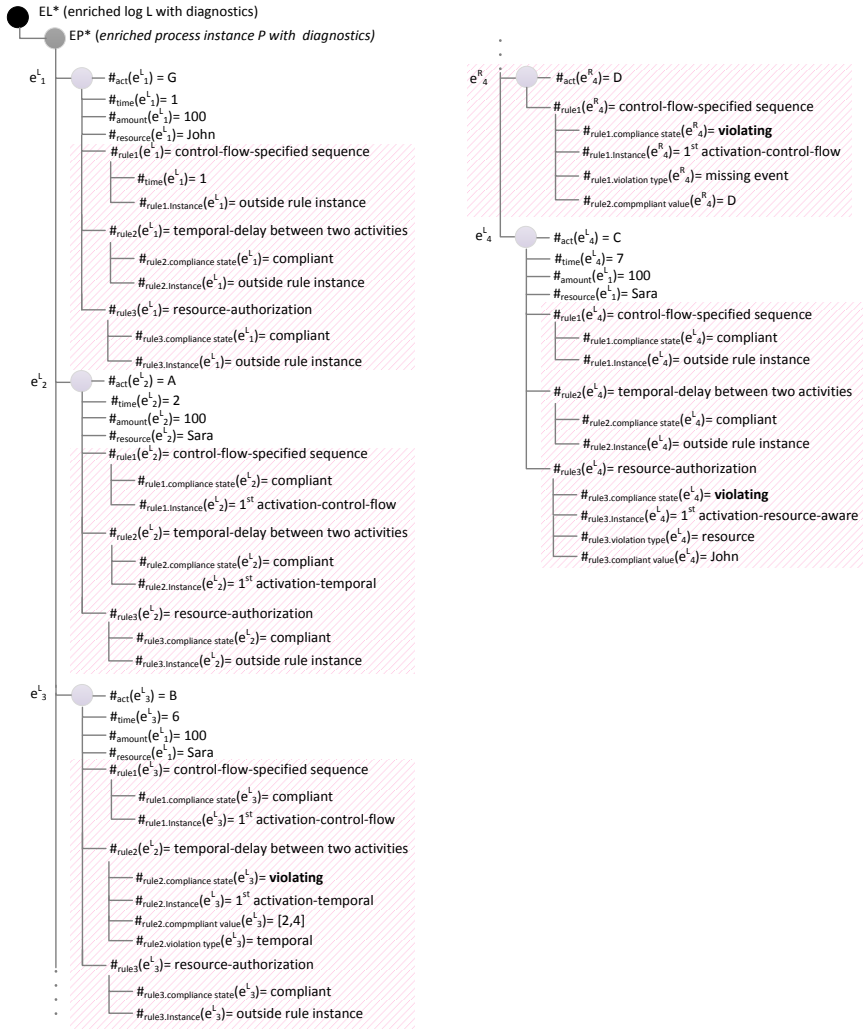


Figure 3.34: The event log  $L$  enriched with compliance diagnostic information on compliance state of each event w.r.t. different rules, rule instances, violation types, and compliant values.

Similarly, we can see that the third event  $e_3^L$  is checked against all the three compliance rules. This event is compliant with the control-flow and resource-aware compliance rules but it violates the temporal compliance rule. This violation took place in the first activation of the temporal rule “*temporal delay between two activities*”. Its violation is of the type *temporal violation*, and the violating value is 6 for the time attribute whereas it should have been a value in [2,4] interval. Note that the *compliant value* will always be taken from the process instance run at the corresponding event.

In this example we have a control-flow violation. This is indicated using the event  $e_4^R$ . Note that this event does not exist in the original log, but was inserted as an artificial event due to a model-only move in the alignment of Fig. 3.23 at the respective position. The violation type assigned to this event is ‘*missing event*’ and the compliant value is activity name *D* indicating that *D* should have occurred at this moment in the execution. Since this event is just checked for the control-flow compliance rule, we do not add diagnostics attributes for other rules in the example.

We see one more violation at the event  $e_4^L$ . As is indicated this event violated the resource-aware “**resource-authorization**”. The violation occurred within the first resource-aware rule instance and it has a violation of type *resource*. The compliant value for this violation returned by its corresponding alignment is ‘John’.

### Enriching an event log with results of checking a set of atomic compliance patterns vs. checking a composite compliance model.

Generating an enriched event log as the result of checking a log against a set of compliance rules that are formalized as atomic compliance patterns requires combining the results of several alignments together as is shown in Fig. 3.33. If we formalize a set of compliance rules as a composite compliance model, it is only required to project the result of one alignment (usually a data-aware alignment) on the original log. Figure 3.35 illustrates this difference. We will discuss this difference and its consequences further in Chap. 8.

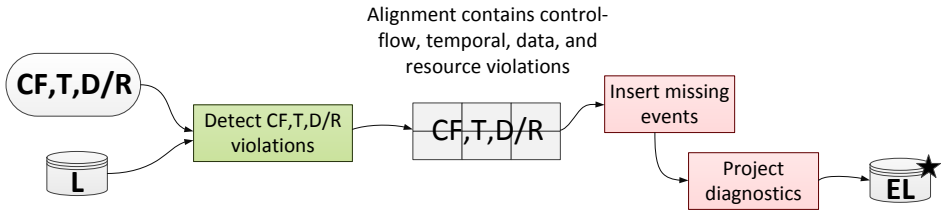


Figure 3.35: Generating enriched event log with diagnostics based on one overall alignment result.

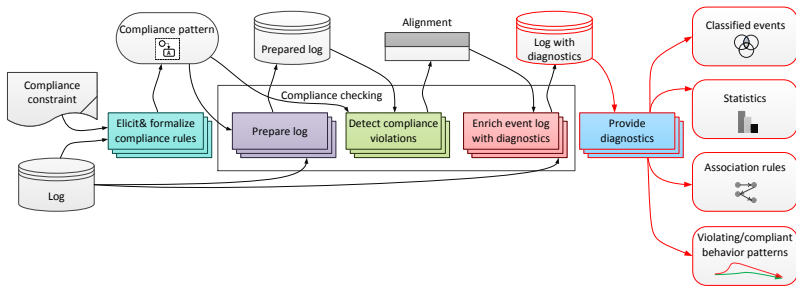


Figure 3.36: Compliance analysis overview.

### 3.6 Providing an Overview of Diagnostics and Root-Cause Analysis of Violations

In the previous section we discussed dedicated techniques for checking the compliance of a business process against different types of compliance rules. These techniques are based on computing optimal alignments between an event log and the “closest compliant run” according to a formal model of a single compliance rule or a set of compliance rules. Such an alignment reveals the details of what went wrong per process instance, per rule instance, and per activity and what should have happened instead. However, these techniques do not provide further diagnostics on root-cause(s) of a given violation nor a complete picture about the violations in the entire process. Moreover, the results produced by these techniques do not address the needs of people who are actually in charge of assessing and understanding compliance. Analyzing the root cause(s) of violations is usually done by experienced analysts having in depth knowledge of

the underlying business process. The findings are interpreted typically in an ad-hoc manner and without leveraging the transparency provided by data analysis techniques. Consequently this makes it difficult for business analysts to identify the causes of deviations when it is hidden in large amounts of data. Providing diagnostics about deviations in a clear and compact way not only helps having an up to date awareness of the compliance state of an organization, it also provides useful information to take operative decisions on how to deal with non-compliance and how to improve current operation.

Therefore, in our approach we have a dedicated step (Fig. 3.36 after checking compliance) for providing further diagnostics about the detected violations during compliance checking. The enriched event log obtained from the previous step is a collection of events that contain all the information about the process and its violations. We can provide various statistics about the violations in different abstraction levels. We visualize in a table for each attribute (Fig 3.37 top), in what kind of compliance violations it was involved and to which extent. The proposed color coding allows a user to quickly spot a particular (group of)

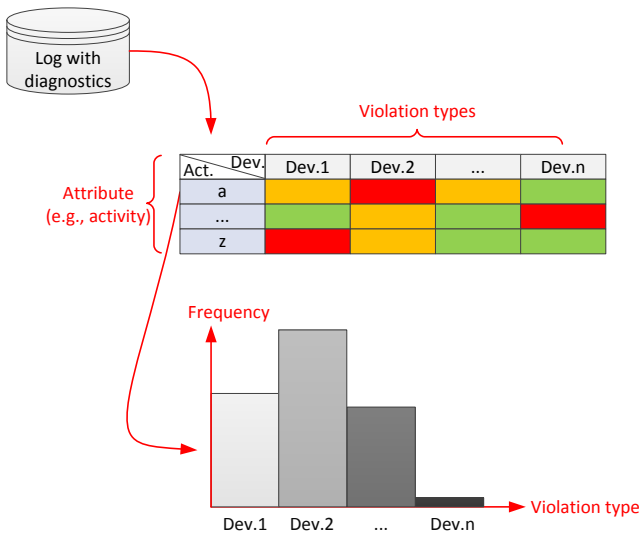


Figure 3.37: Statistics in different abstraction levels are built over enriched log with diagnostics including frequency of violations in total and per attribute (e.g. per activity).



activities or compliance rules involved in many violations. Selecting an individual activity then gives a detailed histogram for all violations of this attribute (Fig. 3.37 bottom).

Any process mining technique can be used to further investigate this log. For instance as shown in Fig. 3.38, we can filter the process instances in a log containing a specific violation and discover a process model that describes them best and compare it with the model that describes process instances without that specific violation.

Since we have diagnostics at the level of events and the context that these events occurred in (i.e., the event attributes and their values), we can also use other machine learning techniques to uncover causes of a violation. We can use association rule mining to detect possible meaningful correlations (exemplified in Fig. 3.39) between certain violations and the context they occurred in. These

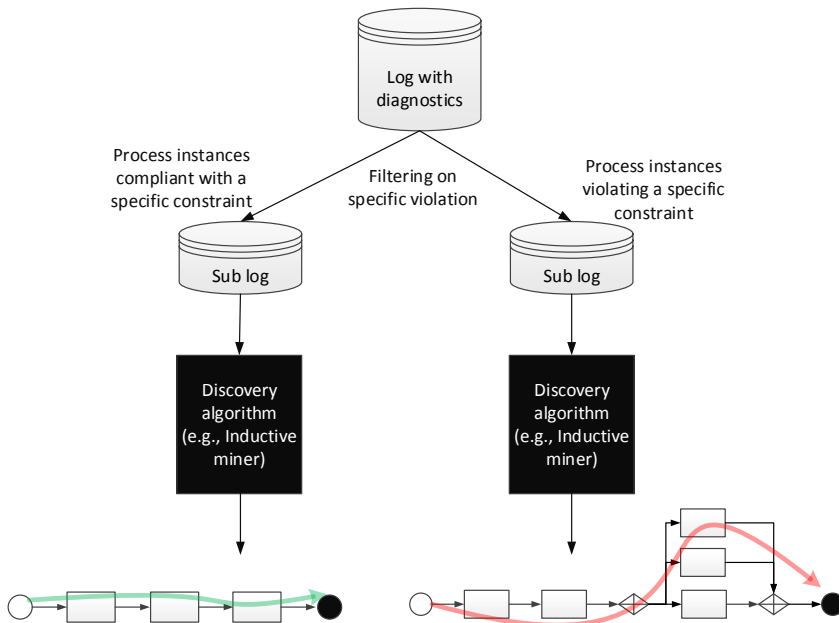


Figure 3.38: Discovering process models that describe the behavior of cases violating a specific constraint compared to those that are compliant w.r.t. that constraint.

correlations can help us to predict future violations.

Using classification techniques, we can compare compliant and violating events (exemplified in Fig. 3.40) w.r.t. their context and identify the conditions under which a certain violation holds or does not hold.

We have adapted a combination of these techniques for root-cause analysis of compliance violations in our compliance analysis approach. We combine these techniques and present them in such a way that end-users are not exposed to the underlying technicalities. These techniques and the presentation of the

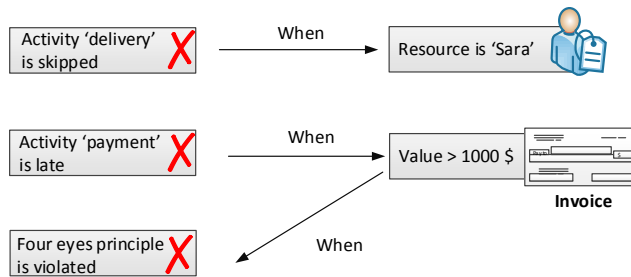


Figure 3.39: Correlations between violations and specific attribute values guide us in hypothesizing about the causes of violations.

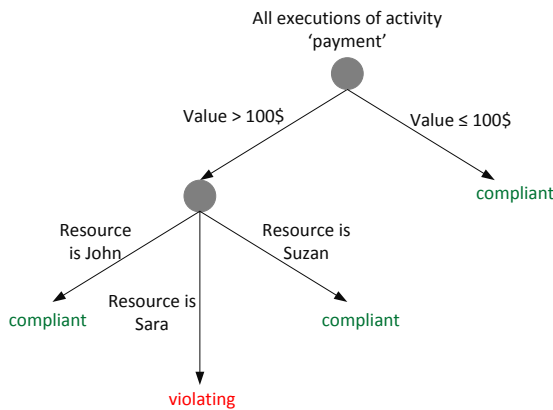


Figure 3.40: Classifying violations and non-violations based on their context help us predict future violations.

results will be discussed in detail in Chap. 7.

### **Overview of diagnostics and root-cause diagnostics using a set of atomic compliance patterns vs. a composite compliance model.**

The input for providing overview of diagnostics and root-cause analysis of violations is the *enriched event log* with diagnostics. Whether we get the *enriched event log* as the result of checking an event log against a set of atomic compliance patterns or a composite compliance model does not matter in providing diagnostics. We can apply our root-cause analysis techniques on enriched event logs obtained in either way.

## **3.7 Concluding Remarks**

In this chapter we discussed all the requirements for compliance analysis. Our goal is to understand non-compliance and provide diagnostics about violations. This goal imposes some requirements on how compliance violations are detected, and how compliance rules are formalized. We discussed our choice to use alignments as the checking technique that provides us with the type of diagnostics that we would like to obtain. We described the checking for different compliance rule types including control-flow, temporal, data-aware, and resource-aware.

We introduced two main approaches for formalizing compliance rules: (1) as a set of atomic compliance patterns and (2) as a composite compliance model. Chapters 4, 5, and 6 discuss how to check atomic compliance patterns and are also expanded to composite compliance models with the limitations on log preparations and diagnostics that can be obtained.

We sketched our ideas for combining different compliance checking results to generate an enriched event log with all process and diagnostic information. This enriched event log then can be used to uncover causes of violations. Generating an enriched event log with diagnostics using a set of compliance patterns will be discussed in Chap. 8. The root-cause analysis of violations obtained using a composite compliance model will be discussed in Chap. 7. The techniques developed and discussed in each chapter are implemented in the Process Mining Toolkit ProM 6.6 (available from <http://www.promtools.org/>). The implementation of the techniques relevant for each chapter is discussed in the same chapter. Cases to illustrate the applicability of our approach and its impact in improving business processes are described in Chap. 9.

# Chapter 4

## Control-Flow Compliance Checking

In Chap. 3, we showed how one can use alignments for compliance checking of control-flow compliance rules. In this thesis we chose to formalize compliance rules Petri nets to enable precise detection of compliance violations. This way we can build on established techniques and tools for Petri nets and alignments. In this chapter, we will discuss in detail how to build and use Petri net constructs for obtaining precise diagnostics on control-flow compliance violations. Figure 4.1 illustrates how the content of this chapter is organized in different sections. In Sect. 4.1, we discuss in detail the Petri net constructs and patterns needed to formalize particular compliance constraints, and how these constructs then yield particular kinds of diagnostic using alignments. We then turn to preparing event logs for control-flow compliance checking in Sect. 4.2. In Sect. 4.3, we discuss results of applying control-flow checking in practice. In Sect. 4.4, we then explore the problem of actually translating a compliance constraint given in natural language into a formal model and present a solution approach in the same section. In Sect. 4.5 we showcase this solution using a real-life example. In Sect. 4.6, we explain how to build composite compliance models. We discuss related work in Sect. 4.7 and conclude the main points of this chapter in Sect. 4.8.

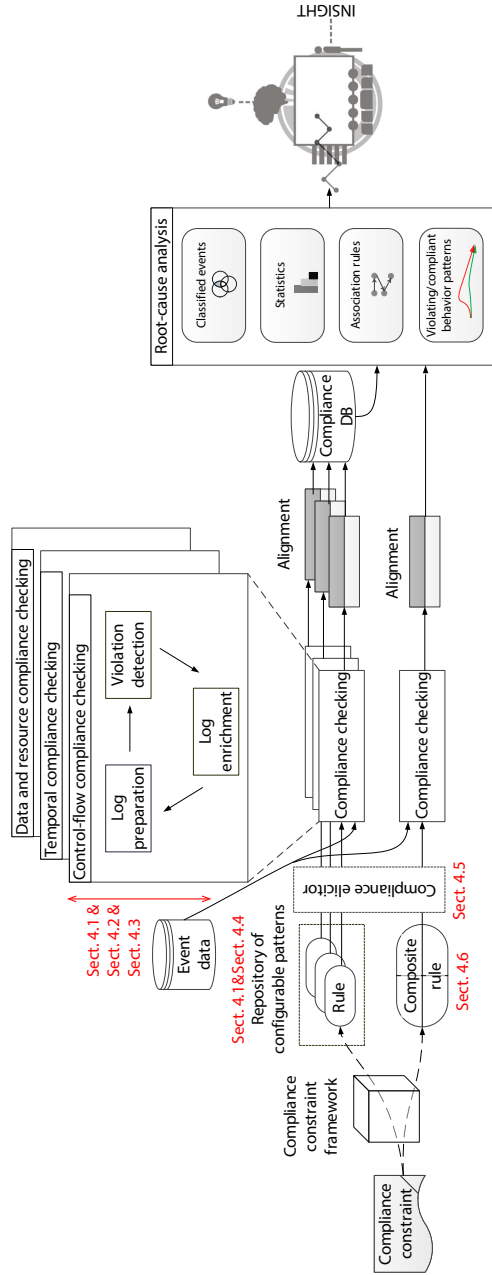


Figure 4.1: Thesis road map gives the mapping of the sections in Chap. 4 on to our compliance analysis approach.

## 4.1 Checking Control-Flow Compliance Rules

Category (Rules)	Description
Existence (2)	Limits occurrence or absence of an activity. [142], [14, 38], [55], [117], [118]
Bounded Existence (6)	Limits the number of times an activity must or must not occur. [38], [42]
Dependent Existence (6)	Limits the presence or absence of an activity with respect to existence or absence of another activity. [42]
Bounded Sequence (3)	Limits the number of times a sequence of activities must or must not occur. [38], [42]
Parallel (2)	Limits the occurrence of a specific set of activities in parallel. [117]
Precedence (10)	Limits the occurrence of an activity in precedence over another activity. [42], [38], [117], [118], [14], [49], [55]
Chain Precedence (4)	Limits the occurrence of a sequence of activities in precedence over another sequence of activities. [42], [38], [55]
Response (10)	Limits the occurrence of an activity in response to another activity. [117], [42], [55], [38], [119], [14], [49]
Chain Response (4)	Limits the occurrence of a sequence of activities in response to another sequence of activities. [42]
Between (7)	Limits the occurrence of an activity within (between) a sequence of activities. [38]

Table 4.1: Categorization of the 54 control-flow compliance rules.

Table 4.1 (also discussed in the previous chapter) provides an overview of 54 control-flow compliance rules. This collection is the result of an extensive literature study [14, 38, 42, 42, 49, 49, 55, 117–119, 142] including compliance constraints used in practise. These rules are distributed over ten categories. Each category includes several compliance rules. Categories are formed based on the type of constraints they include.

Control-flow compliance rules confine the existence, number of executions, and sequence of execution of activities in a business process. The specification of the rules should enable us to distinguish the scope of a rule and its activations throughout a process instance. In addition, the specification should precisely exclude compliant and violating behavior. These requirements give rise to a set of principles while formalizing a rule as an atomic Petri net that allow us to produce particular kind of diagnostics. We approached this prob-

lem by formalizing all compliance constraints we found in literature. Through formalizing these constraints, we developed a set of basic principles that help us to formalize rules as Petri nets. These are related to counting occurrences of activities, sequencing activities, excluding activities, and isolating different activations of a rule into clearly marked rule instances. Next we will present these basic principles by showing the formalization of a few selected compliance rules that use these principles. We will both focus on the formal modelling constructs and how these yield particular kinds of diagnostics. The complete collection of compliance rules and their formalization is provided in Appendix A.

#### 4.1.1 Example 1: Precedence

The precedence relation between two or more activities is a typical compliance constraint. Intuitively, it states that some activity  $B$  must have already occurred prior to an activity  $A$ , though  $B$  can also occur on its own. Rules in *Precedence*, *Between*, *Bounded Sequence*, and *Chain Precedence* categories are variants of this constraint. **Specifying a sequence of activities can be expressed easily in a Petri net by ordering transitions.** Yet, there are subtle aspects about each rule that should be considered in its formalization.

##### Atomic pattern

To explain the basic principles of formalization, we choose the rule stating: "Every time activity  $A$  is executed, it must be preceded directly by an occurrence of activity  $B$ ". If  $A$  occurs without a directly preceding  $B$ , this rule is violated. For instance the sequences of activities  $\langle B, C, C, B, B, C \rangle$  and  $\langle B, A, C, B, B, A \rangle$  comply to the rule, whereas the sequence of activities  $\langle B, A, B, C, A \rangle$  violates the rule. Note that the stated compliance rule is never activated for the sequence of activities  $\langle B, C, C, B, B, C \rangle$  because activity  $A$  has actually never happened.

The Petri net pattern shown in Fig. 4.2 formalizes this compliance rule. If  $B$  does not occur before  $A$ , the rule is violated. An instance of this compliance rule includes execution of activity  $A$  and its preceding  $B$ .

**The scope of this compliance rule is modelled with the *Start* and *End* of the pattern.** As discussed in Chap. 2, we focus on compliance rules constraining activities within a case. Thus, *Start* models the start of the case and *End* denotes the end of the case. A token in the *Final* place indicates the completion of a case. **The core behavior specified by the rule is modelled between  $I_{st}$  (denoting the activation of the rule) and  $I_{cmp}$  (denoting the completion of the rule).** This compliance rule does not limit the number of executions of  $A$  but it requires

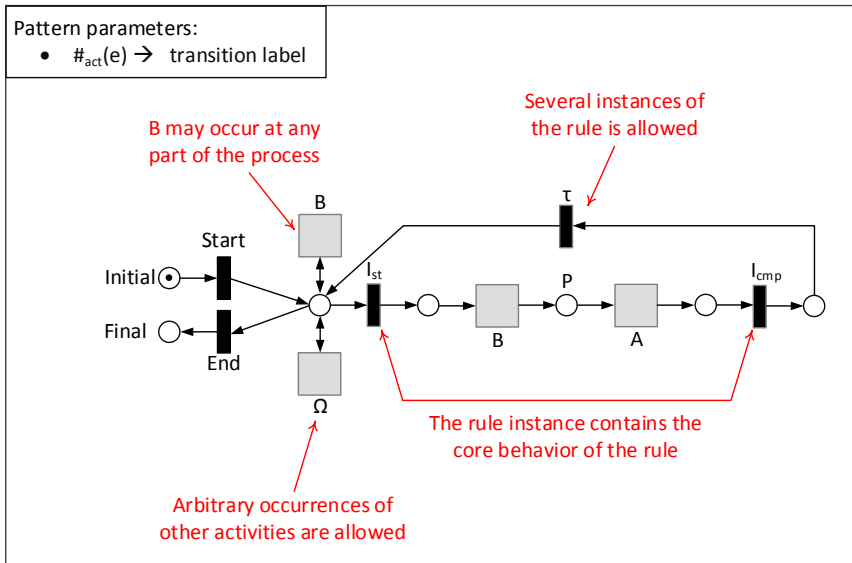


Figure 4.2: Atomic compliance pattern: Direct precedence.

every occurrence to be preceded by  $B$ . In essence the compliance rule can be activated multiple times within the scope of the rule. Hence, the pattern should allow for **multiple** occurrences of a  $B$  and  $A$ , so that  $A$  can only occur if  $B$  has directly preceded it. In the pattern, this is realized by a cycle looping from  $I_{cmp}$  back to  $I_{st}$  (through an invisible transition) allowing for another activation of the rule.

Transitions  $I_{st}$ ,  $I_{st}$ ,  $Start$ ,  $End$ , and  $\tau$  are considered invisible, i.e., they are used for modeling purposes only, therefore skipping them do not refer to a real violation.

As soon as a  $B$  occurs which is followed by  $A$ , the rule instance is triggered. In this situation the rule instance can complete and the pattern may **terminate** by firing the transition  $End$ .

The pattern also has to allow for **all possible compliant behaviors** where other activities than  $A$  and  $B$  occur, or no  $A$  occurs at all. This is achieved by adding the  $\Omega$ -labelled transition. In Fig. 4.2, **the  $\Omega$ -labelled transition models an arbitrary number of occurrences of other activities that are not specified in the rule.** As there is no  $\Omega$ -labelled transition adjacent to the place  $P$ ,  $B$  directly



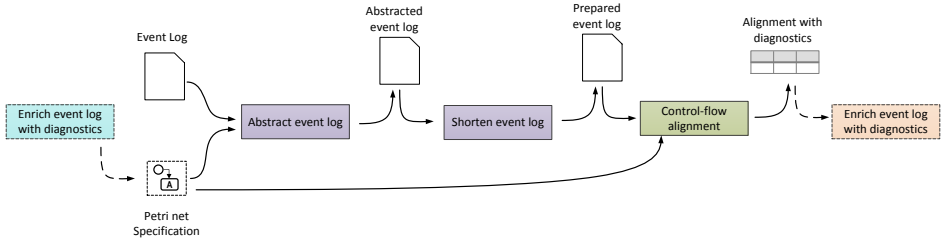


Figure 4.3: Overview of the control-flow compliance checking methodology.

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>	e <sub>9</sub>	e <sub>10</sub>
	time	1	2	3	4	5	6	7	8	9	10
	activity name	A	B	B	A	C	D	E	B	B	A

Figure 4.4: The example event log.

*precedes A* (i.e., occurrence of any other activity but A is excluded).

If there is no A or B just occurred, any activity may occur. Since the rule does not put any restriction on occurrence of activity B, this activity may occur an arbitrary number of times at any part of the process, also independent of a rule activation. We realize this in the model of Fig. 4.2, by the B-labelled transition outside the rule instance.

### Checking

Suppose we would like to check the event log presented in Fig. 4.4 against the *precedence* compliance rule formalized in Fig. 4.2. In the following we demonstrate step by step how we do the checking.

Figure 4.3 recalls our methodology to check control-flow compliance rules from Chap. 3. We prepare the event log in two steps and later detect control-flow violations using the alignment technique introduced in Chap. 2.

**Step 1: Abstracting the event log.** First we prepare the event log by generating the suitable checking attribute. As the activity name captures the occurrence of activities, we use the values of this attribute to determine which activity occurred in which position throughout a business process. Therefore, in this example we take the values for the checking attribute at each event from its

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>	e <sub>9</sub>	e <sub>10</sub>
	time	1	2	3	4	5	6	7	8	9	10
	activity name	A	B	B	A	C	D	E	B	B	A
	checking	A	B	B	A	Ω	Ω	Ω	B	B	A

Figure 4.5: Abstracted event log obtained from step 1.

activity name.

The abstracted event log obtained from the first step is shown in Fig. 4.5. The compliance rule specifies occurrences of activities *A* and *B*. We set  $\#_{checking}(e) : \#_{act}(e)$  for any event *e* where  $\#_{act}(e)$  is *A* or *B*, otherwise we set  $\#_{checking}(e) : \Omega$  to abstract from all events not relevant to the compliance rule. In Sect. 4.2 we give the full definition of generating checking attributes for control-flow compliance checking.

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub> ←	e <sub>8</sub> →	e <sub>9</sub>	e <sub>10</sub>
	time	1	2	3	4	5	8	9	10
	activity name	A	B	B	A	C	B	B	A
	checking	A	B	B	A	Ω	B	B	A

↑  
Ω events deleted but their position in the log is kept

Figure 4.6: Shortened event log obtained from step 2.

**Step 2: Shortening the event log.** The checking attribute will be used to map events in the log to the transitions in the model. As can be seen, the abstracted event log contains three occurrences of  $\Omega$  in a sequence. Since the rule only specifies the relation between activities *A* and *B*, we can simply substitute three occurrences of  $\Omega$  with one as this does not influence the compliance of the case or detection of violations. Yet, it will improve performance while checking because it reduces the size of the event log. In essence, we abstract long sequences of events that are not relevant to the compliance rule to shorter sequences. Note that the order and relative position of compliance-relevant events are preserved

while irrelevant details are abstracted from. The position and ordering of events are used during the projection of diagnostics on the original event log. We will elaborate on this further in Chap. 8.

Figure 4.6 shows the event log that is obtained from this step. There, it is indicated that two occurrences of  $\Omega$  events ( $e_6$ , and  $e_7$ ) are removed.

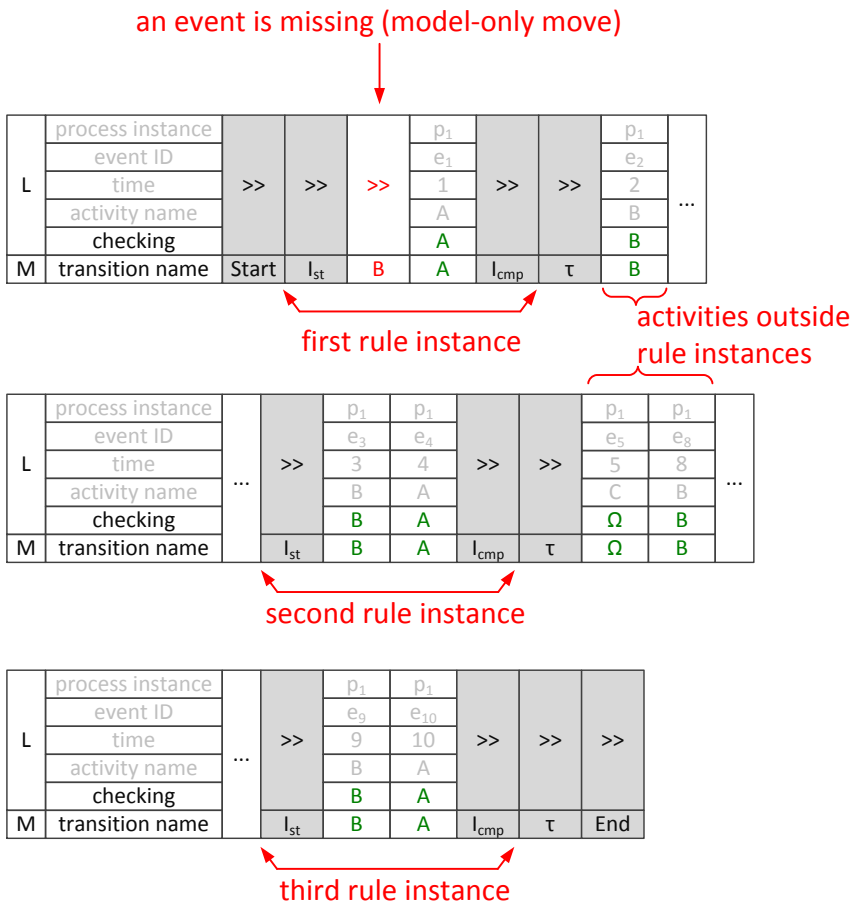


Figure 4.7: Control-flow alignment obtained from step 3.

**Step 3: Control-flow alignment.** We use the prepared event log (abstracted and shortened) and the atomic pattern of the rule (Fig. 4.2) to build a control-flow alignment. Figure 4.7 shows a control-flow alignment for this example.

The alignment indicates three activations of the rule (i.e., three rule instances). It also indicates that the violation was observed in the first rule instance; activity  $A$  was executed without a preceding  $B$ , i.e., according to the model, the execution of activity  $B$  is skipped (*model-only move*). This is represented by  $\gg$  value for the log event in the third move. The second and the third activations of the rule instance are compliant. Note how the definition of rule instance in the Petri net pattern enables us to locate the violation precisely. In addition we can determine activities that fall inside the rule instance and those that fall outside of it (for example two occurrences of  $B$  ( $e_2$ , and  $e_8$ ). Finally, several model moves in the alignment are due to the invisible transitions in the model ( $Start$ ,  $I_{st}$ ,  $I_{cmp}$ , and  $End$ ). These model-only moves do not refer to any real violation.

#### 4.1.2 Parameters for Computing Alignments to Influence Diagnostic Information

As we discussed in Chap. 2, infinitely many alignments are possible for a log and a model. We are interested in alignments which maximize the number of “perfect matches” between log events and model events. Therefore, we typically define a notion of optimal alignment using a cost function for moves where model and log deviate including *log-only moves*, *model-only moves*, and *incorrect synchronous moves*. We assign a cost-function  $k$  to each move ( $e^L, e^R$ ). A compliant move has cost 0 and all other types of moves have a cost  $> 0$ . The choice of costs depends on the particular domain and can be set for instance based on how likely a particular violation is known to happen. By giving frequent violations fewer costs than infrequent violations, the oracle function  $\alpha$  (see Chap. 2) returns the alignment giving the “closest compliant run” as the best alignment. In case there are several “best alignments” (i.e., alignments with the same cost), the oracle picks one of these optimal alignments in a deterministic manner.

The oracle function solves a conformance checking problem. Given a log  $L$  and a model  $Mod$ , the oracle returns a run  $R \in Mod$  and constructs the alignment  $A = (L, R, M)$  with the lowest cost. The  $A^*$ -based search on the space of (all prefixes) of all alignments of a log  $L$  to a model  $Mod$  described in [10, 135] can be used to find the best alignment considering violations of type *log-only move*, and *model-only move*. This approach is extended in [33, 134] to find violations

of type *incorrect synchronous move*; an ILP solver finds among all synchronous moves values for attributes of  $R$  such that the total cost of an alignment related to violations of type *incorrect synchronous move* are minimized.

Tuning the cost function influences the quality of diagnostics. Assume for the previous example we assign the cost 0 to *compliant moves*, cost 1 to violations of type *log-only moves* and cost 10 to violations of type *model-only moves*. In this case, the technique would return a different alignment than what is shown in Fig. 4.7 as the “optimal alignment”. That is, our alignment technique would consider the first occurrence of activity  $A$  as a *log-only move*. This way, two compliant rule instances will be detected and the occurrence of  $A$  will be captured as a violation outside a rule instance. How to select the appropriate cost function is a problem that is addressed in [134, 135]. We use the same setting for cost function in the control-flow compliance checking.

### 4.1.3 Example 2: Bounded Existence of an Activity

In Sect. 4.1.1, we explained the basic steps of formalizing and checking control-flow compliance rules that constrain the *sequencing* of activities on the example of the *Precedence* rule. In the following, we consider how to constrain the number of occurrences of an activity.

The *Bounded Existence* and *Bounded Sequence* categories of compliance rules, include rules that limit the repetition of activities or sequence of activities within a chosen scope. To **bound the repetition of a structure (either activities or sequence of activities), we usually limit the number of times that the structure is enabled or insert a counter that counts the number of times that a certain structure has been executed**. We will illustrate compliance checking for such rules by the bounded existence rule “Activity  $A$  must occur  $k$  times”. If  $A$  is executed less than or more than  $k$  times, the compliance rule is violated. For instance, for  $k = 2$  the occurrence of the sequence  $\langle B, C, A, D, B, C, A, D \rangle$  complies to this rule and occurrence of the sequence  $\langle B, C, A, D, B, C, A, A, D \rangle$  or the sequence  $\langle B, C, D, B, C, D \rangle$  violate the rule.

#### Formalizing as an atomic pattern.

Figure 4.8 shows the atomic pattern that formalizes the bounded existence rule. Activity  $A$  is described by the *A-labelled* transition. As in Sect. 4.1.1, the pattern abstracts from all other events in the process that are not specified by the compliance rule through the  $\Omega$ -labelled transition.

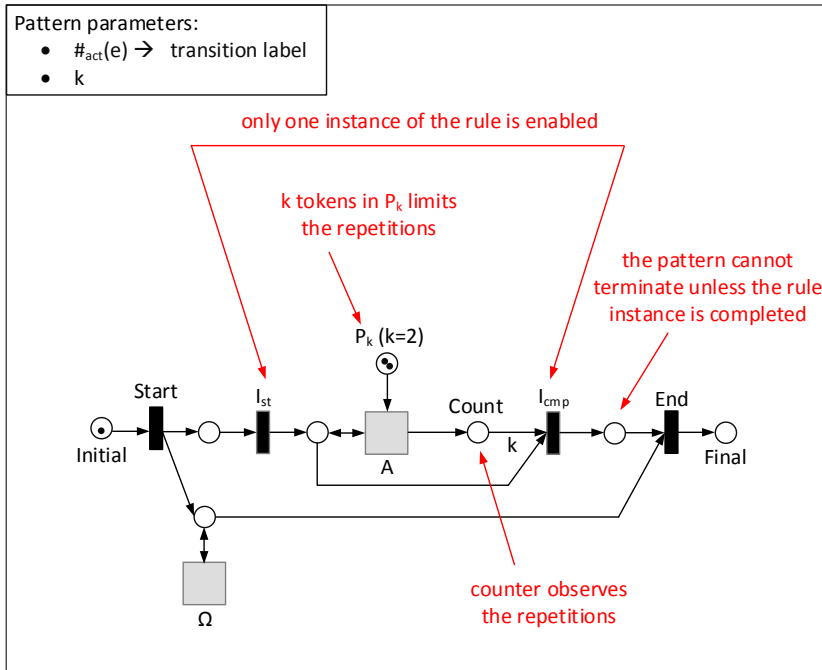


Figure 4.8: Atomic compliance pattern: Bounded existence of an activity exactly  $k$  times.

An instance of this compliance rule includes all occurrences of activity  $A$  within a case - in a compliant case  $A$  occurs exactly  $k$  times. This is realized in the model of Fig. 4.8 by allowing the rule instance (between  $I_{st}$  and  $I_{cmp}$ ) to be activated only once during a case (no loop back from  $I_{cmp}$  to  $I_{st}$ ). In addition, the pattern is designed such that the rule instance must be activated otherwise the pattern cannot terminate. After the pattern started any activity may be executed. The first occurrence of  $A$  triggers the *rule instance*. As soon as activity  $A$  occurs  $k$  times, this rule is satisfied, i.e., the condition of this rule is complete.

The pre-place  $P_k$  of  $A$  is initially marked with  $k$  tokens. The  $k$  tokens in place  $P_k$  assure that activity  $A$  can occur at most  $k$  times, as each occurrence of  $A$  decrements the number of tokens in  $P_k$  and increments the number of tokens in place  $Count$ . Place  $Count$  counts the occurrences of  $A$ . After  $k$  occurrences

L	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$e_8$	$e_9$	$e_{10}$
	time	1	2	3	4	5	6	7	8	9	10
	activity name	A	B	B	A	C	D	E	B	B	A
	checking	A	$\Omega$	$\Omega$	A	$\Omega$	$\Omega$	$\Omega$	$\Omega$	$\Omega$	A

Figure 4.9: Abstracted event log obtained from step 1.

of  $A$ ,  $P_k$  is empty and  $Count$  contains  $k$  tokens. In this situation transition  $A$  is not enabled anymore. The compliance rule is satisfied and the pattern instance is completed ( $I_{cmp}$ ). The pattern can terminate only if the pattern instance is completed implying that the condition of the rule is satisfied. The transition  $End$  models that the end of the process instance has been reached. Please note that  $\Omega$ -labeled transition is enabled throughout the whole pattern and may occur at any point in time.

### Checking

We check also for this example whether the event log (shown in Fig. 4.4) adheres to the rule (formalized as the pattern shown in Fig. 4.8). Similar to the previous example we go through the three steps shown in Fig. 4.3 to check this rule.

**Step 1: Abstracting the event log.** The compliance rule specifies occurrences of activity  $A$ , therefore wherever an event has value  $A$  for its activity name we copy the same value as the value for its checking attribute. Otherwise we record the generic value  $\Omega$  as value for the checking attribute. The abstracted event log obtained from the first step is shown in Fig. 4.9.

**Step 2: Shortening the event log.** Similar to the previous example, we shorten the event log to remove events that are not required for checking. Figure 4.10 illustrates the event log that is obtained from this step.

Note that the position of the removed  $\Omega$  events is kept (between  $e_2$  and  $e_4$ , and between  $e_5$  and  $e_{10}$ ). We need this information when we enrich the original event log with the obtained diagnostics. This will be elaborated in Chap. 8.

**Step 3: Control-flow alignment.** We align the shortened event log with the Petri net pattern of this rule (Fig. 4.8). The resulting alignment is shown in Fig. 4.11.

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub> ↔ e <sub>4</sub>	e <sub>5</sub> ↔ e <sub>10</sub>		
	time	1	2	4	5	10
	activity name	A	B	A	C	A
	checking	A	Ω	A	Ω	A

Ω events deleted but their position in the log is kept

Figure 4.10: Abstracted event log obtained from step 2.

L	process instance			p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>		p <sub>1</sub>	p <sub>1</sub>
	event ID			e <sub>1</sub>	e <sub>2</sub>	e <sub>4</sub>		e <sub>5</sub>	e <sub>10</sub>
	time	>>	>>	1	2	4	>>	5	10
	activity name			A	B	A		C	A
	checking			A	Ω	A		Ω	A
M	transition name	Start	l <sub>st</sub>	A	Ω	A	l <sub>cmp</sub>	Ω	>>

event is not allowed (log-only move)

Figure 4.11: Control-flow alignment obtained from step 3.

The alignment shows one activation of the compliance rule instance. The first two occurrences of A are compliant but the third A cannot be executed according to the model. This is denoted (in red) as a *log-only move* in the alignment.

The obtained alignment locates a violation in the event log and it also indicates what should change to make the event log compliant. Later during the log enrichment we project these diagnostics on the events of the original event log.

### 4.1.4 Example 3: Simultaneous Occurrence of Two Activities

So far we presented rules and their respective Petri net patterns assuming that occurrences of activities are captured as atomic events in event logs. However,



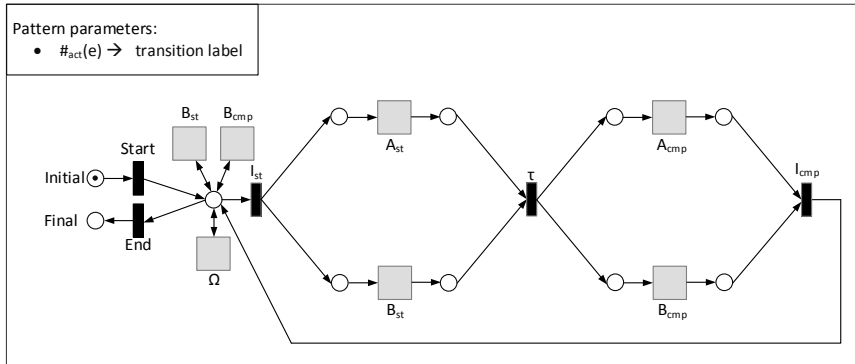


Figure 4.12: Atomic compliance pattern: Simultaneous occurrence of two activities.

some activities may be captured by more than one event. For instance the execution of an activity may be captured by two events denoting the *start* and *completion* of execution of the activity. All the Petri net patterns that are listed in Table 4.1, based on the setting, can be formalized in these different flavours. The next example demonstrates a compliance rule in this way. This example shows how intricate ordering constraints can be formalized using Petri nets in an intuitive way.

The *parallel* category of compliance rules include compliance rules that limit the occurrence of an activity in parallel with or during another activity. In case of ongoing activities, they may overlap in time. Some compliance constraints may require simultaneous occurrences of activities. For example the administration of a specific medicine to a patient during a surgery. In this section we show, by example, how we model and check such a compliance rule.

We pick the *simultaneous* compliance rule from the *parallel* category stating: “Activity *A* must always occur simultaneously with activity *B*”. In this case we need to **capture and model the start and completion of activities separately**. Therefore, we represent start and completion of activity *A* with two different transitions. Similarly activity *B* should be presented.

### Atomic pattern

The Petri net pattern shown in Fig. 4.12 describes this rule. If *A* starts *B* should also start, i.e., activity *A* cannot complete unless *B* has already started. Similarly

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	...
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>	
	life cycle	sched	st	st	cmp	cmp	sched	st	st	
	time	1	2	3	4	5	6	7	8	
	activity name	A	A	B	B	A	A	B	A	

L	process instance	...	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID		e <sub>9</sub>	e <sub>10</sub>	e <sub>11</sub>	e <sub>12</sub>	e <sub>13</sub>	e <sub>14</sub>	e <sub>15</sub>	e <sub>16</sub>
	life cycle		cmp	cmp	sched	cmp	st	cmp	st	cmp
	time		9	10	11	12	13	14	15	16
	activity name		B	A	A	D	A	A	B	B

Figure 4.13: An event log where activities have life-cycle attributes.

if start of the activity *B* activates the pattern instance, it is required that activity *A* also starts. The pattern enforces - by construction - that the activities *A* and *B* must complete together otherwise the pattern instance cannot complete, hence the pattern may not terminate. Please note that  $\Omega$ -labeled activities and *B* may occur outside the rule instance and throughout the pattern. The pattern allows for multiple activations of the rule. The transition *End* models that the end of the case has been reached and the pattern may terminate.

**Checking**

Figure 4.13 exemplifies an event log recorded in a setting where execution of activities are captured with more than one event. As can be seen, events in this log have an extra attribute *life cycle* which denotes the start, completion or in general status of activities. For instance we observe that occurrences of activity *A* iare presented with three different events *scheduled* (*sched*), *start* (*st*), and *complete* (*cmp*) and activity *B* is presented with two events *start* (*st*), and *complete* (*cmp*). We would like to check this log against the rule (shown in Fig. 4.12).

**Step 1: Abstracting the eventing log.** Similar to previous examples, the first step in log preparation includes the choice of the checking attribute and abstraction of the event log accordingly. When the execution of activities are captured by several events, merely the values of attribute *activity name* do not suffice to distinguish whether an event is referring to start or completion of that activity. Therefore, we combine values of *activity name*, and *activity life cycle* for the

L	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	...
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>	
	life cycle	sched	st	st	cmp	cmp	sched	st	st	
	time	1	2	3	4	5	6	7	8	
	activity name	A	A	B	B	A	A	B	A	
	checking	Ω	A <sub>st</sub>	B <sub>st</sub>	B <sub>cmp</sub>	A <sub>cmp</sub>	Ω	B <sub>st</sub>	A <sub>st</sub>	

↑  
 values of activity name and life cycle attributes  
 are combined for the *checking* attribute

L	process instance	...	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID		e <sub>9</sub>	e <sub>10</sub>	e <sub>11</sub>	e <sub>12</sub>	e <sub>13</sub>	e <sub>14</sub>	e <sub>15</sub>	e <sub>16</sub>
	life cycle		cmp	cmp	sched	cmp	st	cmp	st	cmp
	time		9	10	11	12	13	14	15	16
	activity name		B	A	A	D	A	A	B	B
	checking		B <sub>cmp</sub>	A <sub>cmp</sub>	Ω	Ω	A <sub>st</sub>	A <sub>cmp</sub>	B <sub>st</sub>	B <sub>cmp</sub>

Figure 4.14: Abstracted event log obtained from step 1. Values of *activity name* and *life cycle* attributes are combined for the *checking* attribute.

values of the checking attribute.

Figure 4.14 illustrates the abstracted event log obtained from this step. Note that the *checking attribute* will get the value  $A_{st}$  when *activity name*:  $A$ , and *activity life cycle*: *start* ( $st$ ), and it gets the value  $A_{cmp}$  when *activity name*:  $A$ , and *activity life cycle*: *complete* ( $cmp$ ). Similarly values of  $B_{st}$ , and  $B_{cmp}$  for start and completion of activity  $B$  is generated. The remaining events get the value  $\Omega$  including the events related to activity  $A$  when *life cycle*: *scheduled* ( $sched$ ). Although this event is related to activity  $A$ , it is irrelevant for checking of the rule, hence we will abstract from it.

**Step 2: Shorten the event log.** This step can be done as shown in the previous example. The event log obtained from this step is shown in Fig. 4.15.

<b>L</b>	process instance	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>
	life cycle	sched	st	st	cmp	cmp	sched	st	st
	time	1	2	3	4	5	6	7	8
	activity name	A	A	B	B	A	A	B	A
	checking	Ω	A <sub>st</sub>	B <sub>st</sub>	B <sub>cmp</sub>	A <sub>cmp</sub>	Ω	B <sub>st</sub>	A <sub>st</sub>

<b>L</b>	process instance	...	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>
	event ID		e <sub>9</sub>	e <sub>10</sub>	e <sub>11</sub>	e <sub>13</sub>	e <sub>14</sub>	e <sub>15</sub>	e <sub>16</sub>
	life cycle		cmp	cmp	sched	st	cmp	st	cmp
	time		9	10	11	13	14	15	16
	activity name		B	A	A	A	A	B	B
	checking		B <sub>st</sub>	A <sub>cmp</sub>	Ω	A <sub>st</sub>	A <sub>cmp</sub>	B <sub>st</sub>	B <sub>st</sub>

↑  
e<sub>12</sub> is deleted during the shortening of the event log.

Figure 4.15: Shortened event log obtained from step 2. The event e<sub>12</sub> is deleted during the shortening of the event log.

**Step 3: Control-flow alignment.** Figure 4.16 illustrates an alinement of the log to the Petri net pattern of this rule. The alignment indicates three activations of the compliance rule (i.e. three instances of the rule). Two of these instances are compliant (the first and the second one) and the third instance is violating. It is indicated in the third instance of the compliance rule that activity A is completed before activity B starts. Therefore, we observe first a violation of type *log-only move* and later a violation of type *model-only move* (indicating that A<sub>cmp</sub> occurred in a wrong position).

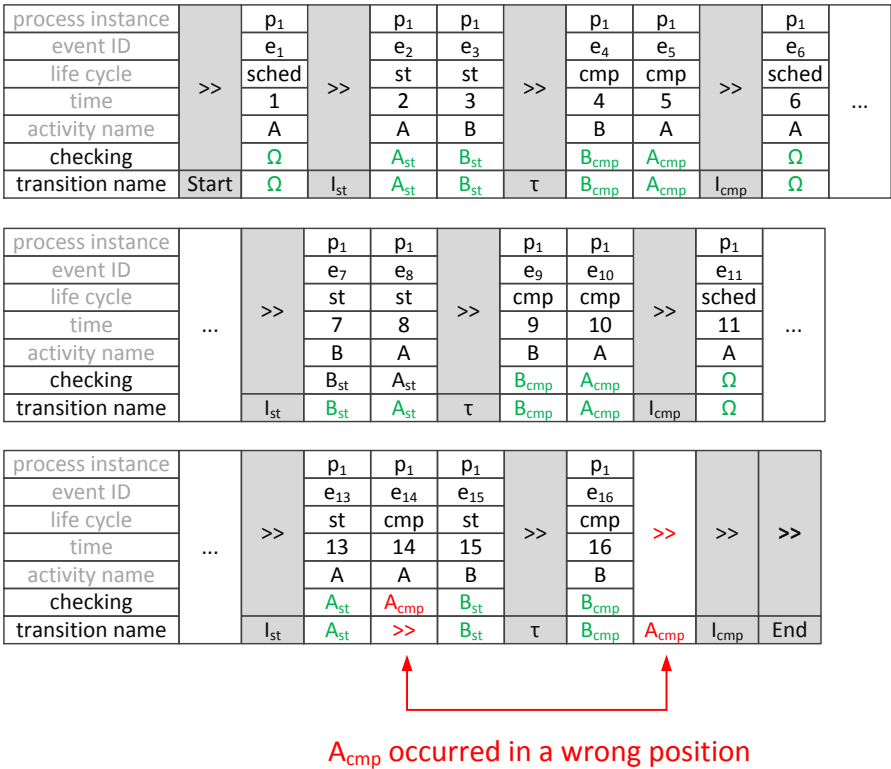


Figure 4.16: Control-flow alignment obtained from step 3.

### 4.1.5 Principles of Designing Atomic Compliance Patterns

The compliance patterns we discussed previously merely serve as a few simple examples from our much larger collection of control-flow compliance rules. Using the Petri net structures we discussed earlier, we can model many more intricate compliance rules. The complete collection of these rules is presented in Appendix A. Regardless of the underlying compliance rule, all the atomic patterns in our collection follow some systematics and basic principles that were partially discussed for the showcased examples. The complete explanation is available in Appendix A. In the following we briefly list some of the principles in designing an atomic compliance pattern.

- Each atomic compliance pattern has a dedicated place *Initial* and a place *Final*. See for example Fig. 4.2.
- A token in the final place defines the final marking of the pattern. When a pattern reaches its final marking, the pattern is properly completed (i.e., all other places of the net is empty).
- Every compliance pattern models a *rule instance*. The rule instance starts as soon as an event occurs which triggers the underlying condition of the rule. The *rule instance* completes as soon as its condition is satisfied. See for example Fig. 4.2.
- The  $I_{st}$ -labeled transition in every atomic pattern indicates the start of a (rule instance) and the  $I_{cmp}$ -labeled indicates the completion of an rule instance. See for example Fig. 4.2.
- A process instance (case) in a log *complies* to a (rule of a) pattern if after executing that case, final transition *End* is enabled, and its occurrence leads to the *final marking*.
- *Start-labeled*, *End-labeled*,  $I_{st}$ -labeled,  $I_{cmp}$ -labeled, and  $\tau$ -labeled transitions are silent transitions. In finding an optimal alignment between a log and a compliance pattern, the ‘alignment based technique’ of Chap. 2, assigns the cost of zero for deviation of these transitions.
- $\Sigma$  denotes the set of transition names for each atomic pattern. Depending on the choice of compliance rules, it may include elements describing start and completion of activities. See for example Fig. A.32.

- Occurrences of event(s) specified in the compliance rule are mimicked by transitions in the pattern having the same label as the values for the checking attribute.
- Occurrences of any other events than the event(s) specified in the compliance rule are mimicked by the  $\Omega$ -labeled transition. This way, the patterns abstract from all other log events that are not described in the compliance rule. Suppose a compliance rule restricts the occurrences of three specific events  $A$ ,  $B$ , and  $C$ ; hence  $\Sigma = \Omega \cup \{A, B, C\}$  and  $\Omega \cap \{A, B, C\} = \emptyset$ .
- In some patterns we need to exclude the occurrence of one specific event from the events that may occur in a marking, therefore we subtract that specific element from  $\Sigma$ . Suppose we need to exclude occurrence of a compliance-relevant activity  $A \in \Sigma \setminus \Omega$  at a marking; this is shown as  $\Sigma \setminus \{A\}$ , specifying that any event but  $A$  may occur at that marking. See for example Fig. A.32.
- Typically an occurrence of an activity  $A$  is represented as an atomic event  $A$  in the log. In some patterns, the respective compliance rules require to model the start and completion of activities in the Petri-net patterns. Any activity, e.g. an activity  $A$  can also be represented by two events  $A_{start}$  ( $A_{st}$ ) and  $A_{complete}$  ( $A_{cmp}$ ) indicating the start and completion of an ongoing activity. Therefore, all atomic patterns of our collection rules come in these two flavors and can be picked based on the setting. See for example Fig. 4.12.
- Arcs in patterns may have weight, the arc weight specifies how many tokens are consumed or produced as a result of firing a transition. If the arc weight is greater than one, the respective arc is annotated with the weight (i.e., a natural number); otherwise, it is assumed to be one.
- In some patterns a *reset arc* connects a place with a transition. Usually we use *reset arcs* to limit the number of executions of an activity to an upper boundary. This arc ensures that when the corresponding transition fires all tokens are consumed from the respective place (even if it contains no token). We usually use *reset arcs* to consume all tokens from the net, thereby guaranteeing that after firing *End*, no transition is enabled anymore and the net is empty. See for example Fig. A.11
- In some patterns we connect a place to a transition with an *inhibitor arc*. Usually we use *inhibitor arcs* to limit the executions of an activity with

a lower boundary. An *inhibitor arc* ensures that corresponding transition can only fire if the place at the other end of the *inhibitor arc* is empty. See for example Fig. A.13

- Every compliance pattern must allow for all possible compliant behaviors and exclude only what is specified as non-compliant by the underlying rule.

## 4.2 Generating the Checking Attribute and Its Values

As we described the checking procedure for all the previous examples, the first step in preparation of an event log comprises the generation of the checking attribute and its values. We described earlier how this task is done in various situations. Here, we would like to describe the preparation step in a more generic way based on the definitions we gave in Chap. 2.

Given *Attr* as the set of all possible attributes, *Val* as the set of all possible attribute values, and  $\mathcal{E}$  as the set of all events:

- The user picks the single attribute  $checking \in Attr$ .
- The user picks a set of attributes  $a_1, \dots, a_k \in Attr$  with  $k \geq 1$  from which the value of the *checking* attribute is generated.
- $V_{a_i}(\mathcal{E}) = \{\#_{a_i}(e) | e \in \mathcal{E}\}$ .
- For each attribute  $a_i$  with  $1 \leq i \leq k$ , the user picks a set of values  $V_i$  such that  $V_i \subseteq Val$  and  $V_i \subseteq V_{a_i}(E)$ .
- if  $(\#_{a_1}(e), \#_{a_2}(e), \dots, \#_{a_k}(e)) \in V_1 \times \dots \times V_k$ , then
  - $\#_{checking}(e) = \#_{a_1}(e) + \dots + \#_{a_k}(e)$ ,
  - else  $\#_{checking}(e) = \Omega$ .

The values of the *checking* attribute can be mapped to transition names in the atomic pattern for the checking. For example in Sect. 4.1.4:

- The user picks the set of attributes  $\{activity\ name, life\ cycle\}$ , and
- the set of values  $\{A, B, st, cmp\}$ .



- If  $\#activity\ name(e), \#life\ cycle(e) \in \{A, B, st, cmp\}$ , then
  - $\#checking(e) \in \{A_{st}, A_{cmp}, B_{st}, B_{cmp}\}$
  - else  $\#checking(e) = \Omega$

values for the checking attribute was generated from concatenating values of attribute *activity name* and *life cycle*. The user picks the attributes to

### 4.3 Applying Control-Flow Compliance Checking Using Real-Life Event Logs

We have evaluated our control-flow compliance checking technique on a real-life log taken from the financial system of a large Dutch hospital. The log contained over 150000 events from over 700 different activities in 1150 cases. Each case representing a patient. The log was obtained from financial system of the hospital in the period of 2005 to 2008. Beside anonymizing the log, all other data in the log is preserved including activity names, involved resources, and etc. We first describe the implementation used in this evaluation and then report on some relevant compliance rules and the results we obtained for them.

#### 4.3.1 Implementation in ProM

The presented technique was implemented as part of the *Compliance* package of the Process Mining Toolkit ProM 6.6. The **Check Compliance Using Conformance Checking** plugin takes a log and a rule formalized in terms of a Petri net pattern as input. For each rule to check, the user then configures its parameters, mostly by mapping events to transition names of the rule. The log preparation is done at the back end accordingly. Then the alignment technique (Chap. 2) is called to align the log to the rule's Petri net pattern. The resulting alignment is shown to the user. The alignment for each process instance is shown in a separate row and deviations are highlighted. A *log-only move* indicates that an event occurred which did not comply to the rule. A *model-only move* indicates that an expected event did not appear in log and therefore the log does not comply to the rule. Several figures in the next section show these diagnostics. Figures 4.17 and 4.18 show some functionalities of the **Check Compliance Using Conformance Checking** plugin.



Figure 4.17: Mapping transition in the compliance pattern to events of the log.

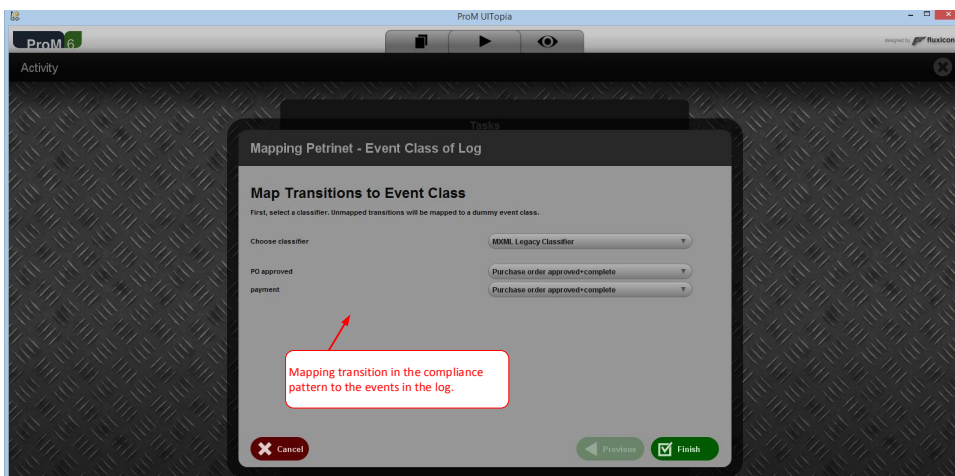


Figure 4.18: Mapping transition in the compliance pattern to events of the log.

### 4.3.2 Case Study Constraints and Results

In the case study we followed the standard use case for compliance checking (See compliance life cycle in Fig. 1.3): (1) check relevant regulations and elicit respective compliance constraints, (2) for each constraint, identify the patterns that precisely express the constraint from the rule collection in Table 4.1, (3) take the corresponding atomic pattern from our pattern collection in Appendix A and map its transitions to the events in the given log, and (4) run the compliance checker.

In the following, we describe our findings for three compliance constraints that were derived from the financial department's internal policies and medical guidelines.

**Compliance Constraint 1.** “The hospital should register each visiting patient and prevent duplicate registrations for a patient.” This constraint is formalized by the compliance rule “Event *patient registration* should occur exactly once per case.” from the category ‘*Bounded Existence*’ of Table 4.1. The pattern shown in Fig. 4.19 formalizes this rule (the pattern was discussed earlier in this chapter, see Sect. 4.1.1). To obtain a reliable result, we needed to filter the data for patients who started their treatment between 2005 and 2008, i.e., to make sure we do not make any conclusion on partial cases that started sometime before the analysis period. We checked compliance for 640 of 1150 cases and identified

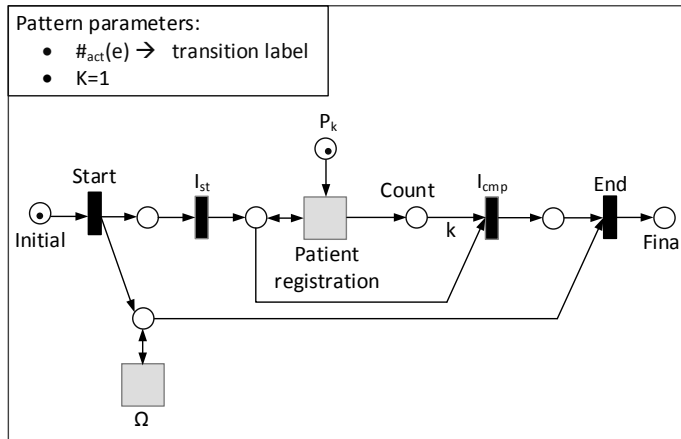


Figure 4.19: Activity X-ray must be executed at least once per case.

622 compliant and 18 non-compliant cases.

Figure 4.20 shows diagnostic information for a non-compliant case. As described above, ProM maps the log on to the compliance-influencing events: *patient registration* occurs twice, where the first occurrence is considered as compliant (highlighted green) and the second one should not have been in the case (highlighted yellow, *move on log*).

**Compliance Constraint 2.** The patients are sent to the hospital for specialized treatment. Therefore a basic X-ray scan has to be performed after a patient was registered. This constraint is formalized by two rules: “The event ‘*patient registration*’ should be followed by the event *X-ray*.”

The atomic pattern formalizing this rule is shown in Fig. 4.21. The rule specifies that activity *patient registration* must be followed eventually by *X-ray*. More executions of activity *X-ray* is also allowed.

Figure 4.22 shows an example of a non-compliant case w.r.t. this rule. It is indicated that activity *X-ray* was executed in a wrong position before the execution of activity *first visit registration* (first violation), while it was expected to occur afterwards (this is denoted by the second violation). In total we found 104 out of 640 cases compliant with this rule.

**Compliance Constraint 3.** “For safety reasons, either a *CT-Scan* or a *MRI-test* of an organ should be taken from a patient but not both.” The corresponding compliance rule from the *Exclusive* category has the atomic pattern of Fig. 4.23.

We checked this rule and identified 1092 compliant cases out of 1150. Fig. 4.24 shows diagnostic information for one non-compliant case. The relevant sequence of events for this case is  $\langle \dots \text{MRI-test}, \text{CT-scan} \dots \rangle$ . The occurrence of

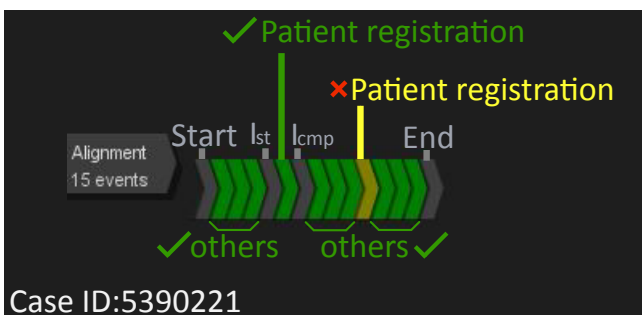


Figure 4.20: A non-compliant case: activity *patient registration* should have occurred exactly once.

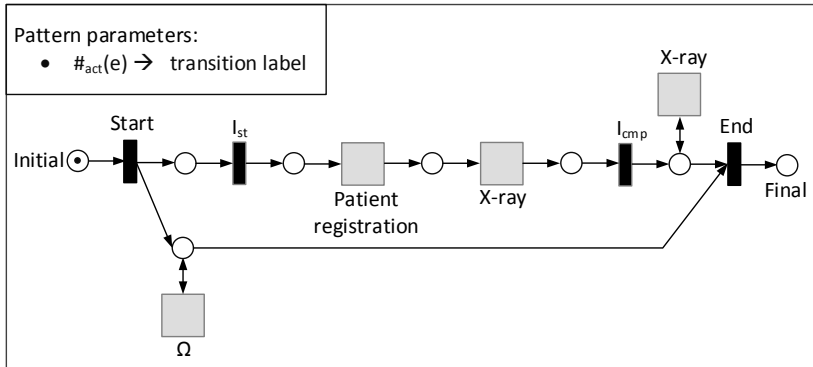


Figure 4.21: Activity patient registration must be followed by activity X-ray.

*MRI* together with *CT-scan* is a violation.

Although the health care process analyzed was quite complex and diverse, using our technique we could abstract from the complexity and focus on the specific rules. Altogether, we could identify all deviating cases from the given compliance rules in the real-life log. Furthermore, we could precisely locate the violations in the process and demonstrate what should have happened to compensate the observed violation.

## 4.4 Elicitation of Formal Compliance Rules

Earlier in this chapter and in Chap. 3, we discussed the necessity of specifying compliance constraints precisely. In practise, compliance constraints are often described in natural language which makes automated compliance checking a difficult task. In addition, the descriptions of these constraints incorporate domain specific terminology, as well as structure and definitions. With the goal to facilitate the automated compliance checking, we already collected a comprehensive collection of control flow compliance rules which allows to formally capture a large set of compliance constraints (these rules are listed in Table 4.1 and their formalization can be found in Appendix A). Yet when instantiating any of these rules in context of a specific business process, many details and subtle aspects should be considered such that the rule exactly specifies the behavior intended by domain experts.

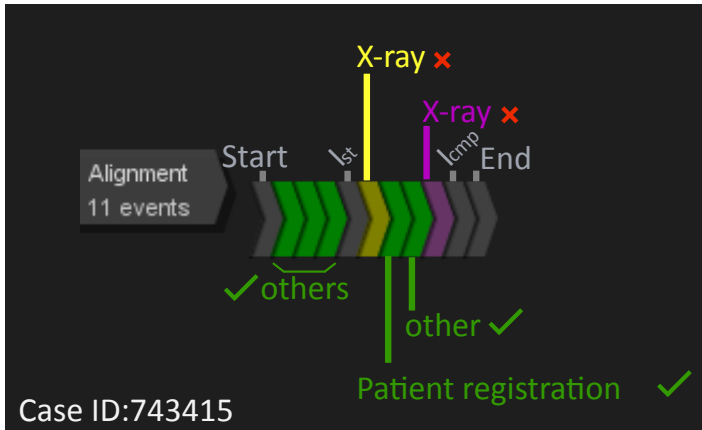


Figure 4.22: An example of a non-compliant case. Activity *X-ray* is executed in a wrong position.

In the following, we first illustrate the problem on a concert example and then present an approach to help domain experts elicit compliance rules.

#### 4.4.1 The Gap Between Informal Compliance Rules and Formal Specification

Consider the compliance constraint that we discussed in Sect. 4.3.2. This constraint is obtained from internal policies of a specialized hospital stating: “For every patient registered in the hospital an X-ray must be taken”.

Our interpretation of this constraint led to the formalized form of this rule shown in Fig. 4.21. This pattern enforces that *patient registration* must be followed eventually by activity *X-ray*. Other activities may occur an arbitrary number of times during the case. Activity *patient registration* may occur only once. Activity *X-ray* may be repeated several times.

However, the rule given above could also be interpreted slightly differently. For example, Fig. 4.21 allows other activities to occur before *patient registration* but one could also have interpreted the rule differently and require that *patient registration* is the first activity in the case. This alternative interpretation is formalized by the model in Fig. 4.25. There is yet another interpretation possible: activities other than *patient registration* and *X-Ray* are only allowed to occur outside the rule instance as formalized in Fig. 4.26. As a side-effect, it

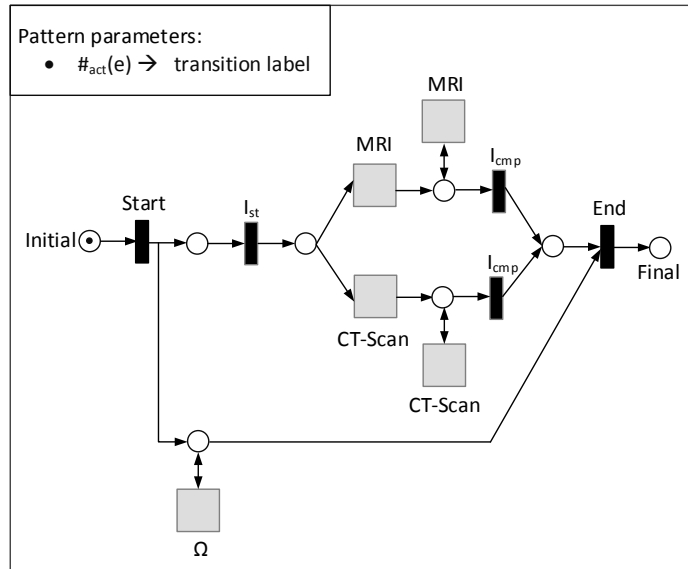


Figure 4.23: Either CT-scan or MRI test must be taken for a patient.

also enforces that activity *X-ray* to be executed directly after *patient registration*.

Although the constraint seems very straightforward, it is important to decide about some details during the formalization, independent of the chosen formalization: e.g., (1) whether *patient registration* should be directly followed by *X-ray* or other activities may occur in between the specified sequence; (2) whether it is allowed that other activities occur before *patient registration* or a patient cannot receive any treatment without registration; (3) whether a patient can be registered several times and if yes; (4) should the specified sequence be followed every time; (5) whether it is allowed that the specified sequence never occurs, i.e., if it is allowed that a patient is never registered. Note that deciding about any of this details may change the result of checking and some cases evaluated initially as compliant, may become non-compliant or vice versa.

Consequently technical experts may need to invest considerable effort choosing a rule description and its formalization and check whether the recorded process executions conform with it, only to later determine that the property has been specified incorrectly, because *in the step from natural language to precise formalization many subtle aspects of the constraint have to be considered*.

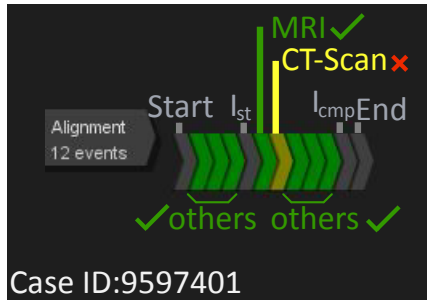


Figure 4.24: An example of a non-compliant case. Activities *MRI* and *CT-scan* both are executed within a case.

Interpreting an informal rule with all its details can be surprisingly difficult and must be done by domain experts who are usually less familiar with the different formalisms. Therefore, an approach is required to hide the complexity of the formalization from business user and at the same time support automated compliance checking.

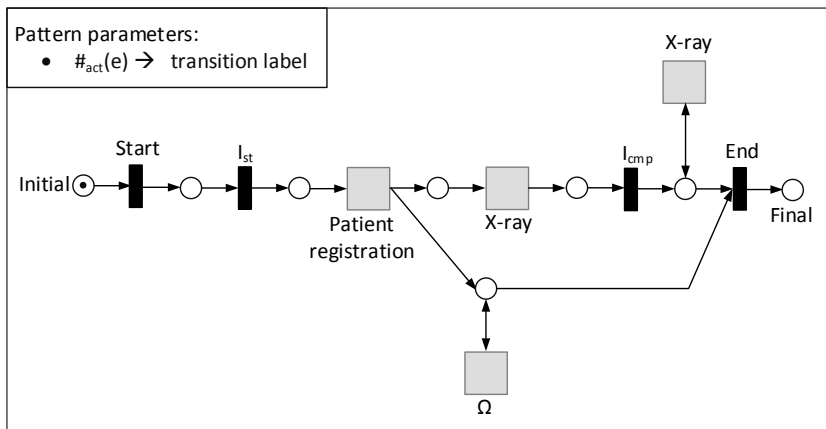


Figure 4.25: The pattern enforces that a patient must be registered before receiving any treatment.



#### 4.4.2 A Methodology for Eliciting and Configuring Formal Compliance Rules

Numerous researchers have developed specification patterns to facilitate construction of formal specification of compliance constraints. Feedback indicates [66] that these patterns are considered helpful but they fail to capture subtle aspects of a specific constraint. In addition, adaption and application of these patterns are not trivial for many practitioners as they are less familiar with the underlying formalization.

To address the gap between informal constraints and formal compliance specifications, we propose an interactive approach for using tacit knowledge of domain experts to specify compliance constraints. Our approach aims at (1) enabling business users and compliance experts to specify compliance constraints as atomic Petri net patterns and (2) encouraging them to think about the subtle aspects of their intended behavior when specifying a constraint. We develop a ‘*question and answer*’ approach based on “disciplined” natural language. Such an approach is also used in property specification for software development in [26, 96, 123] and is a suitable candidate for compliance specification. The key components of our approach are *question trees*, and *generic configurable compliance patterns*.

The atomic compliance patterns we discussed earlier in this chapter and in the previous chapter describe a specific compliance rule precisely and in a compact way. Various atomic patterns may have common constraints but dif-

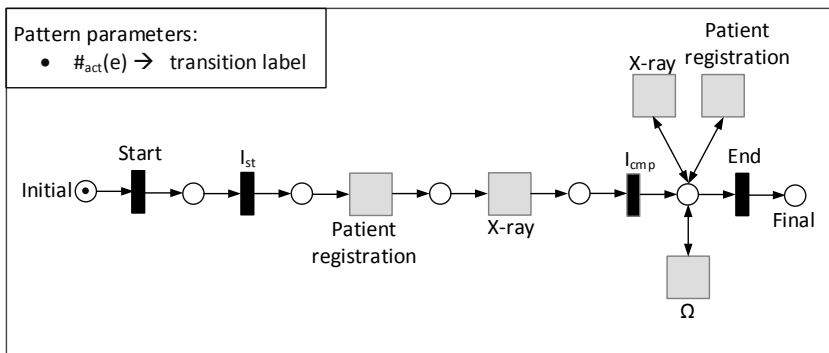


Figure 4.26: The pattern enforces multiple registration of a patient and direct sequence of *patient registration* followed by *X-ray*.

fer in subtle aspects. *Configurable compliance patterns* are pre-formalized in configurable Petri nets and capture common compliance constraints. As mentioned, we have developed a repository of configurable compliance patterns that is listed in Appendix A. Every of such pattern allows for alternative variations of a compliant behavior. Selecting an appropriate configurable pattern and configuring it for its configuration options are done interactively with users and will result in a specific *atomic compliance pattern* that can be used then for compliance checking.

In this section, we explain how our approach can help practitioners elicit a control-flow compliance rule by making informed choices between different variations of a rule and formalize it as an atomic compliance pattern. Fig. 4.27 gives an overview of our approach for compliance specification. This approach is built upon a repository of configurable compliance patterns.

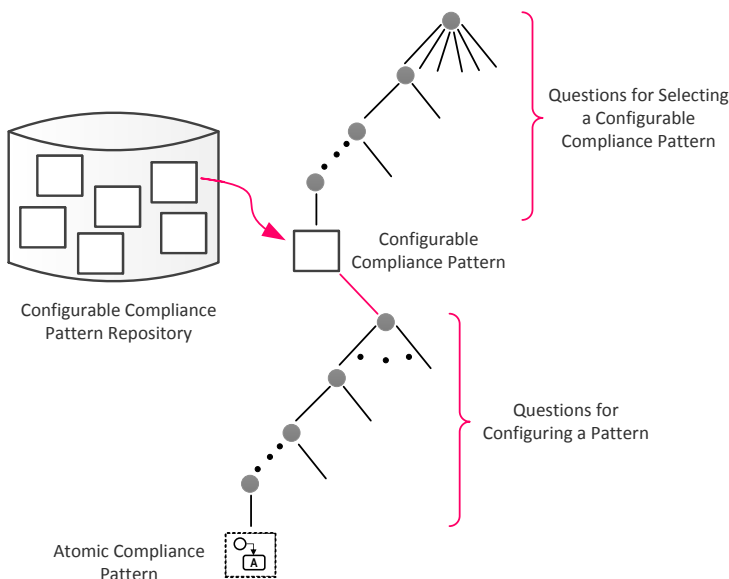


Figure 4.27: Control-flow compliance rule elicitation and formalization overview.

### 4.4.3 Configurable Compliance Pattern Repository

The collection of control-flow compliance rules that is listed in Table 4.1 contains over 50 rules. Considering subtle variations of each of these rule, the number of rules will increase rapidly. To help the user selecting the right rule and a precise variation of a rule, we consolidated the compliance rules by merging similar rules (that differ in variations of subtle semantic aspects) into one configurable compliance pattern that is easier to describe in general terms. Consolidating similar rules into a configurable pattern is done manually following a generic approach. We first define a core behavior for the configurable pattern and then extend the core behavior with all possible configuration options. These configuration options allow us to define different variations of a compliance constraint. The idea is that a user first picks a general configurable pattern with all its configuration options and then configures it w.r.t. various subtle aspects.

#### **Consolidating and organizing compliance rules in a repository.**

The configurable compliance pattern repository is built upon the collection of control-flow compliance rules listed in 4.1.<sup>1</sup> We consolidated these rules by merging similar rules into a configurable pattern to eliminate redundancies and allow for specifying different variations of a rule. A configurable compliance pattern is a configurable Petri net which describes a group of compliance rules in a concise way. Originally configurable process models [108,137] were proposed to describe variants of a reference process. Here, we are applying the concept to describe variants of compliance constraints.

Every configurable compliance pattern is parameterized and formalized in terms of Petri nets with a core component. This core structure enforces a core behavior (e.g. a sequence). In addition a pattern has several other components which determine variations of the core behavior. The core behavior enables a clear distinction between commonalities shared among compliance rules in one category and variability.

To consolidate the rules in Appendix A, we studied rules which share a common behavior. We kept the core component in a configurable pattern and added all possible configuration options to it. The resulting configurable pattern can describe all the original rules it is derived from, and many more because of the

---

<sup>1</sup>The formalizations of these rules in terms of atomic Petri net patterns are available in Appendix A.

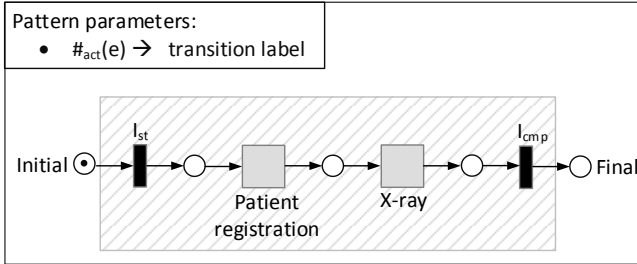


Figure 4.28: Sequence of *patient registration* and *X-ray*.

new possible combination of different configuration options. The configurable patterns are sound by design.

Please recall the example given earlier in Sect. 4.3.2. The Petri net pattern shown in Fig. 4.28 formalizes the core behavior of the requirement of this example.

The core of the rule is formalized in the hatched part between transitions  $I_{st}$  and  $I_{cmp}$  which represents an instance of the compliance rule. The rule becomes active when  $I_{st}$  fires and it is satisfied when  $I_{cmp}$  fires. The core structure of the pattern enforces: if *patient registration* occurs then it must be followed by *X-ray*. If we want to add other options to the behavior specified in this pattern, we need to add some more components to the pattern and build a configurable pattern out of it.

The configurable pattern shown in Fig. 4.29 is parameterized over the activity names such that activity  $A$  is *Patient registration* and activity  $B$  is *X-ray*. The configurable pattern allows for defining variations of the core behavior and by blocking or activating a component we can extend or limit admissible behavior. In the following we will explain the components of the configurable pattern in Fig. 4.29 and explain how blocking or activating a component can change the behavior of the pattern.

- *Comp.1* ( $\Omega$ ): Activating this component allows for occurrences of other activities after the specified sequence.
- *Comp.2* ( $\Omega$ ): Activating this component allows for occurrence of arbitrary other activities in between the sequence  $\langle \textit{Patient registration}, \textit{X-ray} \rangle$  and blocking this component enforces that activity *patient registration* must be followed directly by *X-ray*.

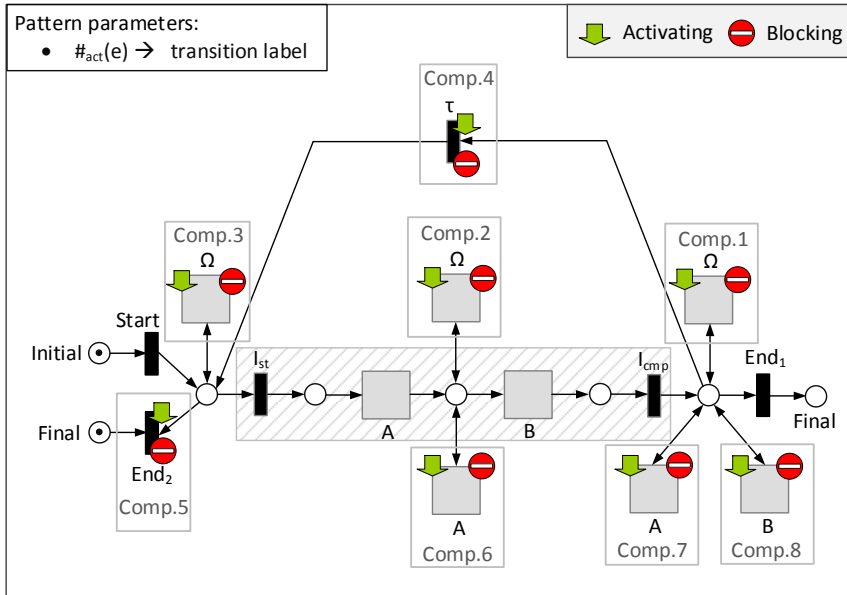


Figure 4.29: Configurable sequence of *Patient registration* and *X-ray*.

- *Comp.3* ( $\Omega$ ): Activating or blocking this component, enforces that other activities may occur before *patient registration* or not.
- *Comp.4* ( $\tau$ ): Activating or blocking this component allows that the sequence  $\langle \textit{Patient registration}, \textit{X-ray} \rangle$  occurs multiple times in a case or not.
- *Comp.5* ( $\textit{End}_2$ ): Activating this component allows that the specified sequence  $\langle \textit{Patient registration}, \textit{X-ray} \rangle$  never happens for a patient and blocking it enforces at least one occurrence of the specified sequence.
- *Comp.6* (*A*): Activating or blocking this component allows that several registrations of a patient can be followed by one execution of activity *X-ray* or not.
- *Comp.7* (*A*): Activating or blocking this component allows that after occurrence of the sequence  $\langle \textit{Patient registration}, \textit{X-ray} \rangle$  a patient gets registered without a following *X-ray* or not.

- *Comp.8 (B)*: Activating or blocking this component allows that activity *X-ray* occurs independently from the specified sequence of  $\langle \textit{Patient Registration, X-ray} \rangle$  or not.

When designing a configurable compliance pattern, we abstract from concrete examples and consider all possible configuration options. The configuration options we address in our approach include: *activating*, *blocking* a transition, an arc or a group of transitions and arcs. In addition, we consider configuring *arc weights*.

By developing configurable patterns, we could eliminate redundancies in a compliance rule family and reuse the commonalities, thus decreasing the number of patterns to 22 configurable compliance patterns having 0-38 configuration options each. This way, over 1000 different compliance patterns can be derived (including the original 50 patterns) through picking different configuration options. The 22 configurable patterns are shown in Appendix C. The configuration will result in an atomic compliance pattern that can be used for compliance checking using the technique described in Sect. 4.1.

#### 4.4.4 Question Tree

In order to enable domain experts to specify the intended behavior of a compliance constraint, we apply an interactive question and answer based approach. We aim to guide users to select an appropriate configurable compliance pattern and elaborate on how to configure its configuration options such that it represents the intended behavior. Thus we apply a Question Tree (QT) representation which is basically a decision tree and its content is based on disciplined natural language.

We apply *two* distinct question trees (see Fig. 4.27); a set of questions which guide the user to *select a specific configurable compliance pattern* and a set of questions which are asked to resolve different configuration options of it in order to specify details of intended admissible behavior.

##### Questions to Select a Configurable Compliance Pattern.

The QT of the first phase breaks the problem of deciding which configurable pattern is most appropriate by asking users to consider only one differentiating criteria at a time which finally leads to choice of a configurable pattern. In this phase, QT has a hierarchical structure and this structure supports the separation of concerns, only presenting a question to the user that is relevant in context of

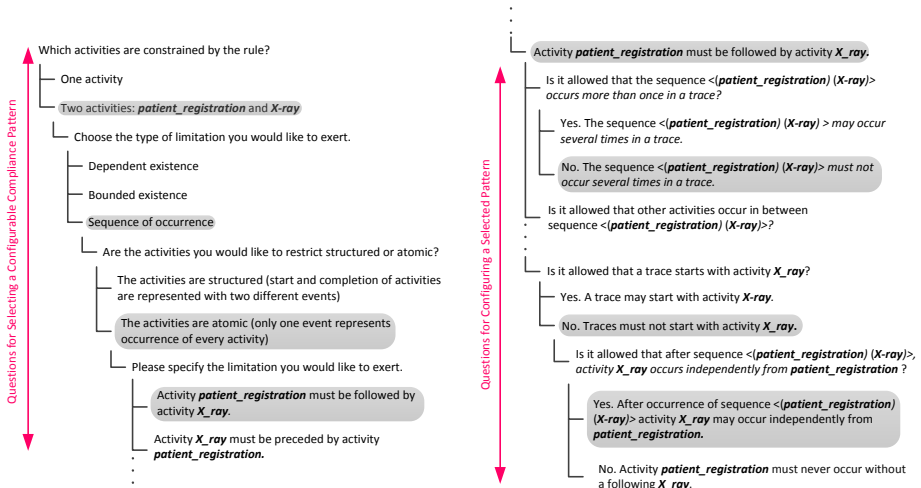


Figure 4.30: QT-phase1 (left), QT-phase2 (right).

their previous answer. A new question that can be revealed after answering a given question is a child question of that previous answer; the previous question is the parent question of that child question. By selecting a different answer to a parent question, the user will explore a different set of child questions that are relevant to that answer and will arrive at a different configurable pattern. Figure 4.30 QT-phase1 (left) presents the question tree for selecting a configurable pattern in the example discussed earlier in the previous section.

### Questions to Configure a Configurable Compliance Pattern.

Questions in the second phase concern configuring subtle behavioral aspects of a specific pattern. Not all questions in this phase have a hierarchical structure. That is, many questions in this phase can be asked in any order, since there are configuration options in each of configurable patterns which are conceptually orthogonal to each other. See for example choices related to *Comp.4* ( $\tau$ ) and *Comp.2* ( $\Omega$ ) in Fig. 4.29. These questions will be presented to the user together and s/he may answer them in any order based on personal preferences and understanding.

However, some options are not orthogonal. For example, choices related to *Comp.4* ( $\tau$ ) and two components *Comp.7* (*A*), and *Comp.8* (*B*) are orthogonal

in Fig. 4.29. If we do not allow for repetitive occurrences of the sequence  $\langle A, B \rangle$ , we cannot allow for activation of both components *Comp.7 (A)*, and *Comp.8 (B)* (note that still activation of one maybe allowed). In such cases, the former question is only asked if a certain pre-configuration holds for it.

Note that the configurable pattern, i.e., the underlying Petri net and its configuration options are *not shown* to the end user and the user only deals with textual descriptions of rules in terms of questions and answers. In the back-end, every *answer node* of QT in the second phase is mapped to a configuration option in a configurable pattern and configures the pattern based on choices user makes. The configuration process is continued until all details of a compliant behavior is decided. Fig. 4.30 QT-phase2 (right) presents partially the question tree of the second phase for the example of the previous section.

#### 4.4.5 Illustrating a Compliance Rule to a Domain Expert.

The configurable compliance pattern is hidden from user and s/he is only represented with questions and answers which are designed in a simple structured and clear text. In order to remove any ambiguity for the user while answering questions regarding subtle behavioral aspects, there are several compliant and non-compliant sample traces given for every answer. That is, a user can easily see how a certain choice can impact (i.e., limit or extend) admissible behavior. The configured compliance pattern determined in the second phase is a Petri net that can be used for automated compliance checking using the alignment technique we discussed earlier in this chapter in Sect. 4.1.

In the following we show a walk-through example illustrating how a user selects and configures a compliance rule using the two question trees.

## 4.5 Supporting Domain Experts to Specify Compliance Constraints

In this section we will elaborate our methodology (Sect 4.4.2) and its implementation by going through a real life example step by step and showcase how a user who is not familiar with any formalism specifies his/her admissible behavior considering its detailed aspects.



### 4.5.1 Implementation in ProM

The technique is implemented in the *Compliance* package of the Process Mining Toolkit ProM 6.6, available from <http://www.processmining.org>. The package contains the repository of all configurable compliance patterns. The **Elicit Compliance Rule** plug-in takes a log as input and returns a compliance rule using the approach of Sect. 4.4.2. The returned rule can be used for compliance checking using the *Check Compliance of a Log* plug-in. In the following we show how a user can use this implementation to select and configure a compliance rule.

### 4.5.2 Elicitation of the Case Study Compliance Pattern

We chose the event log taken from *BPI Challenge 2011* available from [1]. The log is taken from a Dutch Academic Hospital. This log contains some 150.000 events in over 1100 cases. Apart from some anonymization, the log contains data required to check the rule as it came from the hospital's systems. Each case refers to a patient of the hospital's Gynaecology department. The log contains information about when certain activities took place, which group performed the activity and so on. Many attributes have been recorded that are relevant to the process.

To demonstrate the approach, we chose to formalize a rule that captures the following behavior observed on the event log [18]: *Glucose level must be estimated 4 times repetitively if a patient diagnosed for cervical cancer of uterus (diagnosis code M13) and classified as an urgent case*<sup>2</sup>. We have preprocessed this log for patients who are suffering from cervical cancer of uterus. Urgent patients are those cases where at least one activity of type urgent is manifested. A very common activity representing an urgent case is *'haemoglobin photoelectric-urgent'*. If we rephrase the constraint and substitute the activity names with corresponding event names in the log, the rule states: *In case of patients diagnosed for code M13, activity 'haemoglobin-photoelectric-urgent' must be followed 4 times by activity 'glucose-urgent'*.

We take this log as input and run the *Elicit Compliance Rule* plug-in that implements the approach of Sect. 4.4.2. The very first question of the questionnaire always asks the user to specify the number of activities of primary interest. For this a list of available activities in log is shown to user and the user can choose the activities s/he wants to restrict from this list. Depending on

---

<sup>2</sup>Please note that the observed behavior does not indicate a medical rule but we chose this observation to show how we can specify a behavior using *Elicit Compliance Rule* plug-in

the number of activities chosen different sets of questions will be triggered. For instance if the user chooses one activity of primary interest, the next question will ask about the number of times a specified activity is allowed to occur. If more than one activity (e.g. in case of our example two activities) is chosen, the questions related to relationships between chosen activities will be asked. In our example:

- Which type of limitation you would like to exert?
  - Dependent Existence: define whether the occurrence or non-occurrence of an activity imposes an obligation on occurrence or non-occurrence of another activity, e.g. define an inclusive relation between two activities.
  - Bounded Existence: define whether number of occurrences of one activity is dependent on number of occurrences of the other activity.
  - Sequence of Occurrence: define whether there should be a sequential relation between occurrence of two activities, e.g. define a precedence or simultaneous relation between two activities.
  - Bounded Sequence of Occurrence: define whether a specified sequence must be repeated.

We choose *Bounded Sequence of Occurrence* from the list of alternative answers. As the result of this choice, a configurable pattern is selected in the back-end and questions to configure the selected pattern are presented.

The first question from the second phase will ask whether the user wants to limit the repetition of activity 'glucose-urgent' after activity 'haemoglobin-photoelectric-urgent' and if yes how many times 'glucose-urgent' must occur after 'haemoglobin-photoelectric-urgent'. Figure 4.31 illustrates this step using the 'Elicit Compliance Rule' plug-in in ProM where we chose: 4 times repetition of 'glucose-urgent' after 'haemoglobin-photoelectric-urgent'. In order to support the user to make informed choices, for every answer a sample compliant trace and non-compliant trace are given as shown in Fig. 4.31. Additionally, the outcome of the currently chosen configuration is visualized to the user: the selected and partially configured rule is used to check compliance of the log w.r.t. this preliminary rule using the compliance checking technique presented in Sect. 4.1. The screen in Fig. 4.31 shows several compliant and non-compliant traces by which the user can use her domain knowledge to assess which answer translates her intention best. Subsequent questions assist the user in deciding about details of the intended behavior. These questions concern configuration options which

are orthogonal to each other, hence they can be resolved in any order. These questions include:

- Is it allowed that other activities occur between occurrences of activity '*haemoglobin-photoelectric-urgent*' and '*glucose-urgent*'?
- Is it allowed that other activities occur between occurrences of activity '*glucose-urgent*'?
- Is it allowed that several occurrences of activity '*haemoglobin-photoelectric-urgent*' are followed by specified repetitions of activity '*glucose-urgent*'?
- Is it allowed that activity '*glucose-urgent*' occurs before activity '*haemoglobin-photoelectric-urgent*' independently from the defined sequence?
- Is it allowed that the specified sequence of  $\langle \textit{haemoglobin-photoelectric-urgent}, \textit{glucose-urgent}, \dots, \textit{glucose-urgent} \rangle$  occurs multiple times?  
4
- Is it allowed that the specified sequence of  $\langle \textit{haemoglobin-photoelectric-urgent}, \textit{glucose-urgent}, \dots, \textit{glucose-urgent} \rangle$  never occurs?  
4

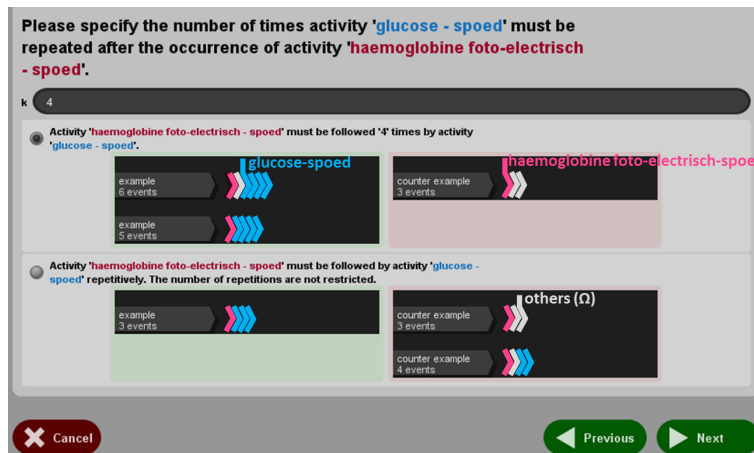


Figure 4.31: *Elicit Compliance Rule* plugin: Example compliant and violating cases when a specific choice is made by the user.

- Is it allowed that after the specified sequence  $\langle \textit{haemoglobin-photoelectric-urgent}, \underbrace{\textit{glucose-urgent}, \dots, \textit{glucose-urgent}}_4 \rangle$ , activity *haemoglobin-photoelectric-urgent* occurs without being followed by repetitions of *glucose-urgent*?

Resolving these questions yields a *configured atomic pattern* which describes precisely the intended behavior. This atomic Petri net pattern can be used further for automated compliance checking.

## 4.6 Control-Flow Compliance Rule Elicitation and Checking of a Composite Compliance Model Versus a Set of Atomic Compliance Patterns

We have described in Sect. 4.4 how to elicit an atomic compliance pattern from a set of configurable compliance patterns. The collection of configurable compliance patterns allows for specifying various compliance rules constraining up to four activities and their relations as an atomic compliance pattern. However, in case we would like to specify more activities in one model or combining various rules we need to design the pattern manually.

The atomic patterns, derived automatically using our methodology, can be combined using the synchronous product of the atomic patterns [143,150]. The synchronous product matches activities by name and then merges their dependencies and relations. However building a composite compliance model as the synchronous product of a set of atomic patterns will usually result in a complex and large model. Therefore we usually develop a composite compliance model manually following the principles listed in Sect. 4.1.5. Earlier in Sect. 3.4.6, we discussed composite compliance models. An example of a composite model is shown in Fig. 7.4 and will be discussed later in Chap. 7.3

We have discussed the checking procedure for atomic compliance patterns in Sect. 4.1. Checking an event log against a set of atomic compliance patterns or a composite compliance model does not differ in terms of the steps that should be taken to actually prepare the event log and detect control-flow violations. The choice of the checking attribute is the same for checking a composite model or a set of atomic patterns. Note that since in a composite compliance model usually more activities are specified, the number of the activities that during the abstraction and shortening of the event log will be mapped to  $\Omega$  events or will be removed from the event log are less.

As we discussed in detail in the previous chapter (see Sect. 3.4.6), when we check an event log against a composite compliance model, the alignment technique returns an alignment which is globally optimized for the combination of the compliance constraints enforced in the respective composite model. Whilst on the contrary, when we check an event log against a set of atomic patterns, the alignment technique for each rule returns an alignment that is optimized local to that rule. This difference has impact on the diagnostics that we obtain (for details see Sect. 3.4.6).

In general when compliance rules overlap, we will receive more precise diagnostics if the rules are modelled combined as a composite compliance model. On the contrary when rules do not overlap, it is better to formalize them as a set of atomic compliance patterns because the diagnostics will be more precise, and the Petri net patterns are smaller. In the context of control-flow compliance rule, rules overlap when they put constraints on common activities.

## 4.7 Related Work

Our review of related work in control-flow compliance checking has two main components: (1) the elicitation and formalization of control-flow compliance constraints and (2) compliance checking and diagnostics obtained as the result of checking. We will discuss related work along these two lines.

### Elicitation and formalization of compliance constraints

The informal description of compliance constraints can be interpreted differently in context of different business operations. Therefore, precise specification of them is necessary [65, 84].

Various approaches and formalizations are proposed for formulating compliance constraints. Some approaches, such as [113], support the modeling of control-objectives within business process structures using a modal logic based approach called FCL (Formal Contract Language). The formalization believed to offer the right trade off between expressive power and computational complexity. However, applying the technique still requires a solid knowledge of the underlying formalization. In this approach process models are annotated by “control tags”; where tags are identified affecting control-flow, data, resource and time. The authors of [52], also adopt FCL as formalism to model control objectives.

Several works such as [48, 81, 85, 136] model compliance concerns in LTL (Linear Temporal Logic). Many constraints formalized in LTL are supported by Declarative templates, which makes using LTL-based checking techniques easier. However, to use these templates one should have a thorough understanding of Declare and the behavior specified by each template. In addition one should be aware that capturing all subtle aspects of a compliance constraint may lead to many Declare rules. It is worth mentioning that declarative modelling is not necessarily perceived easier and more intuitive for business users than procedural modelling of processes [102].

The authors of [79] specify compliance rules in Compliance Rule Graphs (CRG) which is a graph-based rule modelling language. They choose graphs as a suitable representation for expressing occurrence and ordering relation between activities, and general structure of rules (i.e., if some conditions apply, then some consequences must also apply). The problem with above mentioned approach is again the task of formalizing constraints. It requires knowledge about the formalization language, a knowledge that business users who are typically in charge of compliance may be less familiar with. In addition the type of rules that are covered by this approach is less compared to our repository of rules and their variants.

To facilitate the task of formalization, many approaches use pre-formulated templates and specification patterns for formulating compliance constraints [42, 46, 119, 132]. Specification patterns are extensively used in software development [4, 26, 37, 66, 123]. In [42, 132], Elgammal et al. introduce a pattern-based approach for capturing compliance constraints. Their patterns are parameterized and formalized in LTL. Most of these approaches use some type of structured natural language and pre-formulated templates to construct formal specifications that can then be analyzed. Often, these informal specifications are initially mapped to an intermediate representation (e.g. model-driven patterns), at which point context dependencies and ambiguities are resolved. In order to make the approach usable for business users, authors developed a toolset where the user can define compliance constraints using a specialized version of declare modeling notation.

A common problem in most of above mentioned works is that pre-formulated patterns are limited and hard coded; hence they fail to capture subtle aspects of different compliance constraints. In addition in most of the approaches, mapping and adapting patterns in a specific context requires extensive knowledge in specification languages. Our approach aims to allow compliance specification for end users without such an extensive knowledge.

Our compliance elicitation approach [98] is also based on pre-formulated

patterns. Yet it enables end users to specify precisely the compliance rules without being exposed to technicalities of any formalization. Our repository of configurable compliance patterns include a comprehensive collection of compliance rules that can be configured to capture details and various aspects of compliance constraints.

### Compliance checking and diagnostics

The core challenge in addressing compliance on event behavior is to compare the prescribed behavior (e.g., a process model or a set of rules) to an observed behavior (e.g., audit trails, work-flow logs, transaction logs, message logs, and databases). For this various approaches have been developed. In [136], it is shown how constraints expressed in terms of Linear Temporal Logic (LTL) can be checked with respect to an event log. This verification technique checks whether each trace in an observed behavior complies to a LTL formula and the result is shown as violating or compliant. Another logic-based checking approach is presented in [84]. In this approach compliance rules are formalized using Event Calculus (EC). This formalization supports the constraint activation (similar to our concept of rule instance). The verification technique tracks a running case and indicates the activation and evolution of compliance constraints, then it reports on the compliance status of each activation. In [80] the authors even present ways to quantify the degree of compliance.

Another group of compliance checking techniques are automaton-based checking. For instance in [81], an automaton describes a compliance rule. Usually the automaton is not directly generated from a compliance rule, but rules are modelled using a formalism such as linear temporal logic (LTL) and the automaton is generated from the formalization. Then a trace from the observed behavior is replayed on the automaton. If the automaton reaches an accepting state, the rule is satisfied otherwise it is violating.

In [85] both LTL-based and SCIFF-based (i.e., abductive logic programming) approaches are used to check compliance with respect to a declarative process model and an event log.

The authors of [79] model compliance constraints as Compliance Rule Graph and check observed behavior directly on graphs. Their approach does not require any transformations into other representations (for instance to an automaton). They also enable the concept of rule activation.

The authors in [147] leverage complex event processing for monitoring process conformance. They create behavioral profiles from process models (can be

considered as compliance constraints). The technique generates and run complex event queries accordingly. Then the detected violations are aggregated.

The above mentioned approaches are compliance monitoring techniques. That is, they check during the execution of a business processes whether it is compliant or not. Our approach is categorized as *backward compliance checking*, i.e., our technique identifies compliance violations after completion of execution of a process. In that sense, backward checking techniques should provide diagnostics about violations related in a case globally (not per activation of a rule). These diagnostics clarify how the violations in a case must be compensated globally to make the case compliant. Using this setting, we can leverage extensive root-cause diagnostics over all compliance results. In Chap. 7.3, we explain how our root-cause analysis approach is built upon the diagnostics provided by our compliance checking technique.

Several conformance checking approaches are developed for backward compliance checking. In [27], authors present an approach to measure discrepancies between a process and a model. Event streams (sequence of events) are built from process models and are compared with the execution event streams (observed sequence of events). Then the distance between the two event streams are measured. Another conformance checking approach is presented in [111]. In this approach compliance constraints can be formalized as a Petri net model. The technique replays the observed behavior on Petri net model while counting “missing” and “remaining” tokens. This approach does not provide precise and detailed diagnostics about each violation.

State-of-the-art techniques in conformance checking retrieve this information by computing *optimal alignments* [10, 134, 135] between traces in the event log and “best fitting” paths in the model. The authors of [31] adapted the alignment-based approaches defined for procedural models and present a conformance checking approach for declarative constraints. Existing approaches to backwards compliance checking have two main problems. First of all, the *elicitation of compliance rules is not supported well*. End users need to map compliance rules onto Declare models (or expressions in temporal logic) or encode the rules into a Petri-net-like process model. Second, many of the existing checking techniques can discover violations but *do not provide useful diagnostics* [27, 111]. Those that provide detailed diagnostics [10, 31, 135] are not tailored for compliance checking.

Our compliance checking approach [97] builds upon cost-based conformance checking [10, 135] and detects all activations of a compliance rule and can locate compliance violations. It also indicates the compensation for detected violations (i.e., what should have happened instead the detected violation).



## 4.8 Concluding Remarks

In this chapter we discussed control-flow compliance analysis from various angles including the formalization of compliance rules, checking rules and offering precise diagnostics about violations.

We developed a set of principles that can be used to formalize compliance rules. Various control-flow compliance constraints were discussed. We proposed a question and answer approach to improve formalization of compliance rules by enabling business users to precisely elicit and formalize these rules. The question and answer approach presented is based on configurable templates. We have developed a comprehensive repository of configurable compliance patterns that allows for specifying different types of compliance constraints we found in literature and many more. By selecting a configurable compliance pattern and configuring its options we can bridge the gap between informal descriptions of compliance constraints and automated compliance checking for commonly-required compliance constraints. The configurable compliance pattern is selected and configured for its options, interactively with end-users and will result in an atomic compliance pattern. The elicited compliance pattern can be used further for compliance checking. Our elicitation approach was implemented as a ProM plug-in. In addition, we developed some principles that guide the user to build composite compliance models.

We presented a robust technique for backwards compliance checking which enables us to provide diagnostic information in case of violations. Our approach takes an event log and a compliance pattern and computes an alignment between between the model and the log. The alignment then indicates where the event log deviated from the model. It also indicates how the violation could be compensated. This approach is also supported by a ProM plugin. We have tested our techniques using real-life logs and compliance constraints.

We will extensively rely on control-flow checking for checking temporal, data-aware and resource-aware compliance rules in chapters 5 and 6.

# Chapter 5

## Temporal Compliance Checking

This chapter focuses on elicitation and checking of *temporal compliance constraints* in a systematic way. Temporal compliance rules constrain *when* activities may occur. Our temporal compliance analysis approach is built upon the control-flow compliance checking (presented in the previous chapter). To check whether an activity occurred at the right time, we need to first check whether it occurred at all.

In Chap. 2, we illustrated our ideas for formalizing temporal constraints as data-aware Petri nets and in Chap. 3 and 4 we discussed the problem of translating informal compliance constraints to formal models. In this chapter, we go more in-depth showing how to formalize various kinds of temporal constraints using a single generic temporal pattern. The generic pattern should capture all possible temporal rules and instances of these rules. We will discuss in this chapter, how the combination of two techniques, control-flow checking and temporal checking, achieves this goal. We explain why the concept of *rule instance* discussed in previous chapters plays an essential role in our temporal compliance checking approach. Only activities within a rule instance are governed by temporal constraints. During the control-flow checking, we make sure specified activities fall inside an instance of a rule. A clear boundary on activities which are inside rule instances and the ones that are not, together with the generic temporal pattern are inputs for our temporal compliance checking technique.

We listed the type of diagnostics we would like to obtain in Chap. 3. We explained in Chap. 2, depending on how rules are formalized (atomic rules versus a composite model) and how alignments are configured (cost of moves),

diagnostic information can differ. In this chapter, we investigate this matter for temporal constraints. Next to the detection of temporal violations, the temporal checking approach provides explanations about the violations, i.e., how the violations can be compensated to make a case compliant. These compensation solutions are not built locally per violation but they consider the global impact of the proposed compensation values in a case. This feature of our approach leaves a quality on the diagnostics that one should be aware of it when interpreting the results.

Figure 5.1 illustrates how the content of this chapter is organized in different sections. In the remainder of this chapter, we first consider an extensive collection of temporal compliance constraints identified in literature in Sect. 5.1. We discuss different categories of rules in the collection and provide examples for each category. In Sect. 5.2, we explain in detail our approach for temporal compliance checking that was outlined in Chap. 2. We will show how to integrate control-flow and data-aware checking on a technical level to achieve detailed diagnostic information about violations of temporal compliance constraints. In Sect. 5.3, we introduce a generic pattern for formalizing various temporal compliance constraints and show how to instantiate it for concrete examples.

In Sect. 5.4, we discuss the differences between checking atomic rules versus composite models that also cover temporal constraints. We show that the generic pattern of Sect. 5.3 can cover some cases of composite rules but we also discuss which set of temporal rules cannot be checked in a composite model.

In Sect. 5.5, we return to the problem of preparing event logs for compliance checking. We adopt the log preparation techniques discussed in Chap. 3 and Chap. 4, such that the temporal compliance checking technique works for all temporal compliance rules. Finally, we discuss which factors influence the quality of diagnostic information in temporal compliance checking in Sect. 5.6.

We will showcase the applicability of our temporal compliance checking technique on real-life data with the help of a case study in Sect. 5.7. Further, we discuss the diagnostics obtained in this project. The related work will be discussed in Sect. 5.8 and Sect. 5.9 will conclude this chapter.

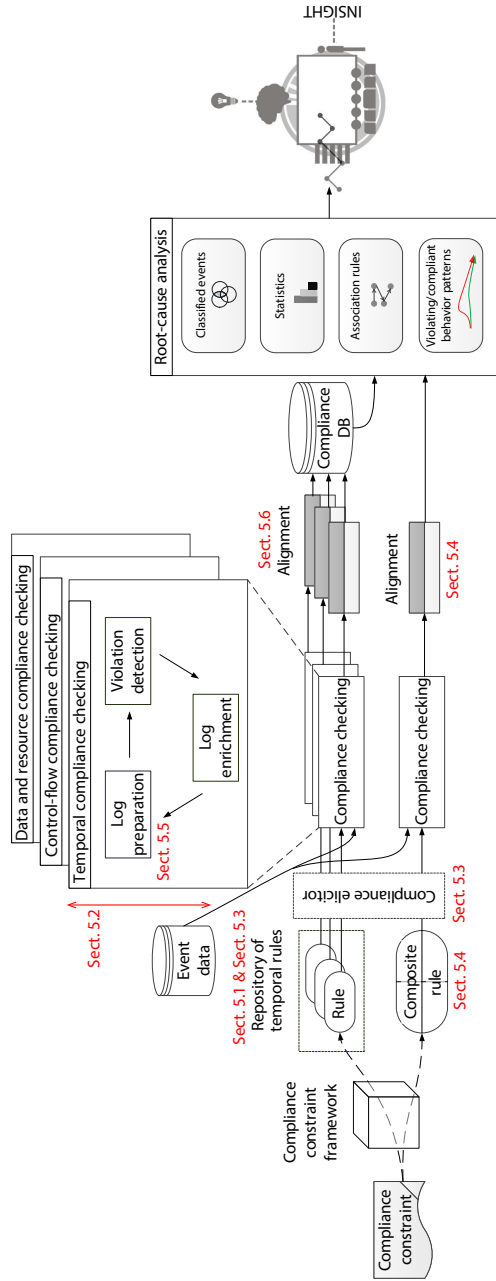


Figure 5.1: Thesis road map gives the mapping of the sections in Chap. 5 on to our compliance analysis approach.

Category (Rules)	Description
Instance Duration (2)	Limits the time period in which a control-flow rule instance must hold. [149]
Delay Between Instances (1)	Limits the delay between two subsequent instances of a control-flow rule [5, 68, 69, 88]
Validity (3)	Limits the time period in which an activity can be executed. [68, 69, 88, 149]
Time Restricted Existence (2)	Limits the execution time of an activity in calendar. [68, 69, 88]
Repetition (2)	Limits the delay between execution of two subsequent activities. [68, 69, 73, 88, 149, 153]
Time Dependent variability(1)	Limits choice of a process path among several ones with respect to temporal aspects. [68, 69, 88, 149]
Overlap (4)	Limits start and completion of an activity to start and completion of another activity. [68, 69, 88, 149]

Table 5.1: Categorization of the 15 temporal compliance rules.

## 5.1 Temporal Compliance Rules

Similar to control-flow compliance rules, we surveyed existing literature on [5, 68, 69, 73, 88, 149, 153] and collected and classified temporal compliance rules into a repository. Table 5.1 (also discussed in Chap. 3) gives an overview of these rules. In total, we collected 15 compliance rules distributed over *seven* categories. As we discussed in Chap. 2, we use data-aware Petri nets to formalize these rules. The complete list of the temporal rules and their formalization can be found in Appendix B.

From this repository (Table 5.1), the *Instance Duration* category of rules constrains the time period in which an instance of a control-flow rule must be executed. An example of this rule can be: “activity  $A$  must occur  $k$  times within  $[\alpha, \beta]$  time units since time  $t$ ”.

This rule already illustrates that each temporal rule has a control-flow aspect (“ $k$  occurrences of activity  $A$ ”) and a temporal aspect (“within  $[\alpha, \beta]$  time unit since time  $t$ ”). Even if the ordering of activities is not restricted in a temporal rule, at least the existence of some activities is specified.

The second category of temporal rules, *Delay Between Instances*, limits the time gap between two instances of a control-flow rule, e.g. the previous rule can be extended as: “activity  $A$  must occur in  $n$  cycles of  $k$  occurrences, and between subsequent cycles there must be  $[\alpha, \beta]$  time units delay”.

In the general case, the control-flow rule constrains more than just the existence of activities. For instance in the following rule taken from the *Repetition*

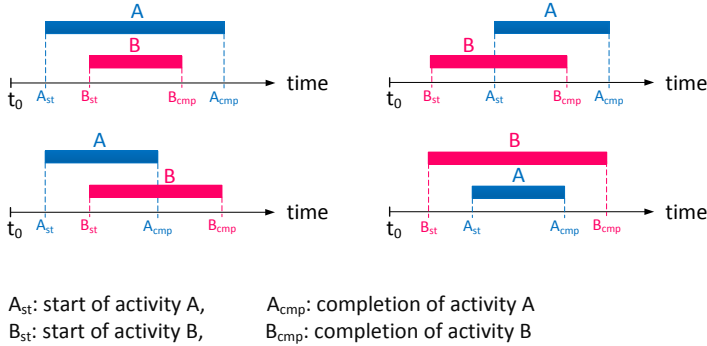


Figure 5.2: Executions of activities may overlap in different ways.

category of temporal rules: “the delay between the execution of two subsequent activities  $A$  and  $B$  must be within  $[\alpha, \beta]$  time units”, the underlying control-flow does not specify only the existence of activities  $A$  and  $B$  but also their sequence.

Form the remaining categories of temporal rules, the *Validity* category limits the time length in which an activity must or must not be executed. An example rule from this category can be: “activity  $A$  must be completed within  $[\alpha, \beta]$  time units since it starts”, or “no activity  $A$  may be executed within  $[\alpha, \beta]$  time units since time  $t$ ”.

The *Restricted Existence* limits the absolute execution time of activities. An example rule from this category is: “activity  $A$  may only be executed (or may not be executed) at times  $t_1, \dots, t_n$ ”. Temporal rules may also constrain the execution time of one activity depending on the execution time of another activity. Rules in the category *Time Dependent Variability* are of this type. An example of such rule is: “activity  $A$  must be executed within  $[\alpha, \beta]$  time units since time  $t_1$  if activity  $B$  is executed within  $[\gamma, \zeta]$  time units since time  $t_2$ ”. Similarly, the *Overlap* category of temporal rules also limits the start and completion of one activity  $A$  to start and completion of another activity  $B$ , and it gives rise to four different rules visualized in Fig. 5.2.

Reviewing different types of temporal compliance rules leads to the following working assumption: **Any temporal compliance rule can be separated into a control-flow aspect and a temporal aspect.** The control-flow aspect can be formalized as a control-flow compliance rule explained in Chap. 4. In the following, we focus on how to formalize the temporal aspect (in a data-aware Petri net) and how to combine two separate checks into one coherent

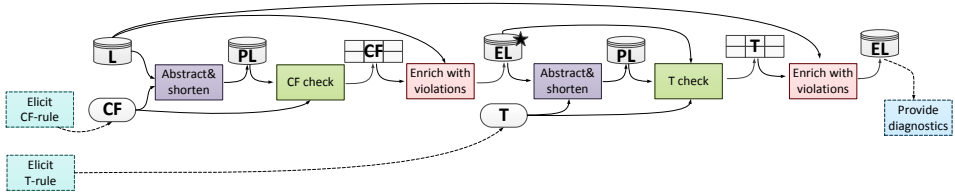


Figure 5.3: Overview of the temporal compliance checking methodology.

diagnostic.

## 5.2 Technicalities of Temporal Compliance Checking

As we discussed earlier, temporal compliance rules are built over control-flow restrictions. The dependency between control-flow rules and temporal rules raises a challenge for temporal compliance checking: we first have to identify the different occurrences of a control-flow rule, for which then the temporal rule can be checked. This gives rise to our approach for temporal compliance checking shown in Fig. 5.3. We decompose a temporal constraint into a control-flow and a temporal rule. The event log is first aligned to the control-flow rule using the control-flow checking technique described in Chap. 4. As a result, we obtain a control-flow alignment that indicates possible control-flow violations in terms of missing events or inserted events. This alignment also distinguishes several occurrences of a control-flow rule (recall the concept of *rule instance* from Chap. 3, and Chap. 4). This information about rule instances and which events are compliant to the control-flow (and hence contain reliable information) is then passed on to the temporal checking by enriching the event log with all diagnostic information from the control-flow alignment.

We model the temporal rule in terms of a data-aware Petri net. The temporal rule and the prepared log are used to build a data-aware alignment. The data-aware alignment then shows the temporal violations. To allow for root cause analysis on all violations of the rule, we again enrich the log, now with the diagnostics from the temporal checking step. In the following, we will go through the temporal compliance checking approach step by step by example.

**Example temporal constraint.** Suppose a compliance constraint *TR 1* stating:

“The treatment with antibiotics must be administered for three days with a delay of one day between each administration. After each cycle of three treatments, in case of necessity, the treatment can be extended for other cycles”.

L	time	1	2	30	54	100	123	162	173
	activity name	B	A	A	A	A	C	A	D
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>	e <sub>8</sub>

Figure 5.4: Example 1: Event log.

Assume we would like to check a given event log, partially shown in Fig. 5.4, against this constraint. This event log has eight events, all belong to the case  $p_1$ . The event with “*activity name=A*” denotes the execution of activity “antibiotic administration”. Other values of *activity name* represent execution of other activities in this case. According to *TR 1*, each cycle of treatment should contain three *antibiotic administration*. However, as shown in Fig. 5.4, two cycles of treatment is observed. One cycle contains only two *antibiotic administration*. In addition, we expect between two occurrence of *antibiotic administration* within a cycle to be 24 hours delay, however, some violations from this rule is observed. For example, the delay between the first two occurrences of *A* is more than 24 hours.

### Step 1 and 2: Log preparation and control-flow checking.

The atomic pattern shown in Fig. 5.5 models the underlying control-flow restriction of *TR 1*.

The rule instance of *TR 1* includes a cycle of three occurrences of activity ‘antibiotic administration’ (denoted as *A* in the pattern). The rule may occur several times (i.e., multiple occurrences of the cycle is possible). This pattern can be elicited using our elicitation technique discussed in Chap. 4 (see the corresponding configurable pattern shown in Fig. C.1).

Figure 5.6 shows the alignment of the example event log of Fig. 5.4 to the control-flow rule of Fig. 5.5. The alignment shows one instance of the rule with three occurrences of *A*, and one instance of the rule with two occurrences of activity *A*. The alignment denotes a missing event (model-only move) in the second instance of the rule. Note, how the structure of the atomic pattern in Fig. 5.5 enables us to explicitly mark two activations of the rule instance.

### Steps 3, and 4: Enrich log with control-flow violations, and prepare it for



**temporal checking.** At this point, we enrich the event log with diagnostics obtained during the control-flow checking. The log enrichment differs from the enrichment for pure control-flow checking presented in Sect. 4.1. We have to provide some additional information for temporal checking. The enriched event log is shown in Fig. 5.7.

During the log enrichment 1) we translate each move in the control-flow alignment to an event, 2) we insert missing events in positions indicated by the control-flow alignment. In our example, event *A* was missing in the second instance of the *TR 1* rule. Similarly, we insert the missing events *Start*, *I<sub>st</sub>*, *I<sub>cmp</sub>*, and *End*. 3) We introduce a new attribute named *CF condition*, this attribute records whether the control-flow condition holds at each event or not. Every event originating in a non-synchronous move is marked *False (F)* for its *CF condition* otherwise it gets the value *True (T)*. 4) Finally, events inserted into the log in steps 1 and 2, due to model-only move violations, do not have a timestamp. To enable temporal checking, we generate *time* attribute values for each such event. In particular, an event with a missing *time* attribute gets the *time* value of the directly preceding event except *Start*, and *I<sub>st</sub>* which get the *time* value of the succeeding event. The enriched event log now contains enough information to check the temporal constraint.

The example log does not need further preparation (i.e., abstraction and shortening). However, in some cases we require another round of log preparation. We will discuss this situation later in Sect. 5.5.

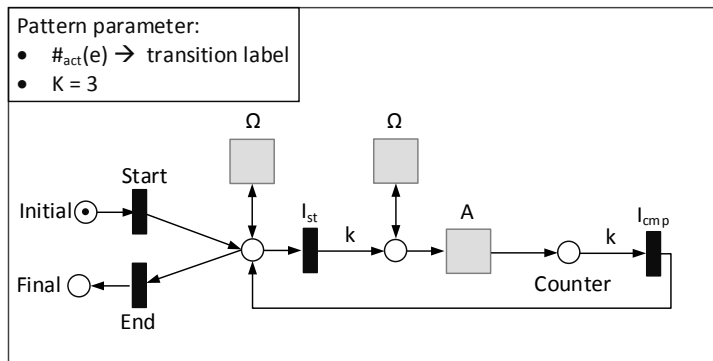


Figure 5.5: Example 1: Control-flow rule modeling cycles of three occurrences of activity antibiotic administration.

L	time		1		2	30	54			100	123		162		173	
	activity name		B		A	A	A			A	C		A		D	
	process instance	>>	p <sub>1</sub>	>>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	>>	>>	p <sub>1</sub>	p <sub>1</sub>	>>	p <sub>1</sub>	>>	p <sub>1</sub>	>>
	event ID		e <sub>1</sub>		e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>			e <sub>5</sub>	e <sub>6</sub>		e <sub>7</sub>		e <sub>8</sub>	
	checking		Ω		A	A	A			A	Ω		A		Ω	
M	transition name	Start	Ω	I <sub>st</sub>	A	A	A	I <sub>cmp</sub>	I <sub>st</sub>	A	Ω	A	A	I <sub>cmp</sub>	Ω	End

Figure 5.6: Alignment of the log of Fig. 5.4 to the control-flow rule of Fig. 5.5 showing a model-only move between e<sub>6</sub> and e<sub>7</sub>, i.e., once occurrence of A was missing in the second rule instance.

As explained, a “real” missing event <sup>1</sup> (e.g. the missing activity A in our example) inherits the time stamp of its direct preceding event. However, we can think of other ways to infer a time stamp for such an event. For instance, Rogge Solti et al. in [107] and [105] enrich a business process with performance information. To this end, they align an event log to a model. Each event in the event log is mapped to a transition. They use observed values extracted from the event log and infer the most likely stochastic model that explains the

<sup>1</sup>By the “real” missing events we mean violations that reflect the absence of a real activity within a process not I<sub>st</sub>, I<sub>cmp</sub>, Start, End or τ-labeled) events.

Project temporal information on missing events.

EL*	time	1	1	2	2	30	54	54	100	100	123	123	162	162	173	173
	activity name	-	B	-	A	A	A	-	-	A	C	-	A	-	D	-
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sup>R</sup> <sub>1</sub>	e <sub>1</sub>	e <sup>R</sup> <sub>2</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sup>R</sup> <sub>3</sub>	e <sup>R</sup> <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sup>R</sup> <sub>5</sub>	e <sub>7</sub>	e <sup>R</sup> <sub>6</sub>	e <sub>8</sub>	e <sup>R</sup> <sub>7</sub>
	CF condition	F	T	F	T	T	T	F	F	T	T	F	T	F	T	F
	checking	Start	Ω	I <sub>st</sub>	A	A	A	I <sub>cmp</sub>	I <sub>st</sub>	A	Ω	A	A	I <sub>cmp</sub>	Ω	End

This attribute records whether the control-flow condition holds or not.

Missing events are inserted.

Figure 5.7: Example 1, event log enriched with control-flow violations and temporal information.

observation best. We could adopt such an approach to build a stochastic model for the timestamp of each activity in the process to assign the most probable timestamp for a missing event when we observe that an activity is skipped in a case. However, the approach in [105, 107] is currently limited to acyclic models whereas our control-flow rules often contain cycles.

Several *imputation* methods are proposed in literature to insert artificial values for missing data [75]. A basic imputation method is *mean substitution*, i.e., replacing missing values of a variable with the sample mean of the observed values. More sophisticated imputation methods are *maximum likelihood estimation*, and *Bayesian multiple imputation*. These models make use of all observed data to increase confidence in the estimation. The technical details of these techniques are not in the focus of this thesis. The interested reader is referred to [75] for this topic.

**Step 5: Eliciting the temporal rule.**

The atomic pattern in Fig. 5.5 was used to specify the control-flow condition of *TR 1* rule. Now, we need to model the temporal dimension of the constraint: “delay between two occurrences of *A* must be *one day (24hrs)*”. We model this rule as a data-aware Petri net. Figure 5.8 models the temporal dimension of the rule.

We abstract the temporal rule to: “the delay between two subsequent executions of *A* in an instance of the control-flow rule, must be within  $[\alpha, \beta]$  time

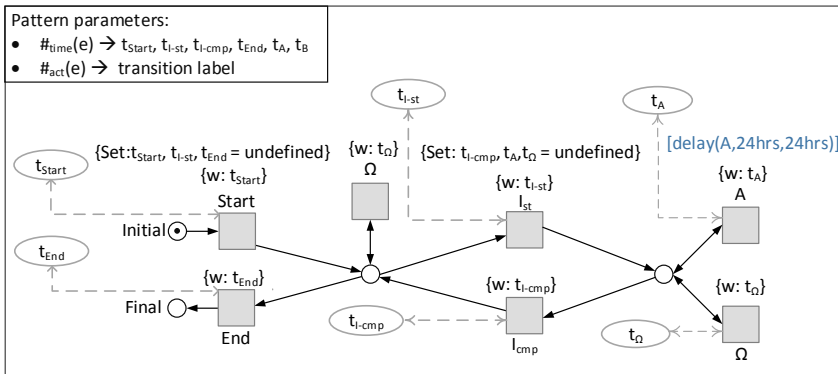


Figure 5.8: Example 1, the temporal dimension of the rule *TR 1* modeled as a data-aware Petri net.

units.” The atomic pattern of this rule (Fig. 5.8) has a very simple control-flow structure that just distinguishes whether the events are *within* an instance of a control-flow rule (i.e., after  $I_{st}$  occurred) or *outside* of it (i.e., after  $I_{cmp}$  occurred). Note that the transitions  $Start$ ,  $I_{st}$ ,  $I_{cmp}$ , and  $End$  are no longer modelled as *invisible* because they represent events in the enriched event log.

The temporal pattern shown in Fig. 5.8 has six variables  $t_A$ ,  $t_{I-st}$ ,  $t_{I-cmp}$ ,  $t_{Start}$ ,  $t_{End}$ , and  $t_\Omega$  that capture the values for attribute *time* at events  $A$ ,  $I_{st}$ ,  $I_{cmp}$ ,  $Start$ ,  $End$ , and  $\Omega$  respectively. For ease of representation, we only show the ellipse related to variable  $t_A$  in Fig. 5.8. Dashed arcs indicate that every occurrence of the transition  $A$  and  $I_{st}$  can update the value of variable  $t_A$ . Note that variables  $t_A$ ,  $t_{I-st}$ ,  $t_{I-cmp}$ ,  $t_{Start}$ ,  $t_{End}$ , and  $t_\Omega$  are all properties of the net that store the value for attribute timestamp of respective run events.

The actual temporal aspect is described by the data annotations at transition  $A$  and  $I_{st}$ . Annotation  $\{W : t_A\}$  ensures that  $t_A$  stores the timestamp of the run event  $A$ . The most important annotation is the guard  $[delay(A, \alpha, \beta)]$  defined by  $delay(A, \alpha, \beta) \equiv t'_A \in [t_A + \alpha, t_A + \beta] \vee t_A = \text{undefined}$ . The guard states that the updated value of the variable  $t_A$  (shown as  $t'_A$ ) has to be in the interval  $[t_A + \alpha, t_A + \beta]$ , where  $t_A$  is the read value of variable  $t_A$ .

As the rule only ranges over occurrences of  $A$  within the same instance of the control-flow rule, we have to take special care for the first  $A$  in an instance. Therefore, the second statement of the guard ( $t'_A = \text{undefined}$ ) ensures that the guard holds for first occurrences of  $A$  in each rule instance. The guard at  $I_{st}$  initializes  $t'_A = \text{undefined}$  so that the guard of  $A$  also holds for the first  $A$  in every instance.

By setting parameters  $A = \text{antibiotic administration}$  and  $\alpha = \beta = 24$  hours, the pattern of Fig. 5.8 formalizes the temporal rule.

**Step 6: Temporal checking.** We check compliance of the enriched log (shown in Fig. 5.7) against the temporal pattern. The data-aware alignment technique explained in Chap. 2 compares the time stamp of the events in the enriched log with admissible time stamps specified in the temporal pattern. Note that the enriched event log fits perfectly the control-flow structure of the temporal pattern. Hence, the data-aware alignment will not detect any control-flow violation. However, it detects deviations of observed attribute values, e.g. *time*, from specified values as *incorrect synchronous moves*. The data-aware alignment obtained from aligning the enriched event log and the temporal pattern is shown in Fig. 5.9, revealing three incorrect synchronous moves where the events occurred not at the specified time.

As is shown in the alignment the second  $A$  in the first instance occurred

EL*	time	1	1	2	2	30	54	54	100	100	123	123	162	162	173	173
	activity name	-	B	-	A	A	A	-	-	A	C	-	A	-	D	-
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sup>R</sup> <sub>1</sub>	e <sub>1</sub>	e <sup>R</sup> <sub>2</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sup>R</sup> <sub>3</sub>	e <sup>R</sup> <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sup>R</sup> <sub>5</sub>	e <sub>7</sub>	e <sup>R</sup> <sub>6</sub>	e <sub>8</sub>	e <sup>R</sup> <sub>7</sub>
	CF condition	F	T	F	T	T	T	F	F	T	T	F	T	F	T	F
	checking	Start	Ω	I <sub>st</sub>	A	A	A	I <sub>cmp</sub>	I <sub>st</sub>	A	Ω	A	A	I <sub>cmp</sub>	Ω	End
	M	transition name	Start	Ω	I <sub>st</sub>	A	A	A	I <sub>cmp</sub>	I <sub>st</sub>	A	Ω	A	A	I <sub>cmp</sub>	Ω
admissible time		1	1	2	2	26	54	54	100	100	123	124	148	162	173	173

Figure 5.9: Example 1: Data-aware alignment of the enriched event log of Fig. 5.7 to the temporal compliance pattern of Fig. 5.8.

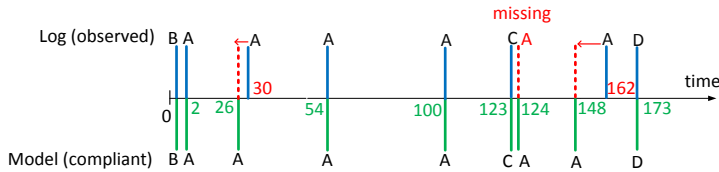


Figure 5.10: Projection of diagnostics on a timeline.

28 time units after its preceding A; violating the *TR 1* rule. The data-aware alignment returns the time at which the event should have occurred at the bottom row of the alignment. In the same way, two deviations in the second rule instance are highlighted. However, the “correct” time stamps 124, and 148 suggested by the alignment have to be inspected carefully. Note that in the second instance, the second A is missing in the log (a control-flow violation indicated by the value *False* for attribute *CF condition*). In Sect. 5.6, we investigate further how to interpret and fine tune these diagnostics. Next, we will explain how we enrich the event log with the obtained diagnostics.

**Step 7: Enrich log with temporal violations.** Finally, we enrich the original event log with all diagnostics we obtained including control-flow and temporal violations and other diagnostics such as: the type of each violation, the violating instance of a rule, and the values that would have led to a compliant execution. Figure 5.10 visualizes the diagnostics obtained on a timeline and Fig. 5.11 illus-

trates the enriched event log. Each event is annotated with two new attributes for the two rules (one on control-flow compliance and one on temporal compliance). In Fig. 5.11, these attributes are hachured. For instance, it is shown in Fig. 5.11 that the first event ( $e_1^L$ ) was evaluated against two rules: a control-flow rule (5 iterations of  $A$ ) and a temporal rule (delay between two  $A$ ). This event occurred outside of a rule instance for both rules and it was evaluated to compliant with both rules.

The third event ( $e_3^L$ ), however, is compliant w.r.t. the control-flow rule but it violated the temporal rule. It occurred inside the first activation of the temporal rule and it should have occurred at  $time = 26$  to be compliant. Event  $e_7^S$  violates both rules and has violations of type: *missing event*, and *temporal*. Note, the event  $e_7^S$  is inserted in the position where an execution of activity  $A$  was missing, all the other events originate from the log.



Figure 5.11: Enriched event log with control-flow and temporal diagnostics.

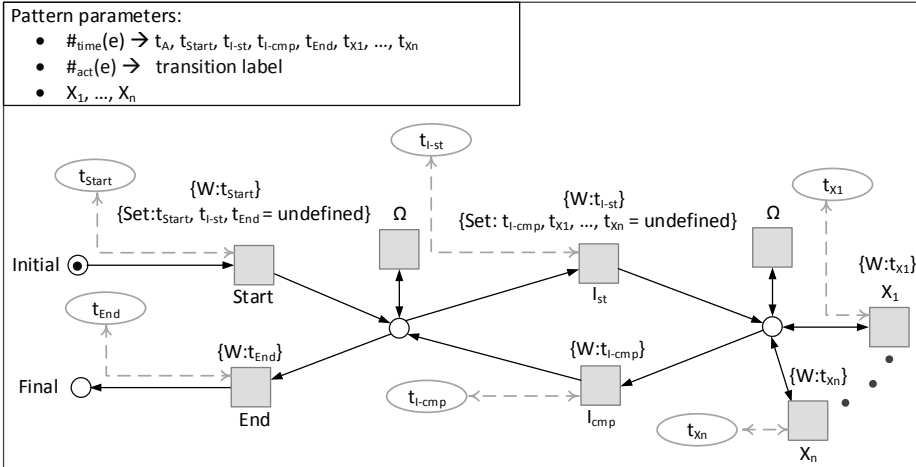


Figure 5.12: The generic temporal pattern.

### 5.3 A Generic Temporal Pattern

In Chap. 3 and in Sect. 5.2, we sketched the basic principles for formalizing temporal compliance rules in data-aware Petri nets. There, we showed that the temporal constraints can be expressed as guards on transitions that distinguish whether an activity occurs within an instance of the rule or outside a rule instance. We now use this insight and present a generic temporal compliance pattern that can be instantiated to formalize each of the 15 temporal compliance rules that we collected in Sect. 5.1.

The generic pattern is shown in Fig. 5.12. It permits to constrain occurrences of  $n$  activities  $X_1, \dots, X_n$ , as well as the *Start* and *End* of a process instance and start and end of each rule instance (by  $I_{st}$  and  $I_{cmp}$ ). For each activity  $X_i$ , the net has a variable  $t_{X_i}$  that gets updated with the value of the *time* attribute of any event of  $X_i$ , therefore,  $t_{X_i}$  always holds that last moment  $X_1$  was executed. Similarly the variables  $t_{start}$ ,  $t_{End}$ ,  $t_{I_{st}}$ , and  $t_{I_{cmp}}$  get updated by firing any of the transitions *Start*, *End*,  $I_{st}$ , and  $I_{cmp}$ . The transition *Start* initializes the value for  $t_{I_{st}}$ , and  $t_{I_{cmp}}$  and the transition  $I_{st}$  initializes the value for  $t_{X_i}$ . Each formalization of a compliance rule assigns a guard to one or more transitions of the pattern. Recall from previous section that all artificial events related to *Start*,



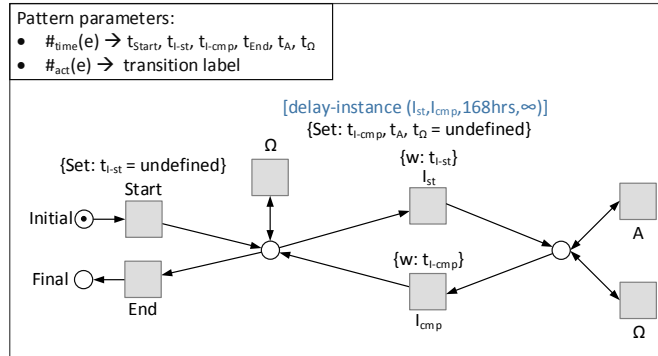


Figure 5.13: The generic temporal pattern specifying delay between rule instances.

$I_{st}, I_{cmp}$ , and  $End$  get valid timestamps.

In the following, we show how to instantiate this generic pattern for several rules. The complete list of rules and the instantiation of the generic temporal compliance pattern for each rule are explained in Appendix B.

**Delay between instances.** Consider the following variant of the compliance constraint *TR 1* from Sect. 5.2, “The treatment with antibiotics must be administered for three days with a delay of one day between each administration. After each cycle of three treatments, in case of necessity, the treatment can be extended for other cycles. The delay between cycles of treatment must be at least one week.”

This constraint has two parts  $R_1$ ) between two administration of antibiotics there should be a one day delay  $R_2$ ) between two cycles of treatment with antibiotic there should be a delay of at least one week.

The instantiated *generic temporal pattern* for the second temporal rule ( $R_2$ ) is given in Fig. 5.13. The overall structure is similar to the first pattern given in Fig. 5.8 as it distinguishes occurrences of  $A$  inside and outside of a rule instance. The actual temporal logic is specified in the guards.

In this pattern  $n = 1$  and the variables  $t_{I_{st}}$ , and  $t_{I_{cmp}}$  record the time values for  $I_{st}$ , and  $I_{cmp}$ . For ease of representation, we do not show the ellipses related to different variables and the write statements for each transition.

$R_2$  specifies a delay of at least one week (168 hours) between two occurrences of the control-flow rule instance (two treatments with three antibiotic administration). Therefore, the guard  $[delay-instance (I_{st}, I_{cmp}, \alpha, \beta)]$  annotat-

ing the transition  $I_{st}$  specifies the gap between the two cycles of treatment. The guard is defined by *delay-instance*  $(I_{st}, I_{cmp}, \alpha, \beta) \equiv t'_{I_{st}} \in [t_{I_{cmp}} + \alpha, t_{I_{cmp}} + \beta] \vee t_{I_{st}} = \text{undefined}$ . This guard states that the time  $t_{I_{st}}$  of the current occurrence of  $I_{st}$  has to be in the interval  $[t_{I_{cmp}} + \alpha, t_{I_{cmp}} + \beta]$ , where  $t_{I_{cmp}}$  is the time stamp of the most recent occurrence of  $I_{cmp}$ . For the first instance of the rule (i.e., first occurrence of the  $I_{st}$ ), the annotation at the transition *Start* initializes  $t_{I_{st}} = \text{undefined}$  so that the guard at transition  $I_{st}$  holds.

By setting parameters  $A = \text{antibiotic administration}$  and  $\alpha = 168$  hours, and  $\beta = \infty$ , the pattern of Fig. 5.13 formalizes the second temporal rule  $R_2$ . Note that  $R_2$  states that the delay between two cycles must be **at least** 168 hours. Hence, the interval  $[\alpha, \beta] = [168, \infty]$  does not have an upper bound.

**Delay between activities of different kind.** First temporal rule ( $R_1$ ) constrains delays between repeated occurrences of the same activity; Fig. 5.8 shows how to formalize such delays. Next, we show how to instantiate the generic pattern to constrain delays between different activities e.g. two given activities  $A$  and  $B$ .

In this case, the generic temporal pattern will be instantiated for  $n = 2$ . Therefore, the corresponding events having the value  $A$ , and  $B$  for their *checking* attribute will be mapped respectively to the  $X_1$ -labelled and  $X_2$ -labelled transitions in the temporal pattern. The guard *delay-diff-activities 1*  $(A, B, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_B\} \wedge t_B \neq \text{undefined}$ , is assigned to the  $X_1$ -labeled transition in Figure 5.12 to which  $A$  is mapped. Similarly the guard *delay-diff-activities 2*  $(A, B, t, \alpha, \beta) \equiv t'_B \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_A\} \wedge t_A \neq \text{undefined}$ , is assigned to the  $X_2$ -labelled transition to which  $B$  is mapped.

**Overlap.** We can extend the temporal rules that specify the delay between two activities to rules that limit start and completion of one activity  $A$  to start or completion of another activity  $B$ . For these temporal rules, the generic temporal pattern (Fig. 5.12), has ( $n = 4$ ) transitions. Therefore events  $A_{st}, A_{cmp}, B_{st}$  and  $B_{cmp}$  would be mapped respectively into  $X_1, \dots, X_4$ -labeled transitions in the pattern. Suppose a rule states: “activity  $B$  must start within  $[\alpha, \beta]$  time units after activity  $A$  starts and activity  $B$  must complete within  $[\gamma, \zeta]$  time units before activity  $A$  completes.” In this case, the rule has two guards: *start-after-complete-before 11*  $(A_{st}, B_{st}, t, \alpha, \beta) \equiv t'_{B_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{A_{st}}\} \wedge \{t_{A_{st}}\} \neq \text{undefined}$ . The first guard is assigned to transition  $X_3$  in the generic temporal pattern which is mapped to  $B_{st}$ .

The guard *start-after-complete-before 2*  $(A_{cmp}, B_{cmp}, t, \gamma, \zeta) \equiv t'_{A_{cmp}} \in M \wedge M = [t + \gamma, t + \zeta] \wedge t \in \{t_{B_{cmp}}\} \wedge \{t_{B_{cmp}}\} \neq \text{undefined}$  is assigned to the  $X_2$ -labeled transition in the generic temporal pattern to which  $A_{cmp}$  is mapped. Similarly other

rules in this category will be specified by assigning the guards to respective transitions.

The first guard and the second guard ensures that the execution of activity  $B$  overlaps with execution of activity  $A$ , i.e.,  $B$  is executed during execution of  $A$ . See Fig. 5.2 (top-left) for this situation. The first guard ensure that activity  $B$  is allowed to start after  $A$  already started and  $B$  should complete before  $A$  completes.

**Pattern duration.** Some compliance rules may limit the time length in which a control-flow rule instance may be executed. For instance, the temporal rule *negation pattern duration* states: “no instance of a control-flow rule must be executed within  $[\alpha, \beta]$  time units since time  $t$ ”. To specify these type of rules, we instantiate the generic temporal rule similar to previous examples and we annotate  $I_{cmp}$  to specify the duration of an instance. For example, in case of the *negation pattern duration*, the guard *negation-pattern-duration*  $(t, \alpha, \beta)$  will be assigned to  $I_{cmp}$ . This guard is defined by *negation-pattern-duration*  $(t, \alpha, \beta) \equiv t'_{I_{cmp}} \in M \wedge M = [0, \infty) \setminus [t + \alpha, t + \beta]$  where  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{case}, t_{calendar}, t_{X_{i_{st}}}, t_{X_{i_{cmp}}}\}$ .

This guard ensures that  $I_{cmp}$  occurs  $[\alpha, \beta]$  after time  $t$  as a reference. This reference time can be start of the rule instance ( $t_{I_{st}}$ ), start of the case ( $t_{case}$ ), a specific time in calendar ( $t_{calendar}$ ) or start/completion of an activity in the process ( $t_{X_{i_{st}}}, t_{X_{i_{cmp}}}$ ).

**Validity.** These group of rules limits the time during which an activity execution must complete. To specify these type of rules, the generic temporal pattern is instantiated for  $n = 1$  or  $n = 2$  depending on whether the activity execution is recorded as an atomic event or by events with *start* and *complete* life-cycle events. For instance, for the rule *activity duration*, we map an event  $A_{st}$  (start of activity  $A$ ) to  $X_1$ -labelled transition and  $A_{cmp}$  (completion of activity  $A$ ) to  $X_2$ -labelled transition. The guard *activity duration* then is defined by *activity-duration*  $(A_{st}, A_{cmp}, t_{A_{st}}, \alpha, \beta) \equiv t'_{A_{cmp}} \in M \wedge M = [t_{A_{st}} + \alpha, t_{A_{st}} + \beta]$  and is assigned to the  $X_2$ -labelled transition where  $A_{cmp}$  is mapped.

For the rules that specify execution time of an atomic activity  $A$ , we instantiate the pattern for  $n = 1$  and map the  $X_1$ -labelled transition to the event  $A$ . We assign to the  $X_1$ -labelled transition the guard *activity-execution*, defined by *activity-execution*  $(A, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [t + \alpha, t + \beta]$ . The reference time  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{case}, t_{calendar}, t_{X_i}\}$ . Note that if  $t = t_{X_i}$ , then the generic temporal pattern has to be instantiated with a second transition  $X_2$  ( $n = 2$ ) to allow tracking the occurrence times  $t_{X_2}$  of  $X_2$ .

**Time Dependent Variability.** Some temporal rules specify the execution of an activity or in general the control-flow of the process depending to the execution of another activity. For example “a given activity  $B$  must be executed within  $[\alpha, \beta]$  time units since time  $t^1$  if  $A$  has occurred within  $[\gamma, \zeta]$  time units since time  $t^2$ ”.

For this temporal rule, the generic temporal pattern has  $n = 2$  transitions. Therefore the events  $A$  and  $B$  are respectively mapped to the  $X_1$ -labeled and  $X_2$ -labeled transitions in the pattern.

The guard, *time-dependent-variability*  $(A, B, t^1, t^2, \alpha, \beta, \gamma, \zeta) \equiv t'_B \in M \wedge M = [t^1 + \alpha, t^1 + \beta] \wedge t'_A \in [t^2 + \gamma, t^2 + \zeta]$ , is assigned to the  $X_2$ -labeled transition in the generic temporal pattern to which  $B$  is mapped. The user chooses  $t^1$  and  $t^2$  from  $t^1 \in \{t_{I_{st}}, t_{I_{cmp}}, t_{Start}, t_{Calendar}, t_{X_i}\}$  and  $t^2 \in \{t_{I_{st}}, t_{I_{cmp}}, t_{Start}, t_{Calendar}, t_{X_i}\}$ . Note that if  $t^1 \in \{t_{X_i}\}$ , or  $t^2 \in \{t_{X_i}\}$  then the generic temporal pattern in Fig. 5.12 will have more than 1 transitions, where  $X_i$  transition is mapped to the additional transition.

Similar to the examples we mentioned above, all other temporal compliance constraints identified in literature can be formalized by instantiating the generic temporal pattern of Fig. 5.12, see Appendix B for details. Each formalization is then eligible for temporal compliance checking using data-aware alignments. Our temporal compliance checking technique is not limited to pre-defined control-flow rules and temporal rules, but is extendible. Next, we show how we can adapt a generic temporal compliance rule for compound and complex temporal restrictions.

## 5.4 Specifying a Set of Temporal Rules as Atomic Temporal Patterns or A Composite Compliance Model

As is discussed by the examples above, the concept of *rule instance* is essential in formalizing temporal rules. If we can specify the boundaries of a compliance rule, then we can specify any constraint on temporal dimension, even a combination of temporal rules.

Recall the example of ‘antibiotic administration’. The constraint has two temporal rules:  $R_1$ ) Between two antibiotic administration there must be 24 hours delay, and  $R_2$ ) between two cycles of antibiotic treatment there must be 168 hours delay. The underlying control-flow condition of both of these temporal rules is specified in the atomic pattern shown in Fig. 5.5 which explicitly

defines the *rule instance* through the  $I_{st}$  and  $I_{cmp}$  transitions. As both temporal rules use the same notion of rule instance, we can combine both rules together and instantiate the generic temporal pattern so that it specifies the temporal aspect of both rules together; the resulting pattern is shown in Fig. 5.14. For ease of presentation, we do not show the ellipses related to different variables and the write statements at each transition. As can be seen, we can constrain the delay between the two cycles of treatment by the guard *delay-instance* for transition  $I_{st}$ , and we constrain the delay between two administrations of antibiotic inside a cycle by the guard *delay* for transition  $A$ . By combining the two rules in one composite model, we can get diagnostics about compliance of the two rules combined in one data-aware alignment.

We may only express two temporal rules in the same temporal pattern if both rules have the same notion of rule instance. Suppose two temporal rules stating: *rule 1*) “activity  $D$  must be followed directly by activity  $A$  within two time units”, and *rule 2*) “activity  $B$  may only occur if activity  $A$  has occurred one time unit before it”. In this case we cannot define a single rule instance. Suppose the sequence of activities  $\langle D, A, A, A, B \rangle$  (as is shown in Fig. 5.15) have been executed. As can be seen the first  $A$  is part of the rule instance of the

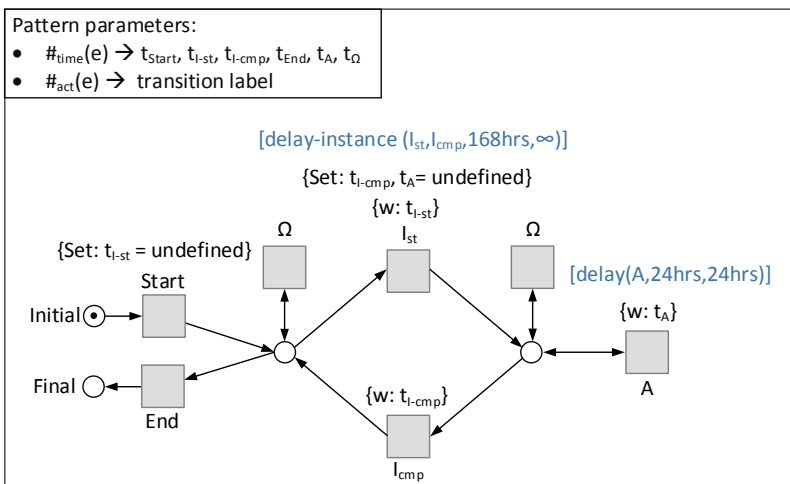


Figure 5.14: Both  $R_1$  and  $R_1$  modelled together as a composite model by instantiating the generic temporal pattern.

## 5.4 Specifying a Set of Temporal Rules as Atomic Temporal Patterns or A Composite Compliance Model

first rule, and the third *A* is part of the rule instance of the second rule. We cannot simply use the generic temporal pattern 5.12 to specify both temporal rules together in one pattern.

Assume we define a rule instance such that it contains all the three activities *A*, *B*, and *D*. The instantiated generic temporal pattern for this setting is shown in Fig. 5.16. The pattern for  $n = 3$  has three transitions inside the rule instance that will be mapped to events *A*, *B*, and *D*. The guard of transition *A* specifies the delay between *D* and *A*, and the guard at transition *B* specifies the

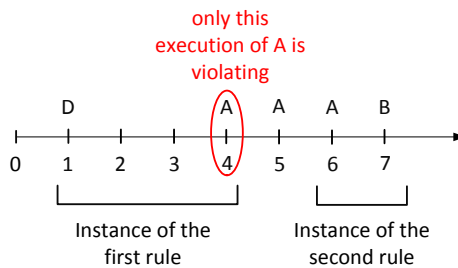


Figure 5.15: Sequence of activities  $\langle D, A, A, A, B \rangle$ .

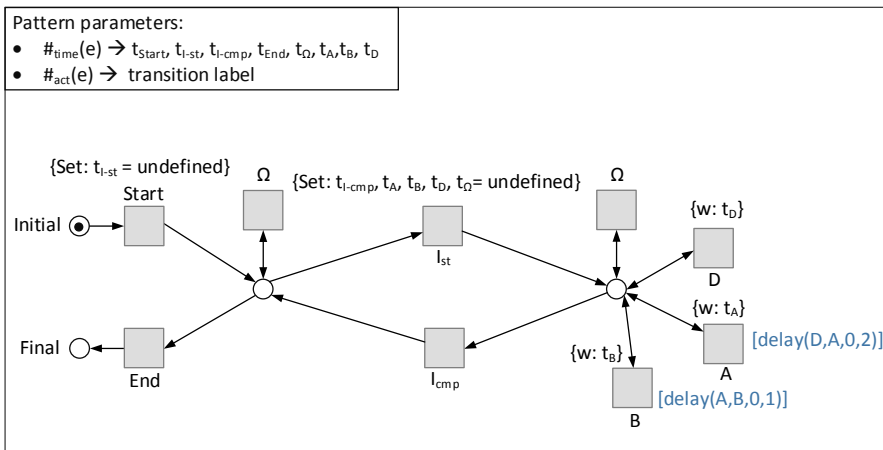


Figure 5.16: A composite model for specifying rules *rule 1* and *rule 2* together.

delay between  $A$  and  $B$ . With this formalization, all three occurrences of  $A$  will be evaluated with the guard annotating this transition and consequently all the three occurrences of  $A$  in our example sequence (Fig. 5.15) will be detected as violating whereas only the first  $A$  is violating the first temporal rule. That is, because we could not differentiate between the rule instances, all executions of  $A$  are evaluated with the same guard. Therefore, the generic temporal pattern must be changed to better reflect the underlying control-flow structure. Nevertheless, although it is possible to express some temporal rules together by the generic temporal pattern, there is no generic way for all possible combination of temporal rules.

Section 5.1 to Sect. 5.4 were mainly about understanding temporal compliance rules and their formalization. We showed how we can formalize various atomic temporal rules using the generic temporal pattern. We explained our temporal checking approach step by step using several examples. Next, we will discuss the checking approach further by explaining log abstraction as a preparation step for checking. In addition, we will investigate the quality of diagnostics obtained using temporal compliance checking.

## 5.5 Log Abstraction for Temporal Checking

In Sect. 5.2, we introduced the log abstraction and log enrichment steps to enable temporal compliance checking. While this technique works in general, it makes the assumption that no instance of a specified activity occurs outside rule instances. In this section, we discuss a case that violates this assumption and show how to generalize log abstraction for all cases.

Suppose a compliance rule stating, *rule 3*: “every activity  $A$  must be followed by activity  $B$  within *two* time units.” We would like to check whether the given log shown in Fig. 5.17 adheres to this rule or not.

**Steps 1, 2, and 3: Log preparation, control-flow checking, and log enrich-**

L	time	1	2	4	7	8	9	10	11
	activity name	B	B	A	B	B	C	A	B
	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$e_8$

Figure 5.17: Rule 3, Event log.

ment. The atomic pattern that describes the control-flow condition of the rule

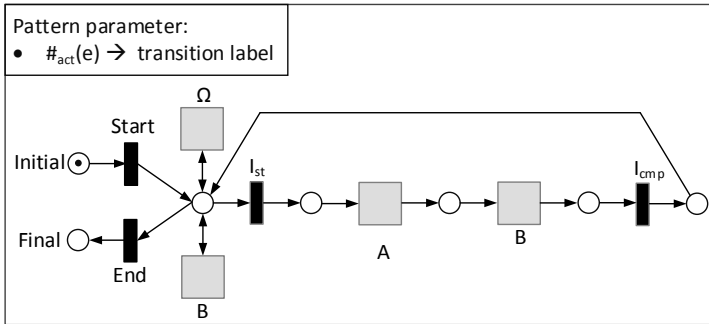


Figure 5.18: Rule 3- Atomic pattern describing the control-flow rule instance.

L	time		1	2		4	7		8	9		10	11		
	activity name		B	B		A	B		B	C		A	B		
	process instance	>>	p <sub>1</sub>	p <sub>1</sub>	>>	p <sub>1</sub>	p <sub>1</sub>	>>	p <sub>1</sub>	p <sub>1</sub>	>>	p <sub>1</sub>	p <sub>1</sub>	>>	>>
	event ID		e <sub>1</sub>	e <sub>2</sub>		e <sub>3</sub>	e <sub>4</sub>		e <sub>5</sub>	e <sub>6</sub>		e <sub>7</sub>	e <sub>8</sub>		
	checking		B	B		A	B		B	Ω		A	B		
M	transition name	Start	B	B	I <sub>st</sub>	A	B	I <sub>cmp</sub>	B	Ω	I <sub>st</sub>	A	B	I <sub>cmp</sub>	End

↑ ↑ ↑  
occurrences of activity B outside rule instances

Figure 5.19: Rule 3- Control-flow alignment of the log of Fig. 5.17 to the control-flow rule of Fig. 5.18.

EL	time	1	1	2	4	4	7	7	8	9	10	10	11	11	11
	activity name	-	B	B	-	A	B	-	B	C	-	A	B	-	-
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sup>R</sup> <sub>1</sub>	e <sub>1</sub>	e <sub>2</sub>	e <sup>R</sup> <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sup>R</sup> <sub>3</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sup>R</sup> <sub>4</sub>	e <sub>7</sub>	e <sub>8</sub>	e <sup>R</sup> <sub>5</sub>	e <sup>R</sup> <sub>6</sub>
	CF condition	F	T	T	F	T	T	F	T	T	F	T	T	F	F
	checking	Start	B	B	I <sub>st</sub>	A	B	I <sub>cmp</sub>	B	Ω	I <sub>st</sub>	A	B	I <sub>cmp</sub>	End

Figure 5.20: Rule 3- Enriched event log with diagnostics obtained from control-flow alignment in Fig. 5.19.



is shown in Fig. 5.18. After abstracting the event log, we check it against the control-flow rule and obtain the control-flow alignment shown in Fig. 5.19. The log fits the pattern and therefore no control-flow violation is detected. Note that in this alignment, some occurrences of activity *B* fall outside rule instances. The enriched event log with diagnostics are shown in Fig. 5.20. Note that in the previous example discussed in Sect. 5.2, all instances of specified activities were inside rule instances.

**Step 4: Preparing enriched event log for temporal checking.** We discussed previously that the *generic temporal pattern* is used to specify the temporal aspect of a compliance rule, i.e., it only allows to detect temporal violations. Therefore, any log used for temporal checking should fit the control-flow structure of the temporal pattern. Hence, we should prepare the event log enriched with control-flow diagnostics to fit perfectly the control-flow structure of the temporal pattern, i.e., the alignment shall only show temporal violations, but no control-flow violations.

To this end, we abstract from any activity outside rule instances and mark them as  $\Omega$  at their *checking* attribute. As a result we will obtain the prepared log shown in Fig. 5.21. The two executions of activity *B* that are outside rule instances are abstracted and get value  $\Omega$  for their consistency attribute *checking*.

This discussion emphasizes once more the importance of the *rule instance* concept in our approach for temporal compliance checking. That is, in temporal checking we only focus on activities that occurred inside a rule instance.

EL	time	1	1	2	4	4	7	7	8	9	10	10	11	11	11
	activity name	-	B	B	-	A	B	-	B	C	-	A	B	-	-
	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_1^R$	$e_1$	$e_2$	$e_2^R$	$e_3$	$e_4$	$e_3^R$	$e_5$	$e_6$	$e_4^R$	$e_7$	$e_8$	$e_5^R$	$e_6^R$
	CF condition	F	T	T	F	T	T	F	T	T	F	T	T	F	F
	checking	Start	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	End



  
 occurrences of activity B outside rule instances get  
 the value  $\Omega$  for their *checking* attribute

Figure 5.21: Rule 3- Preparing the event log of Fig. 5.20 for temporal compliance checking by abstracting from activities outside rule instances.

**Steps 6: Instantiating the generic temporal pattern.** The temporal rule of this example specifies the delay between two activities of different kind and was discussed in Sect. 5.3. The instantiated temporal pattern for this rule is shown in Fig. 5.22. The guard  $[delay(A,B,0,2)]$  at the transition  $B$  limits the execution time of this activity to be at most two time units after the execution

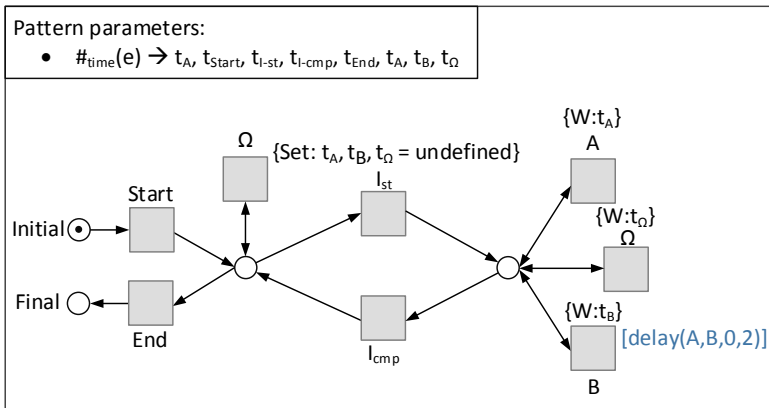


Figure 5.22: Rule 3- Temporal pattern instantiated to specify the delay between activities A and B.

	time	1	1	2	4	4	7	7	8	9	10	10	11	11	11
	activity name	-	B	B	-	A	B	-	B	C	-	A	B	-	-
PL	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_{-1}^R$	$e_1$	$e_2$	$e_{-2}^R$	$e_3$	$e_4$	$e_{-3}^R$	$e_5$	$e_6$	$e_{-4}^R$	$e_7$	$e_8$	$e_{-5}^R$	$e_{-6}^R$
	CF condition	F	T	T	F	T	T	F	T	T	F	T	T	F	F
	checking	Start	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	End
M	transition name	Start	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	$\Omega$	$\Omega$	$I_{st}$	A	B	$I_{cmp}$	End
	admissible time	1	1	2	2	4	6	7	8	9	10	10	11	11	11

↑ activity B occurred later than specified
 ↑ the prepared log fits the control-flow structure of the temporal pattern perfectly

Figure 5.23: Rule 3- Data-aware alignment of prepared log of Fig. 5.21 with the temporal pattern of Fig. 5.22 indicates a temporal violation.

of activity A.

**Step 7: Temporal checking.** Aligning the prepared log and the temporal pattern, we obtain the data-aware alignment shown in Fig. 5.23. The alignment indicates that the activity B in the first instance of the rule occurred later than it should. As the next step, we can enrich this event log with the temporal diagnostics obtained.

**Step 8: Log enrichment with temporal violations.** This step is done as similar as for the previous examples. The result of log enrichment is shown in Fig. 5.24.

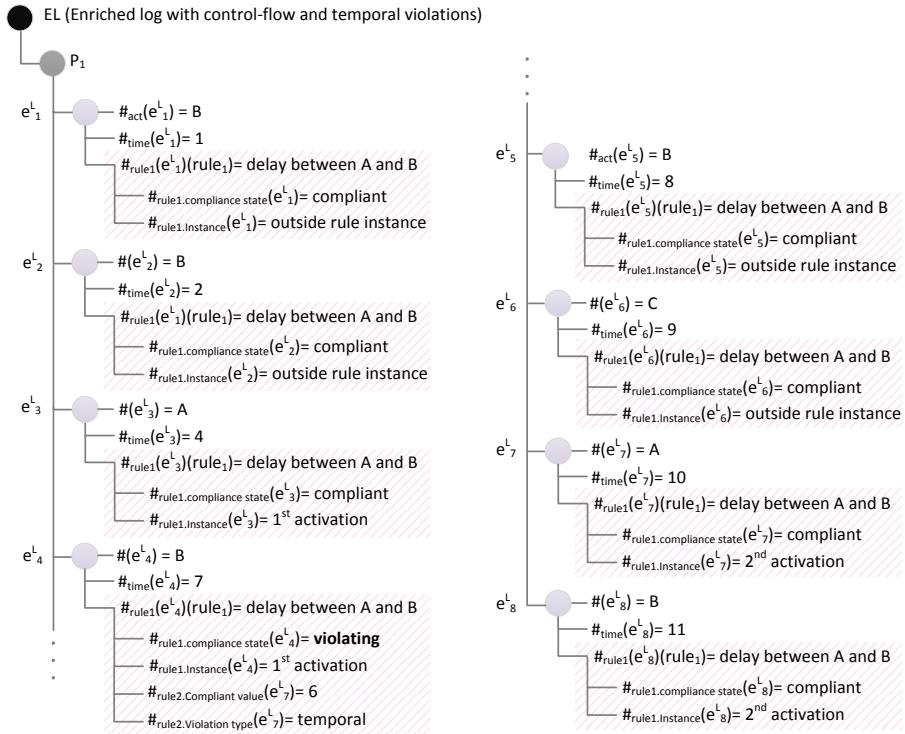


Figure 5.24: Rule 3- Enriched event log with complete diagnostics obtained from data-aware alignment of Fig. 5.23.

## 5.6 Diagnostics in Temporal Checking

So far, we assumed an oracle that will produce the data-aware alignment to provide us with diagnostic information. In the following, we discuss the parameters of a specific technique, a multi-perspective conformance checking approach [33], and how these influence the diagnostic information.

### Minimizing the total cost of violations globally

The technique of [33] builds a suitable Integer Linear Programme (ILP) to minimize the total cost of deviations between an observed process instance and a possible process instance by a model. Using this approach we are able to provide diagnostics that consider the global impact of the proposed compensation in the process rather than compensate the deviation locally. We will elaborate on this by an example.

Suppose a temporal constraint stating: “activity *a* must be followed within two time units by activity *B* and activity *C* must occur later than one time unit after *B*”, i.e.,  $t'_B \leq t_A + 2$  and  $t'_C > t_B + 1$ . The constraints are visualized in the top part of Fig. 5.25. Assume activities *A*, *B*, and *C* occurred as given in the bottom part of Fig. 5.25.

As can be seen, activity *B* is violating the constraint at transition *B*; it occurred later than specified. The technique of [33] (which our temporal compliance checking is based on) will indicate the execution of activity *B* as violating. In addition it suggests that activity *B* must occur at time 2 to compensate the violation. Time 2 is the value that satisfies both constraints together. When

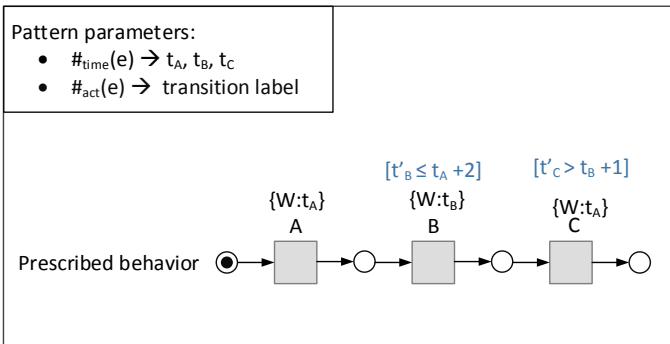


Figure 5.25: The data-aware alignment will suggest that activity *B* must occur at time 2.

the constraint  $t_B \leq t_A + 2$  is considered in isolation, then  $t_B$  may occur in the interval  $[1, 3]$  to be compliant. However, value 3 violates the second constraint  $t_C > t_B + 1$ . By considering all constraints together when searching for compliant values, the approach of [33] returns a value that satisfies all constraints (unless no such solution exists).

If we merely compare the execution time of activity  $B$  with the time that is specified in the prescribed behavior, the activity  $B$  may occur sometime in the  $[1, 3]$  interval. However, if  $B$  occurs at time 3, which is correct considering merely the constraint at transition  $B$ , then execution of activity  $C$  will be violating. Our temporal compliance checking technique considers the impact of suggested compliant value for activity  $B$  globally and will limit the compliant value to be only 2. Thereby, minimizing the number of violations.

### Minimizing the total value of deviation

In addition to minimizing the number of violations, the compliant value suggested by temporal checking minimizes the total value of deviation as well. We will also elaborate this point with the help of an example. Suppose a temporal constraint says that there should be a delay of *three* hours between executions of activities  $A$ ,  $B$ , and  $C$ . Assume the sequence of activities  $\langle A, B, C \rangle$  are executed as shown in the top part of Fig. 5.26. Apparently, this sequence is violating the constraint as there is not *three* hours delay between execution of these activities.

Among the many alignments that can be built, there are three compensation explanations shown in Fig 5.26 that are correct according to the constraint and will minimize the total number of violations. That is, if we assign a cost of *one* to each violation, these solutions all have the same total cost of *two*. The first explanation, suggests that activity  $B$  and  $C$  are violating and they should have been executed respectively at 15 : 00, and 18 : 00. If we take this explanation, then the total deviation of observed behavior from the prescribed behavior is *five* hours. The second explanation indicates that executions of activities  $A$  and  $C$  are violating and they should have happened respectively at 10 : 00, and 16 : 00. If this explanation is chosen, then the total deviation of the observed behavior from the prescribed behavior is *three* hours. Similarly the third explanation detects activities  $A$ , and  $B$  as violating and suggests they should have happened respectively at 09 : 00, and 12 : 00. This explanation has the total violation of *four* hours. In this situation, the technique of [33] will suggest the second explanation as the compliant values for activities  $A$  and  $B$ .

**Choice of the best control-flow alignment for temporal checking** As our approach uses control-flow checking to identify rule instances and control-flow violations prior to temporal analysis, we now discuss how the quality of the

control-flow alignment impacts the temporal analysis and the overall diagnostics.

We discussed in Chap. 3, and Chap. 4 that several control-flow alignments may have the same total cost. In this situation, the control-flow compliance checking technique picks an alignment randomly from the variants of an alignment with the same cost. However, if variants of a control-flow alignment have the same cost, the total cost of temporal violations may vary depending on which of the control-flow alignment variants are chosen for temporal checking.

In our example of ‘antibiotic treatment’ (presented in Sect. 5.2) several control-flow alignments have the same cost as the alignment shown in Fig. 5.6. For instance, the violation (missing event *A*) in this alignment (for now we call it ‘preferred alignment’) is detected in the second rule instance. Suppose the cost of this violation (missing activity *A*) is *one*. In this case, another control-flow alignment (we call it ‘not-preferred alignment’) shown in Fig. 5.27 have the same cost as the ‘preferred alignment’, while this alignment positions the violation in the first rule instance. Although both control-flow alignments have an equal cost *one*, they will result to different total cost (control-flow and temporal) if we also consider the cost of temporal violations w.r.t. the temporal rule

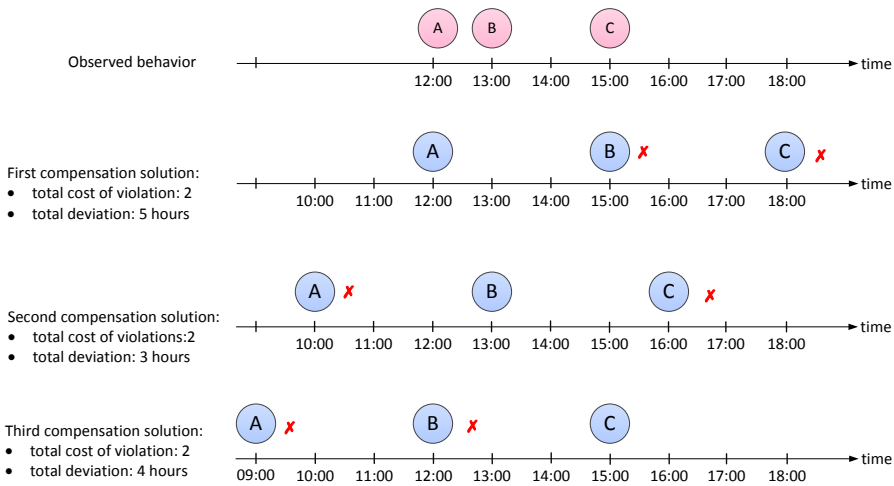


Figure 5.26: The temporal compliance checking minimizes the total deviation of observed behavior from prescribed behavior.

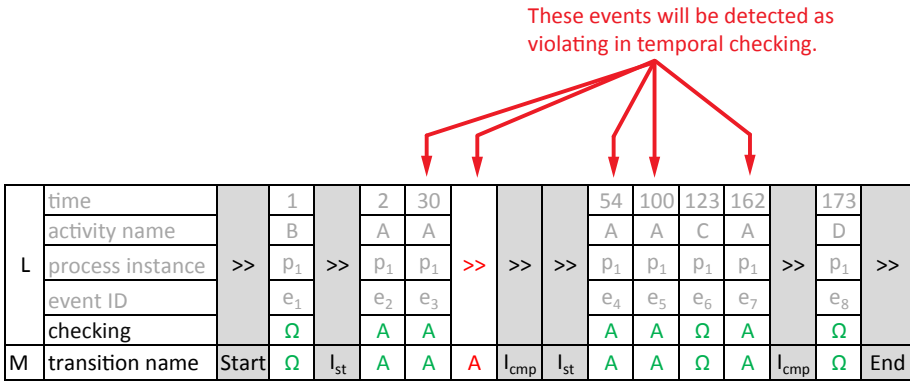


Figure 5.27: The ‘Not-preferred’ control-flow alignment will lead to more temporal violations.

$R_1$  and  $R_2$ . In this case, the ‘not-preferred alignment’ will lead to higher total cost of control-flow and temporal violations together. As indicated in Fig. 5.27, more events will be detected as violating (having temporal violations). Apart from the missing activity  $A$ , the event  $e_4$  will be violating because it should have happened at least 168 hours after the previous  $A$ . The events  $e_3$ ,  $e_4$ ,  $e_5$ , and  $e_7$  will be detected as violating because according to the rule there should be a delay of 24 hours between these events and their preceding execution of activity  $A$ .

To overcome this problem, our technique first computes all variants of a control-flow alignment having the lowest control-flow cost. Then we create for each control-flow alignment a new enriched log. We then compute a temporal alignment for each of these logs and then choose the temporal alignment with the lowest cost. All other temporal alignments are discarded.

**Balanced consideration of control-flow and temporal violations**

By building temporal checking upon control-flow checking, we are giving a priority to the control-flow perspective of a process. Therefore, the diagnostics we receive highly depends on the sequence of first control-flow alignment followed by data-aware alignment. Recall the example of activities  $A$ ,  $B$ , and  $C$  in Fig. 5.26. The constraint states that there must be a delay of *three* hours between execution of these activities. All the three compliant solutions shown are based on a control-flow alignment with cost 0, i.e., no control-flow violation. However, another way to explain the violation in the observed behavior would

be that activity *B* should have not occurred. In this case, the delay between two activities *A*, and *C* is *three* hours, i.e., no temporal violation happened but activity *B* should have not occurred.

If cost of any violation (control-flow and temporal) is 1 or *temporal* violations have a higher cost than *control-flow* violations, then the explanation (activity *B* should have not occurred) will have a lower total cost compared to the previous three explanations shown in Fig. 5.26.

The technique in [44] balances violations w.r.t. *all* process perspectives including control-flow and time based on a customizable cost function. However, using this technique we will loose the generality of our temporal checking approach that is based on the generic temporal pattern and is tailored for temporal checking. Recall from Sect. 5.2 that we combine control-flow checking and data-aware checking in two steps: the control-flow checking for clearly defining the boundary of rule instances and the data-aware checking for detecting temporal violations. The clear boundary of rule instances allows us to prepare the log such that it fits perfectly the control-flow of the generic temporal pattern and focus only on temporal violations. However, using the technique in [44], we will loose the clear distinction between the two steps and hence, we will not be able to use the generic temporal pattern. Consequently, the user should prepare a data-aware Petri net for each rule to be checked. In addition, domain knowledge is required to adjust the cost function.

## 5.7 Applying Temporal Compliance Checking on Real-Life Event Logs

Our temporal compliance checking technique has been implemented in ProM, available from [www.promtools.org](http://www.promtools.org), and was applied in a case study on real-life logs. We briefly discuss the implementation in ProM and then provide details on the case study.

### 5.7.1 Implementation in ProM

The temporal compliance checker is available in the package *Compliance* that provides *two* user-friendly plugins for temporal compliance checking. The first plugin provides control-flow compliance checking as described in the previous chapter. The control-flow compliance checking plugin takes as input a log and an atomic pattern and returns compliance diagnostics in form of a control-



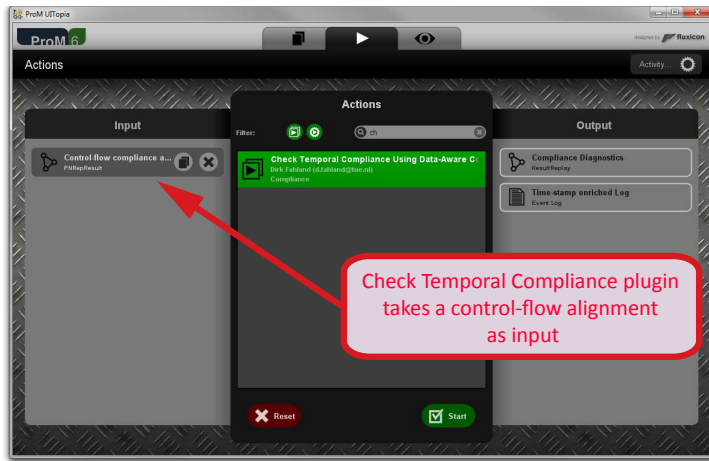


Figure 5.28: Temporal compliance checking using data-aware alignment plugin takes a control-flow alignment as input.

flow alignment. The control-flow atomic pattern can be built using the elicitation technique described in the previous chapter. The second plugin takes the control-flow alignment (See Fig. 5.28), produces an enriched log and then checks temporal compliance of the log to a temporal rule that can be specified by the user through a wizard. The resulting alignment then provides diagnostic information by showing control-flow compliance violations and temporal violations projected into the events of the original log. Figures 5.29, 5.30, 5.31, 5.32, 5.33, 5.34 show several screenshots of different wizards in *temporal compliance checking using data-aware alignment* plugin.

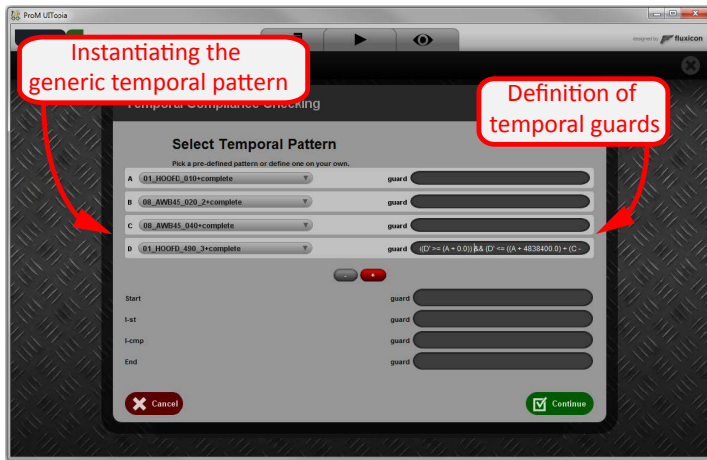


Figure 5.29: Instantiating the generic temporal pattern and definition of guards.

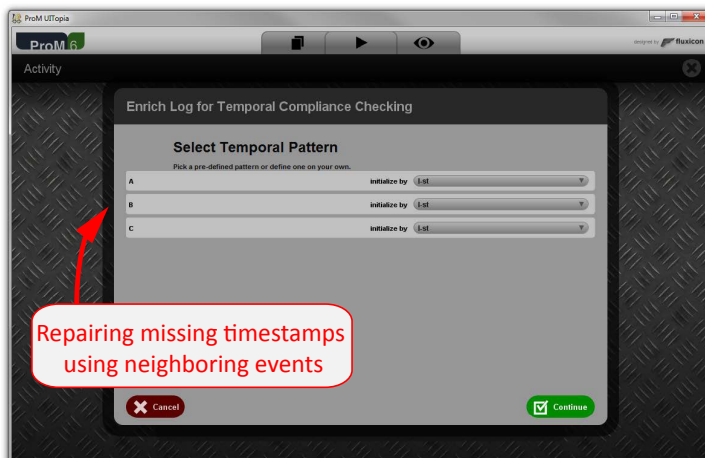


Figure 5.30: During temporal compliance checking, we allow the user to specify how the missing temporal values should be repaired.

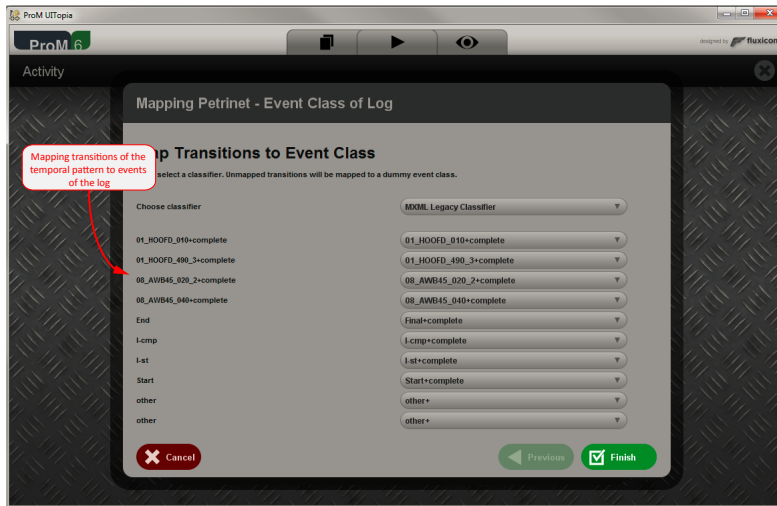


Figure 5.31: Mapping transitions of the generic temporal pattern to events of the log.

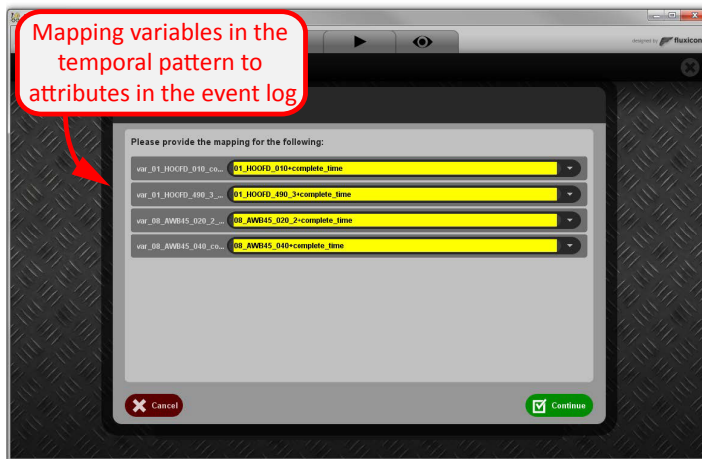


Figure 5.32: Mapping variables in the temporal pattern to attributes in the log.



Figure 5.33: Setting costs for different temporal violations.

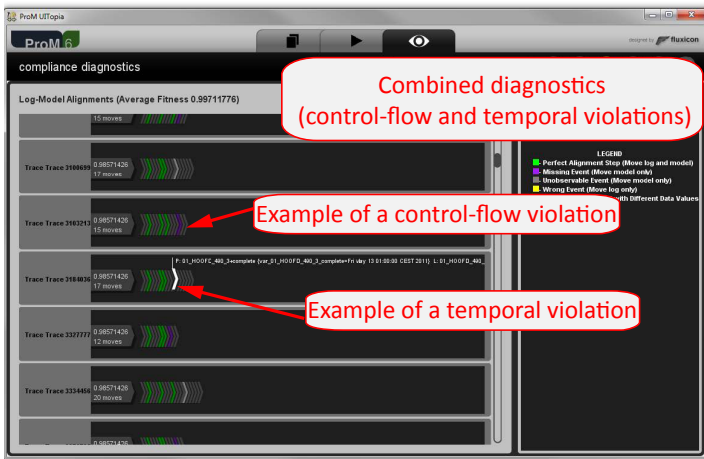


Figure 5.34: Temporal compliance checking provides combined diagnostics including control-flow and temporal violations.

## 5.7.2 Case Study Compliance Constraints and Results

We applied this implementation of the compliance checker in a case study for checking compliance of the building permit process of five Dutch municipalities ( $M_1, \dots, M_5$ ). The municipalities may carry out the building permit process in different ways, as long as it is compliant to the regulations issued by the Dutch legislative.

To test the feasibility of our temporal compliance checking technique, we selected a rather involved temporal compliance constraint that combines static and dynamic temporal aspects.

The compliance constraint was given informally: “Every application must be processed within at most 8 weeks from the date of a submitted request. If during the processing of the request, the organization requires additional information from the applicant, the time interval between asking for additional information and providing the information by the client must be added to the 8 weeks.” This requirement is decomposed into two control-flow compliance rules and one temporal compliance rule:

**Control-Flow Compliance Rule 1:** “Every time activity *submit request* (*submit*) is executed, it must be followed eventually by activity *publish result* (*publish*).”

**Control-Flow Compliance Rule 2:** “The sequence of activities *ask for additional information* (*ask*) and *receive additional information* (*receive*) may only be executed after the execution of activity *submit* and before the execution of activity *publish*.”

**Temporal Compliance Rule:** “The delay between execution of two subsequent activities *submit* and *publish* in all instances of a control-flow pattern, must be  $[\alpha, \beta + \beta_2]$  time units since time  $t$ , where  $\beta_2$  is the time between executing *ask* and *receive*.”

We formalized the two control-flow compliance rules using the elicitation technique described in the previous chapter. The formalization of the temporal compliance rule was derived by instantiating the generic temporal pattern of Fig. 5.12 as follows. The temporal rule requires *four* activities *submit*, *ask*, *receive*, and *publish* and a variation of the guard *delay-diff-activities* introduced earlier. The guard *delay-3* (*submit,ask,receive,publish,  $\alpha, \beta$* )  $\equiv t'_{publish} \in [t_{submit} + \alpha, t_{submit} + \beta + (t_{receive} - t_{ask})] \vee t_{submit} = \text{undefined}$  is assigned to transition *publish* with  $\alpha = 0$  and  $\beta = 8 \text{ weeks}$ . As activities *ask* and *receive* are optional, we have to provide valid time stamps in variables  $t_{ask}$  and  $t_{receive}$  in each iteration. Therefore, the instance start transition  $I_{st}$  of the temporal pattern (Fig. 5.12) initializes both variables to 0, i.e.,  $\{t'_{ask} = 0, t'_{receive} = 0\}$ , making their difference 0 if *ask* and *receive* are absent.

Municip.	Cases	Violations		
		#	delay (months) avg.	max.
M1	257	51	3	8
M2	166	37	4	15
M3	353	54	3	10
M4	269	38	3	11
M5	319	53	4	9

Table 5.2: Temporal compliance violations.

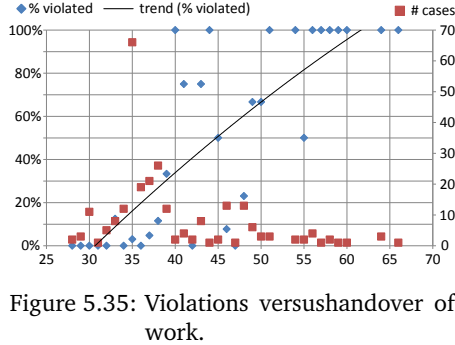


Figure 5.35: Violations versus handover of work.

In order to check compliance of the building permit process to these requirements, we obtained five event logs, each coming from a different municipality. Each log was extracted from the municipality’s case handling system and contained all activities performed for a case together with time stamps and resource information. In total we obtained 1408 cases as shown in Table 5.2 together containing 35352 events. Cases had 37 events on average and 97 events at most, distributed over 178 different event classes.

We first checked for compliance violations of the control-flow rules, followed by temporal compliance checking. In our analysis of the control-flow violations, we found 4 real violations and 40 *false positives* out of total number of 1408 cases in all municipalities. Most of the *false positives* occurred due to inaccurate time stamps and mistakes in data entries by a human user. The remaining real violations were mostly caused by the publication of the result before the municipality processed the additional information provided by the applicants. In general, the control-flow violations have not been severe because the process under analysis is quite standardized in all 5 municipalities. However, the temporal violations in all municipalities seem to be significant. Table 5.2 shows the result of temporal compliance checking in different municipalities.

Based on the diagnostic information we got from the temporal compliance checking, we investigated the cause of the high number of temporal violations. We found that in compliant cases requests for additional information were issued no later than 2 months after receiving the application. In all violating cases, requests for additional information were made only later than 2 months after receiving the application, i.e., when compliance was violated already. This suggests that the process primarily needs to be improved in the initial phase when employees gather and assess information about a particular application.

Another influential factor in increasing the violations, is the number of handovers among employees working on a case. The diagram of Fig. 5.35 shows the distribution of cases and the percentage of violating cases over the number of resource handovers happening in a case, for *M1*. The *x*-axis of Fig. 5.35 ranges over the number of hand-overs of work; the majority of cases had around 30-50 handovers, though there are several cases with up to 66 handovers. The *y* coordinate of a blue diamond gives the percentage of cases with *x* handovers of work that faces a compliance violation. The share of violations increases as handovers increase. This observation suggests that less compliance violations occur when an employee handles several subsequent activities of a case.

Figure 5.36 visualizes the data set of the municipality *M1*. Each colored dot represents an event, the color of the dot indicates the resource involved in the event. All events of a case are arranged in one line, and we ordered cases based on their duration

In total 16 different resources worked on this process. The zoomed-in part of the figure shows the activities executed for one of the violating cases on 9<sup>th</sup> Oct. 2012. We chose this case based on the diagnostics we obtained during the temporal checking. This case is violating the temporal constraint and has a delay of nearly 5 months. As can be seen only on this day several hand-overs between *four* resources are detected. Similarly many other violating cases have a large number of handovers. On the contrary Fig. 5.37 shows a case that was processed by only one resource.

Based on our analysis, we suggest to make further analysis on organizational structure and division of work in *M1*. A process oriented division of work could decrease the number of handovers. In addition, a job rotation programme could enrich the skill set of employees to be able to execute more activities with respect to one case, hence decreasing the number of handovers and improve compliance.

Applying the technique presented in this paper, we were able to check compliance of all the cases in the event logs rather than being limited to sample based compliance checking. The technique is fast and works on large event logs because we can focus on events relevant to a specific compliance rule and abstract from all other events. The remaining effort for a human user is in formalizing the compliance constraints and analyzing results. The effort in formalizing constraints was kept low in our case study. We could elicit the control-flow rule using the elicitation techniques described in the previous chapter. The main effort was in expressing the temporal compliance constraint as the guard  $delay-3 (submit,ask,receive,publish, \alpha, \beta)$  presented earlier; once the guard was identified, the constraint could quickly be formalized by instantiating the generic

pattern through a wizard. Note that this formalization need to be done once. Checking could then be continued for all cases automatically.

The technique could identify, locate, and determine the extent of deviations. Such diagnostic information can be used by the business analyst to analyze the cause of the deviations.

## 5.8 Related Work

The work presented in this chapter is mainly based on [130]

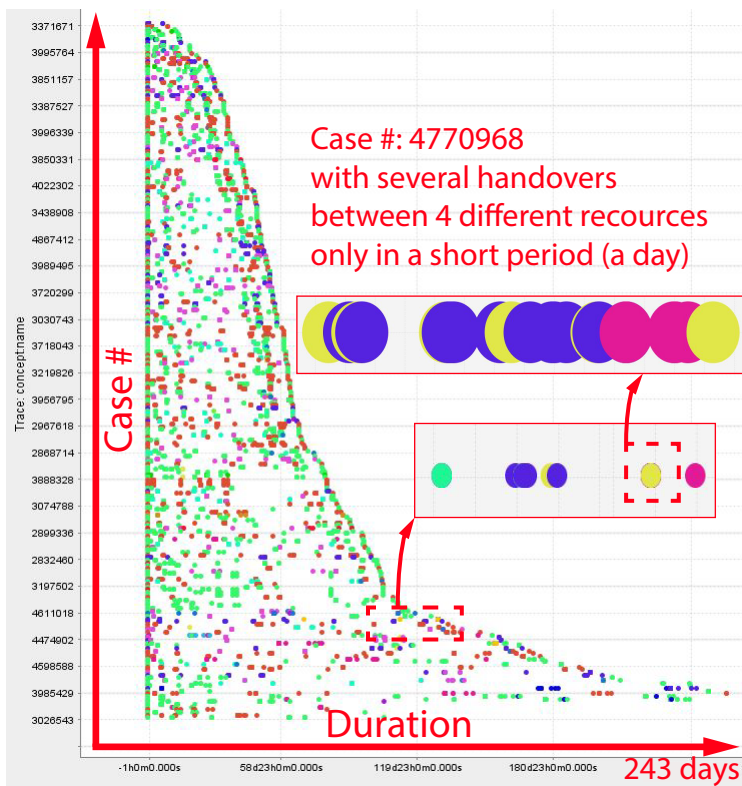


Figure 5.36: Several hand-overs observed for a violating case within a short period of time.



Existing work in temporal compliance checking primarily focuses on verification at design time or at run time.

It is possible to derive temporal properties of acyclic process models by annotating tasks with intervals of execution and waiting times; execution times and waiting times of the entire process can then be derived by interval computations and compared against predefined constraints of total execution times [39]. In addition, the time-critical paths of a process model can be computed [93]. In a similar fashion, the approach in [74] formulates temporal constraints in terms of deadlines for completing an activity (relative to another activity). Reasoning on time intervals is used to verify whether a constraint is violated.

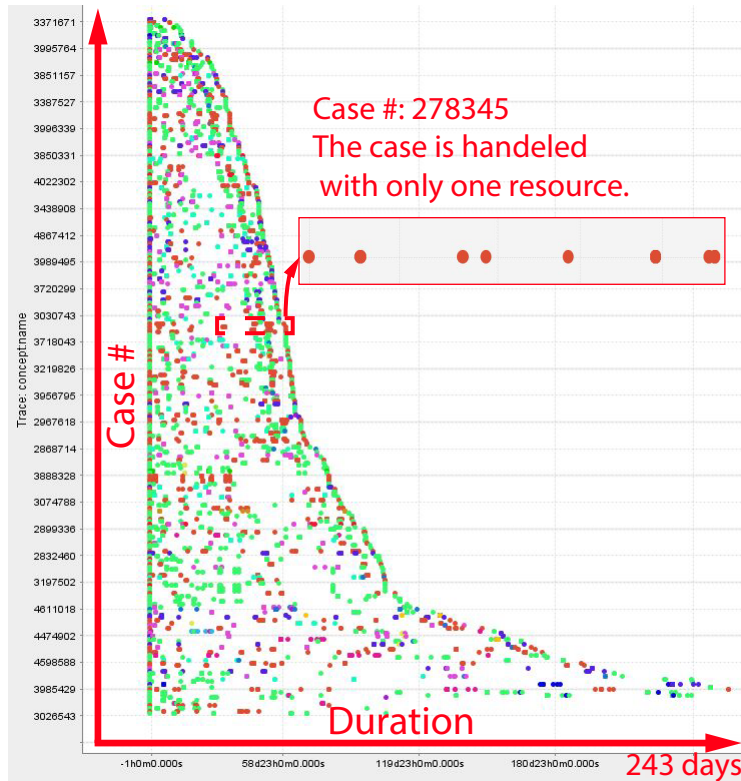


Figure 5.37: Example of a compliant case processed by a single resource.

For verifying that a process with loops satisfies a general time-related constraint, typically temporal model checking techniques are applied. The properties of interest are *metric* temporal constraints, e.g. deadline on execution of activities in a business process. *Metric temporal logic* (MTL), a temporal logic with metric temporal constraints, can express typical compliance requirements as presented in this paper. Unfortunately, the model checking problem for MTL is undecidable over models with infinite traces [67]. By introducing so called observers on atomic propositions, the problem whether a process model, given as a timed transition system (TTS), satisfies an MTL formula becomes decidable by a reduction to LTL model checking [5]. This approach allows to check temporal compliance of a real-time extension of Dwyer's specification patterns [38]. A similar approach is followed in [47] for checking whether an extended CCSL (Clock Constraint Specification Language) specification holds in a timed Petri net; CCSL is less expressive than the constraints that can be expressed and checked with our technique.

An alternative approach to describe temporal constraints is *timed Declare* [149] in which LTL-like constraints are extended with the notion of time. By a translation to timed automata, such constraints can be monitored at runtime to evaluate whether a process instance might or will violate a temporal constraint. A similar approach is proposed in [83].

In comparison, the technique presented in this paper focuses on backwards checking of temporal constraints in execution logs. The generic Petri net pattern proposed in Sect. 5.3 is capable to express all temporal constraints that we encountered in the works discussed above, and other temporal constraints such as cyclic temporal constraints not discussed elsewhere. Our technique detects *all* temporal violations in a trace, not just the first temporal violation encountered as it happens in model checking approaches. In case of violations also the compliant behavior (when a non-compliant event should have happened) is returned as diagnostic information.

The work presented in [106] focus on detecting temporal anomalies of an event log. The inputs of this approach is a Petri net representation of a process and an event log. The historical executions of activities in the log is used then to build a stochastic Petri net. A Bayesian network is inferred from an instantiation of the Petri nets mode during the replay of each case in the event log. Finally the likelihood whether an observation is drawn from the model distribution is computed. This approach cannot be used in context of temporal compliance checking because first of all the event log itself is used to generate the stochastic Petri net from the Petri net model of the process. Hence, the model reflects the log rather than compliance rules. That is, it is not clear how we can specify the

temporal rules with this approach, especially in case of intricate temporal rules. Next, one cannot get the precise diagnostics our technique provides e.g. location of violations, the deviation distance and compensation values.

## 5.9 Concluding Remarks

In this chapter we showed a complete set of temporal compliance rules. We discussed how we can specify all the rules listed in this repository by a single pattern that can be instantiated for each temporal compliance rule. The instantiated temporal pattern and an event log are the inputs for our temporal checking approach. We presented a robust technique for temporal compliance checking. For this, we first check whether the activities specified by a temporal rule occurred or not and then we check whether they occurred at the right time. By leveraging an existing data-aware conformance checking technique [33], we obtain detailed diagnostic information for both control-flow and temporal violations by computing an optimal data-aware alignment between an event log and the instantiated temporal pattern. The alignment then indicates where the event log deviated from the temporal rule. It also indicates how the violation could be compensated. We discussed the quality of diagnostics obtained using the temporal compliance checking technique. This approach is also supported by a ProM plugin. We have tested our techniques using real-life logs and compliance constraints.

# Chapter 6

## Data-Aware and Resource-Aware Compliance Checking

This chapter focuses on elicitation and checking of data-aware and resource-aware compliance constraints in a systematic way. Similar to temporal compliance checking (discussed in chap. 5), data-aware and resource-aware compliance checking build upon control-flow compliance checking (discussed in chap. 4).

In Chap. 5, we explained in detail how we formalize a temporal compliance rule as a data-aware Petri net and how to combine two techniques of control-flow checking and temporal checking to obtain the diagnostics we listed in Chap. 3. Data-aware Petri nets are used for formalizing data-aware and resource-aware compliance rules as well. However, these rules can be categorized in two main groups. The rules that restrict the data-flow of a process and rules that restrict control-flow of a process when a given data condition holds. In this chapter we will explain how to check compliance of each group of rules. The selected approach influences the diagnostics obtained. Hence, we will discuss how diagnostics obtained from both checking approaches need to be interpreted.

We discussed in Chap. 3 that depending on how rules are formalized (atomic versus a composite model) and how alignments are configured (cost of moves),

diagnostic information can differ. We investigated this matter for temporal constraints in Chap. 5. We will briefly discuss this matter in context of data-aware and resource-aware rules.

Figure 6.1 illustrates how the content of this chapter is organized in different sections. Section 6.1 presents a collection of data-aware and resource-aware compliance rules and the categorization of these rules into two main groups mentioned before. Section 6.2 sketches the two approaches for checking rules. The details of each approach are discussed by the help of examples in Sect. 6.3, and Sect. 6.4. We will discuss the quality of diagnostics obtained using these approaches in Sect. 6.5. Section 6.6 will briefly explain the differences of specifying a set of data and resource-aware compliance rules individually or together as a composite compliance model. We will show the feasibility and applicability of the checking techniques on real-life event logs in two different case studies. Section 6.7 explains the implementation of the techniques and then the details of the constraints in each case study and the results obtained, will be discussed. We will review related work in Sect. 6.8 and conclude with Sect 6.9.

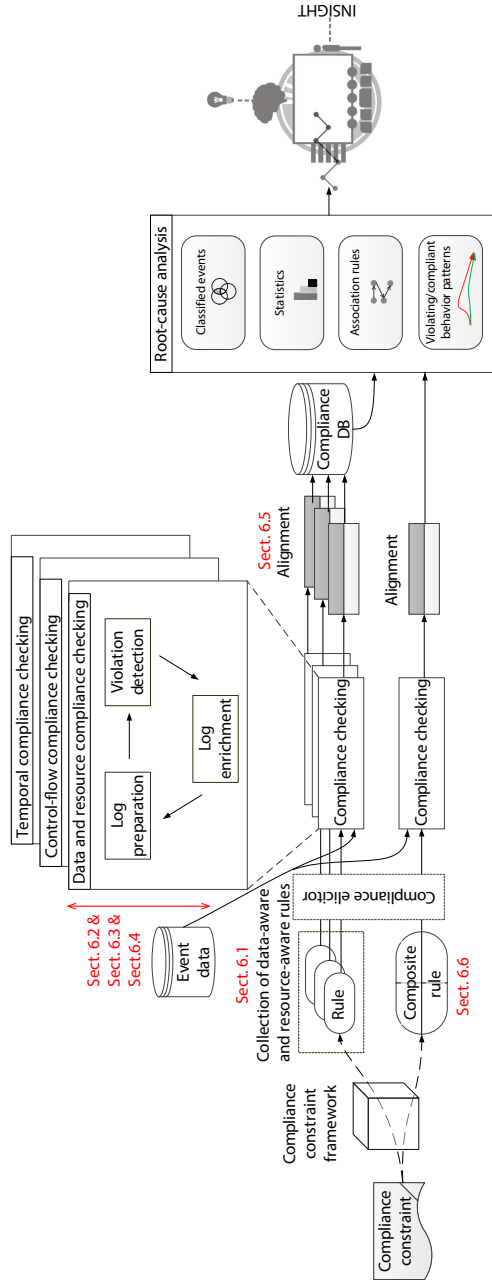


Figure 6.1: Thesis road map gives the mapping of the sections in Chap. 6 on to our compliance analysis approach.

Rule Description	Example
Four-eye principle: A security principle that requires segregating the execution of critical tasks and associated privileges among multiple users [3,7,19,36,41,116,119,132]	The person requesting purchase of goods should not be the one who approves it. A purchase order approval requires two signatures from different agents.
Authorization (Access control): A security principle that limits execution of activities or accessing a data object to authorized individuals [3, 7, 19, 36, 41, 116, 119, 132].	Only a financial manager can approve a loan.
Two (three) (four)-way match: An accounting rule that requires the value of two or more different data objects to match [3].	All vendor invoices that are based on purchase orders should be matched with purchase orders (two-way matching).
Activity $T$ may/must (not) be executed if attribute $X$ has the value $v$ in a given range; ( $X$ may be local to the activity $T$ or may appear anywhere in a case) [3, 41, 119].	An account must not be opened in case risk is high. During ventilation, patient must receive "propofol" with dosage of (5mg).
Activity $T_1$ may/must (not) be executed if attribute $X$ has value $v$ in a given range at activity $T_2$ (attribute $X$ is local to activity $T_2$ ) [3, 41, 119].	In case the respondent bank rating review is rejected during evaluation, an account must never be opened.
Activity $T$ must not change the value of attribute $X$ [3].	Bank account data must not change during payment.
The value of attribute $X$ must not change after activity $T$ is executed [3].	All invoices must be archived and no changes must be made to the document.
Activity $T_1$ may occur only if the value of attribute $X$ is increased/decreased by activity $T_2$ with $d$ (ICU medical guideline in a Dutch hospital (internal policy)).	If gastric tube feeding cannot be increased by (1,20 kcal/ml), then use 'Erythromycin'.
If attribute $X$ has value $v$ in a given range, then resource $R$ must execute the activity [3, 7, 19, 36, 41, 116, 119, 132].	Loans with value more than 1000000 Euro must only be approved by CFO.
If activity $T_1$ is done by agent $A$ , then activity $T_2$ must be done by the same agent [36].	A customer complain must be handled with the same agent registered the customer request.

Table 6.1: Collection of data-aware and resource-aware compliance rules.

## 6.1 Data-Aware and Resource-Aware Compliance Rules

Like for control-flow and temporal compliance rules, our extensive literature study in compliance checking [3, 7, 19, 36, 41, 116, 119, 132] resulted in a collection of rules confining process data and process resource. Table 6.1 also shown in Chap. 3 presents our collection of compliance rules taken from these sources and some more taken from practise e.g. medical guidelines. We found some typical restrictions on process data and resources such as four-eye principle (separation of duties), authorization level or three-way match and some domain specific compliance rules.

In addition to the classification presented in Table 6.1, all different types of data-aware and resource-aware constraints fall into two main categories: (1) constraints that enforce a restriction on data attributes, and (2) constraints that restrict activities when a certain data condition holds. For example a compliance rule such as the *four-eye principle* is of the first category. This rule specifies that two activities  $A$  and  $B$  must be executed by two different resources. The rule assumes that the underlying control-flow sequence is correct, i.e., no matter in which order two activities  $A$  or  $B$  are executed, the restriction is on the corresponding data attribute (resource). In case  $A$  is executed first by resource  $R_1$ , the rule will urge that  $B$  must not be executed by  $R_1$ . Whereas, if  $B$  was executed first, the restriction would be on the resource executing activity  $A$ . These rules are very similar to the temporal rules, i.e., we assume the underlying control-flow is correct and then we check whether it occurred with the correct resource or data. The checking procedure of these rules is also very similar to the temporal checking.

In contrary with rules like *four-eye principle*, a rule stating “activity  $B$  must not be executed for *gold* customers” belongs to the second category. This rule restricts the execution of activity  $B$  when a certain value (*gold*) for data attribute *customer type* holds. That is, based on a certain data condition, the constraint restricts the control-flow, i.e., restricting execution of activity  $B$ .

Next section will discuss our methodology for checking data-aware and resource-aware compliance rules of each category.



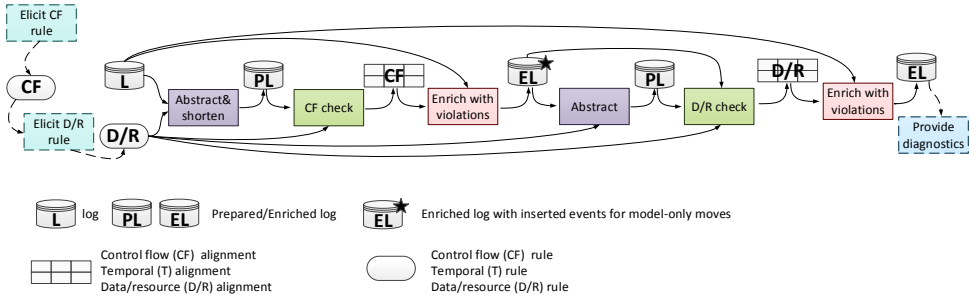


Figure 6.2: Methodology for checking the first category of data-aware and resource-aware compliance rules.

## 6.2 Methodology of Data-Aware and Resource-Aware Compliance Checking

This section presents our main contribution in the context of data-aware and resource-aware compliance checking as two dedicated approaches for checking the two categories of rules on past executions recorded in event logs.

Figure 6.2 depicts the approach we use for checking the compliance rules of the first category. For these rules, we need to check if a specified activity was executed with a correct data or resource. Hence, the restriction on data attributes must be checked and the underlying control-flow is assumed to be correct. To check every compliance rule, we need to detect its instances, i.e., we identify whenever the rule is triggered. We capture boundaries of a compliance rule by formalizing it as a data-aware Petri net.

An instance of a rule from the first category is defined based on the occurrence of an activity or sequence of activities. We use the elicitation technique described in Chap. 4 to specify an instance of a rule. The obtained atomic pattern is then augmented with guards to restrict data or resource. Accordingly, we prepare the event log and abstract from irrelevant information. Similar to temporal compliance checking, we first check whether the specified activities occurred or not by aligning the event log and the elicited atomic pattern. Afterwards we use the control-flow alignment to enrich the event log with control-flow diagnostics. The enriched event log and the atomic pattern augmented with guards are then used to build a data-aware alignment. After detecting data or resource violations, we enrich the event log with these diagnostics.

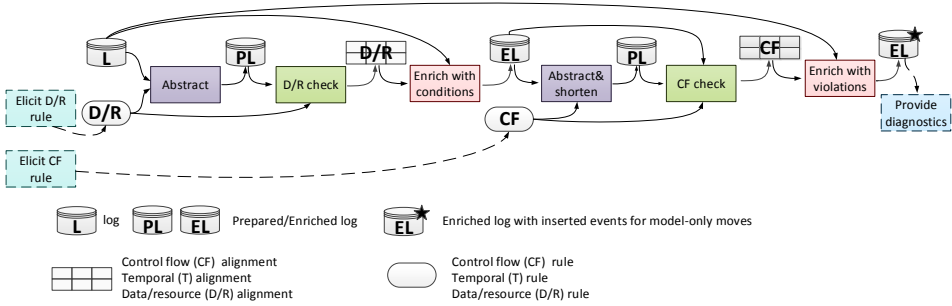


Figure 6.3: Methodology for checking data-aware and resource-aware compliance rules of the second category.

The compliance rules of the second category confine the execution of activities when a certain data or resource condition holds. These rules assume the data-flow is correct and the control-flow, i.e., the execution of activities under that data or resource condition, must be checked. Figure 6.3 describes the approach we employ for checking compliance rules of the second category. We specify the data or resource condition of a rule in terms of a data-aware Petri net with a generic and simple structure. The event log is prepared accordingly. After log preparation, we align the log and the data-aware Petri net to capture the situations where the condition of the rule holds. We use the obtained data-aware alignment to specify where in a log an instance of a rule is triggered. The the event log is enriched with this information. The next step is the elicitation of the control-flow condition of the rule as a Petri net. For this we use the rule elicitation technique described in Chap. 4. The elicited control-flow rule and the prepared log are then used for detecting control-flow violations. In the following we will elaborate on both approaches by example.

### 6.3 Compliance Checking of Rules Restricting Data Attributes

Consider the four-eyes principle in the context of a procurement process stating, *rule 1*): “purchase orders must be created and approved by different resources.” Suppose we would like to check the given event log *L* partially shown for a

L	resource	Chloe	Luis	Luis	Clara	Luis	Chloe	Amir
	role	clerk	expert	expert	manager	expert	clerk	director
	time	1	2	3	4	5	6	7
	activity name	receive PR	evaluate PR	create PO	approve PO	send to supplier	receive goods	pay
	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$

Figure 6.4: An example event log partially shown for process instance  $p_1$ .

single process instance in Fig. 6.4<sup>1</sup> against this rule.

This rule falls in the first category of compliance rules. For checking such rules we employ the approach shown in Fig. 6.2. In the following we will explain the checking procedure step by step for this example.

PL	resource	Luis	Luis	Clara	Luis
	role	expert	expert	manager	expert
	time	2	3	4	5
	activity name	evaluate PR	create PO	approve PO	send to supplier
	process instance	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_2$	$e_3$	$e_4$	$e_5$
	checking	$\Omega$	create PO	approve PO	$\Omega$

Figure 6.5: The event log, built upon original log of Fig. 6.4, is abstracted and shortened from irrelevant information.

<sup>1</sup>In the event log partially shown in Fig. 6.4, *PR* stands for purchase request and *PO* stands for purchase order.

**Step 1: Log preparation (Abstracting and shortening the event log).** Log preparation has been extensively discussed in Sect. 3.5.1, Sect. 4.1.1, and Sect. 5.2. In this step, we first select the attributes that are required for checking, i.e., *activity name*, and *resource*. Then we introduce the consistency attribute *checking*. The *checking attribute* gets value *create PO* when '*activity name = create PO*' and it gets the value *approve PO* when '*activity name = approve PO*', else it will get the generic value  $\Omega$ . We also shorten the event log by removing the extra  $\Omega$  events. Consequently we obtain prepared log *PL* shown in Fig. 6.5.

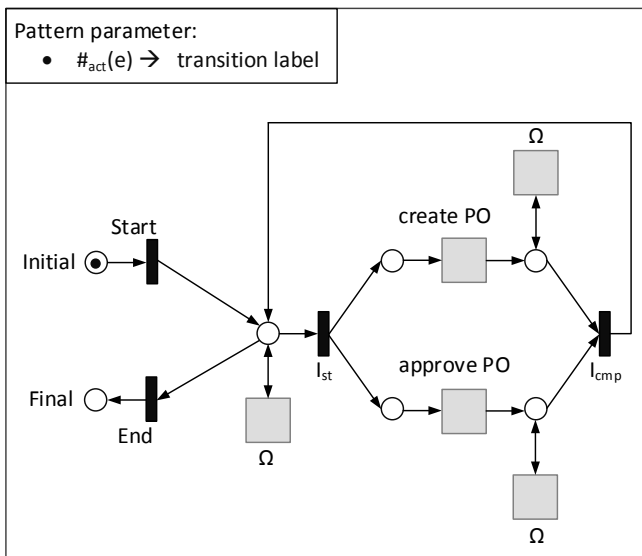


Figure 6.6: An atomic pattern specifying the control-flow condition of *rule 1*.

PL	resource	>>	Luis	>>	Luis	Clara	>>	Luis	>>
	role		expert		expert	manager		expert	
	time		2		3	4		5	
	activity name		evaluate PR		create PO	approve PO		send to supplier	
	process instance		p <sub>1</sub>		p <sub>1</sub>	p <sub>1</sub>		p <sub>1</sub>	
	event ID		e <sub>2</sub>		e <sub>3</sub>	e <sub>4</sub>		e <sub>5</sub>	
	checking		Ω		create PO	approve PO		Ω	
M	transition name	Start	Ω	I <sub>st</sub>	create PO	approve PO	I <sub>cmp</sub>	Ω	End

Figure 6.7: The control-flow alignment obtained from aligning *PL* of Fig. 6.5 and the atomic pattern of Fig. 6.6.

**Step 2: Control-flow checking.** Similar to temporal compliance checking, we first need to detect the control-flow condition of the rule, i.e., whether the two activities *create PO* and *approve PO* occurred or not. The atomic pattern shown in Fig. 6.6 captures the occurrences of the two activities. We can use the elicitation technique described in Chap. 4. The pattern in Fig. 6.6 is a configuration of the pattern in Fig. C.16 from the configurable rule repository which is instantiated for two activities *create PO* and *approve PO*. Note that the sequence of occurrence of these activities is not important for the underlying resource-aware rule (*rule 1*). Therefore as long as the control-flow pattern captures the occurrences of the specified activities, we can use it. An instance of this rule is triggered as soon as one of the two activities is executed. We align the prepared log *PL* with this pattern. The resulting control-flow alignment is shown in Fig. 6.7 indicating one instance for this rule.

**Step 3: Log enrichment with control-flow diagnostics.** The control-flow alignment obtained in the previous step does not indicate any violation. Therefore, the log enrichment step is straight-forward by adding the attribute *CF condition* which states whether the event in the log was compliant to the control-flow aspect of the rule. Value *F* for attribute *CF condition* in Fig. 6.8 indicates a control-flow violation and value *T* indicates that the event has been compliant with the control-flow aspect of *rule 1*. As can be seen in Fig. 6.8, the start and completion of the rule instance is marked.

EL*	resource	-	Luis	-	Luis	Clara	-	Luis	-
	role	-	expert	-	expert	manager	-	expert	-
	time	-	2	-	3	4	-	5	-
	activity name	-	evaluate PR	-	create PO	approve PO	-	send to supplier	-
	process instance	-	p <sub>1</sub>	-	p <sub>1</sub>	p <sub>1</sub>	-	p <sub>1</sub>	-
	event ID	e <sup>s</sup> <sub>1</sub>	e <sub>2</sub>	e <sup>s</sup> <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sup>s</sup> <sub>3</sub>	e <sub>5</sub>	e <sup>s</sup> <sub>4</sub>
	CF condition	F	T	F	T	T	F	T	F
	checking	Start	Ω	l <sub>st</sub>	create PO	approve PO	l <sub>cmp</sub>	Ω	End

Figure 6.8: The enriched log, built from control-flow alignment of Fig. 6.7, contains control-flow diagnostics.

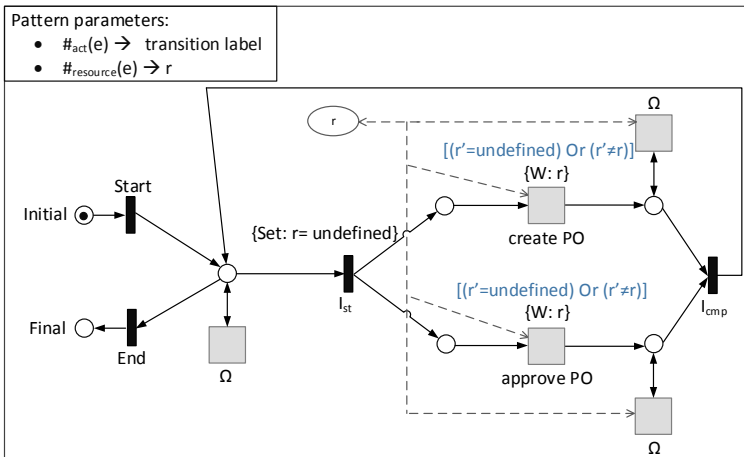


Figure 6.9: The atomic pattern of Fig. 6.6 augmented with resource condition of rule 1.

**Step 4: Data-aware alignment.** To detect violations on *resource attribute*, we first need to specify the data dimension of *rule 1* to constrain values for the *resource attribute*. For this, we will augment the atomic control-flow pattern (Fig. 6.6) with an appropriate guard. The data-aware pattern shown in Fig. 6.9 illustrates the control-flow pattern with required annotations. The attribute *resource* is shown as the ellipse in this pattern. The connections between transitions and resource attribute indicate the transitions that write or read a value from attribute resource.

The write statements  $\{W : r\}$  at *create PO*, and *approve PO* specify that *r* records the resource value at transitions *create PO*, and *approve PO*. We annotate the transition  $I_{st}$  with  $[r' = undefined]$  to express that the moment an instance of the rule is activated the value of variable *r* is set to undefined. The guard  $[(r = undefined) \text{ Or } (r' \neq r)]$  annotated at transition *create PO* and *approve PO* specifies that *r* is allowed to get a new value if its previous value is undefined, i.e.,  $(r = undefined)$  or in case it has already a value, the new value must be different with current value, i.e.,  $(r' \neq r)$ . The latter statement implies that activities *create PO* and *approve PO* must be executed with different resources.

We use the enriched log and the augmented atomic pattern with the resource condition to build a data-aware alignment. The data-aware alignment in Fig. 6.10 illustrates that both guards evaluated to *true*. Hence the log is compliant with the resource-aware rule.

Activities were executed with different agents.

EL*	resource	-	Luis	-	Luis	Clara	-	Luis
	role	-	expert	-	expert	manager	-	expert
	time	-	2	-	3	4	-	5
	activity name	-	evaluate PR	-	create PO	approve PO	-	send to supplier
	process instance	-	p <sub>1</sub>	-	p <sub>1</sub>	p <sub>1</sub>	-	p <sub>1</sub>
	event ID	e <sup>s</sup> <sub>1</sub>	e <sub>2</sub>	e <sup>s</sup> <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sup>s</sup> <sub>3</sub>	e <sub>5</sub>
	CF condition	F	T	F	T	T	F	T
	checking	Start	Ω	I <sub>st</sub>	create PO	approve PO	I <sub>cmp</sub>	Ω
M	transition name	Start	Ω	I <sub>st</sub>	create PO	approve PO	I <sub>cmp</sub>	Ω
	admissible resource	-	Luis	-	Luis	Clara	-	Luis

Figure 6.10: The data-aware alignment of the enriched log of Fig. 6.8 to pattern of Fig. 6.9, indicates that the two activities were executed by different resources.

**Step 5: Enrich log with diagnostics.** In this step, we enrich the event log with the diagnostics obtained. Figure 6.11 illustrates the enriched log. The diagnostics are marked with a hachured background.

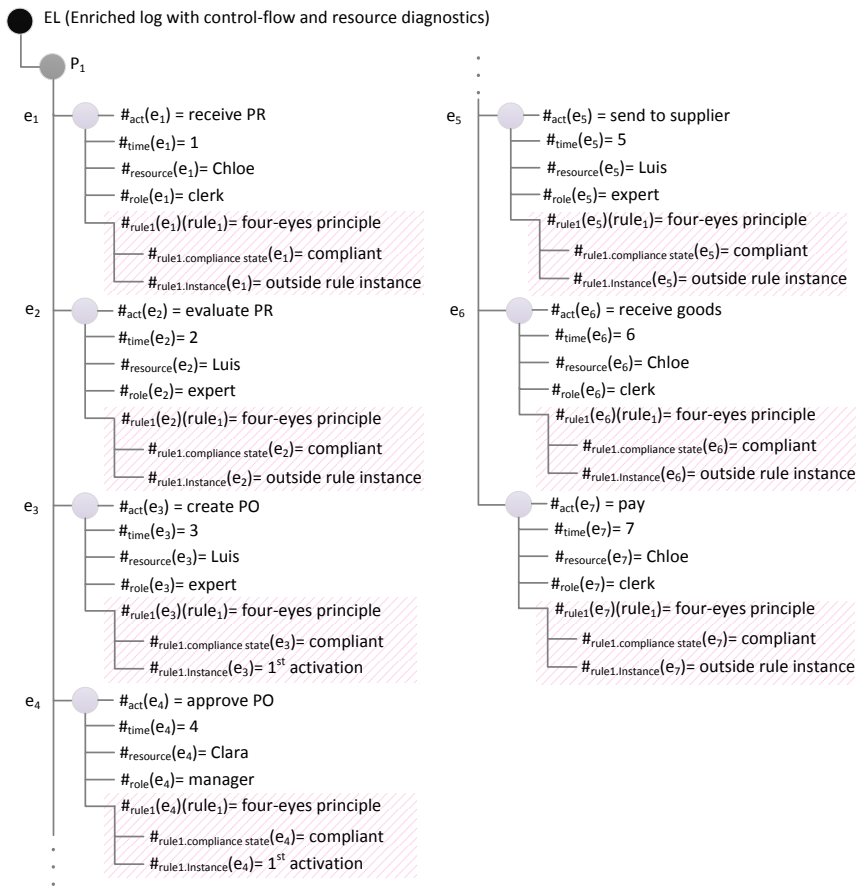


Figure 6.11: The enriched log obtained from data-aware alignment of Fig. 6.10 contains diagnostics about compliance of *rule 1*.



L	nutrition (multi-fiber)	10 kcal/ml	-	11 kcal/ml	-	13 kcal/ml	14 kcal/ml	-
	medication dosage				5 mg			
	diagnostis	C12	C12	C12	C12	C12	C12	C12
	time	1	2	3	4	5	6	7
	activity name	tube feeding	ventilation	tube feeding	Demp. Administration	tube feeding	tube feeding	X-ray
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>

Figure 6.12: Part of an example event log of a patient in ICU.

## 6.4 Compliance Checking of Rules Restricting Activities When a Certain Data Condition Holds

As was discussed previously, the second category of data-aware and resource-aware compliance rules assume the data-flow of a process is correct and one needs to confine the execution of activities under certain data condition. Consider a compliance rule taken from a medical guideline used in ICU department of a Dutch hospital stating *rule 2*: “If nutrition with multi-fiber cannot be increased via tube feeding by 2 (kcal/ml), then medication *Demperidone* must be administered to the patient”. This rule does not constrain the increase or decrease in multi-fiber nutrition but it requires the medication *Demperidone* to be administered to patients in case the nutrition is not increased. Hence, we need to capture the situation the rule must hold, i.e., where nutrition is not increased by the specified amount.

Suppose the event log in Fig. 6.12 is given: it shows events related to a process instance  $p_1$ . We would like to check whether this event log is compliant with the rule or not. To do so, we follow the approach visualized in Fig. 6.3. We will go through this approach step by step.

**Step 1: Log preparation.** At first, we select the attributes that are relevant for the rule, i.e., *activity name*, and *nutrition*. Next the consistency attribute *checking* is introduced. However, in this case the *checking* attribute does not reflect the values of the *activity name* but the data attribute to be checked (i.e., *nutrition*). For each event, we assign to the *checking* attribute the value *data write* wherever the attribute *nutrition* has a value, else the *checking* attribute will get the generic value  $\Omega$ . The prepared log is shown in Fig. 6.13.

**6.4 Compliance Checking of Rules Restricting Activities When a Certain Data Condition Holds** 209

PL	nutrition (multi-fiber)	10 kcal/ml	-	11 kcal/ml	-	13 kcal/ml	14 kcal/ml	-
	medication dosage	-	-	-	5 mg	-	-	-
	diagnostis	C12	C12	C12	C12	C12	C12	C12
	time	1	2	3	4	5	6	7
	activity name	tube feeding	ventilation	tube feeding	Demp. Administration	tube feeding	tube feeding	X-ray
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>1</sub>	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>
	checking	data write	Ω	data write	Ω	data write	data write	Ω

checking attribute captures the situations where a value for nutrition is recorded

Figure 6.13: The prepared log, built upon original log of Fig. 6.12, obtained from the first step.

**Step 2: Capturing the data condition.** The *checking* attribute in the prepared log indicates wherever an event has a value for its attribute *nutrition*. The next step would be to detect the situations that the data condition of the rule holds, i.e., (whenever *nutrition* has not increased by 2 kcal/ml). To do so we align the prepared log with the data-aware Petri net shown in Fig. 6.14. This net has a simple structure with only two transitions Ω, and *data write* and two attributes modelled as the two ellipses. The write statement {W:x} captures the value of attribute *nutrition* at each event that is mapped to *data write* and the guard [x' = x + 2k/cal] checks whether the new values of x are 2k/cal more than its current value.

We map respectively the values of the *checking* attribute in the log to the corresponding transitions in the net. Therefore events with '*checking*= data write' are mapped to the transition *data write* in the net and similarly the events with '*checking*=Ω' will be mapped to the Ω-labelled transition. The event attribute *nutrition* will be mapped to the modelled attribute *nutrition* in the net.



**6.4 Compliance Checking of Rules Restricting Activities When a Certain Data Condition Holds** **211**

We align the prepared log to this net and obtain the data-aware alignment shown in Fig. 6.15. As is indicated, the data-aware alignment detects two situations where the value of *nutrition* is not increased sufficiently (present at events  $e_3$  and  $e_6$ ). Recall from *rule 2* that these are exactly the situations where *Demperidone* must be administered. In the next step, we will check whether this indeed happened or not. Note that the data-aware Petri net described in Fig. 6.14 is generic and in essence can capture any type of data condition, we only require to adjust the guards and its parameters according to the rule.

EL	nutrition (multi-fiber)	10 kcal/ml	-	11 kcal/ml	-	13 kcal/ml	14 kcal/ml	-
	medication dosage	-	-	-	5 mg	-	-	-
	diagnostis	C12	C12	C12	C12	C12	C12	C12
	time	1	2	3	4	5	6	7
	activity name	tube feeding	ventilation	tube feeding	Demp. Administration	tube feeding	tube feeding	X-ray
	process instance	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$
	event ID	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$
	data condition	T	NA	F	NA	T	F	NA

Figure 6.16: The enriched event log with data condition obtained from the data-aware alignment of Fig. 6.15.

**Step 3: Enrich the event log with data condition obtained from data-aware alignment.** We enrich the event log with the data diagnostics obtained from the the data-aware alignment of Fig. 6.15. The enriched log is shown in Fig. 6.16. By introducing the attribute *data condition* we mark the situations where the data condition of *rule 2* holds or not. The value *T* (*true*) for the attribute *data condition* indicates that the nutrition was increased according to the rule, *F* (*false*) indicates that the nutrition was not increased according to the rule, and *NA* (*not applicable*) indicates that the event was irrelevant for the rule.



**Step 4: Log preparation for control-flow checking.** At this phase we need to prepare the event log for detecting control-flow violations. Similar to the log preparation step that was described in Chap. 4, log preparation at this phase has three parts: introducing the *checking* attribute, log abstraction, and log shortening. Figure 6.17 illustrates how we perform the first two parts.

The second checking step shall test whether “*tube feeding*” with less than a 2 kcal/ml increase was always followed by “*Demperidone*” as this would not be necessary when there was a proper 2 kcal/ml increase in *nutrition*. Rather, we have to select only those “*tube feeding*” events **without** a 2 kcal/ml increase, i.e., where the data-aware alignment of Step 2 returned False. To achieve this, we introduce as *checking* attribute the combination of *activity name* and *data condition*. On this *checking* attribute, an event “*tube feeding-F*” has to be followed by a “*Demperidone-x*” event; all other events are irrelevant for the rule and can be abstracted.

The ‘*checking attribute before abstraction*’ in Fig. 6.17 illustrates how we combine the values of *activity name*, and *data condition* at each event and abstract the values that are not relevant for control-flow checking and store it in the attribute *checking*.

PL <sub>2</sub>	nutrition (multi-fiber)	-	11 kcal/ml	-	13 kcal/ml	14 kcal/ml	-
	medication dosage	-	-	5 mg	-	-	-
	diagnostis	C12	C12	C12	C12	C12	C12
	time	2	3	4	5	6	7
	activity name	ventilation	tube feeding	Demp. Administration	tube feeding	tube feeding	X-ray
	process instance	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>	p <sub>1</sub>
	event ID	e <sub>2</sub>	e <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>7</sub>
	data condition	NA	F	NA	T	F	NA
	checking attribute	Ω	tube feeding-F	Demp. Administration-NA	Ω	tube feeding-F	Ω

↑  
sequence of Ω events are shortened

Figure 6.18: The prepared log has a reduced size by shortening sequence of Ω events.

The shortening of the event log is shown in Fig. 6.18 where we reduce the sized of the log by shortening sequences of Ω events (in this case only one event is discarded).

**Step 5: Detecting control-flow violations.** The atomic pattern shown in Fig. 6.19 models the rule in a way that matches our chosen *checking attribute* of

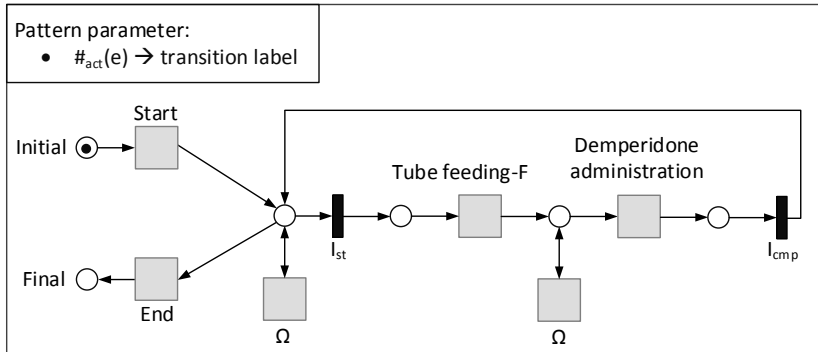


Figure 6.19: The atomic pattern specifying the control-flow condition of the rule.

the log in Fig. 6.18. An instance of this rule starts as soon an activity *tube feeding* occurs that did not increase *nutrition* by the specified amount. If such an activity occurs, then the rule requires that *Demperidone* needs to be administered to the patient.

We align the prepared log with this pattern and obtain the control-flow alignment shown in Fig. 6.20. The alignment detects two instances of the rule. The first instance is compliant and the second instance is violating. Occurrence of activity *Demperidone administration* is missing after a situation where *tube feeding* did not increase the nutrition by 2 kcal/ml.

**Step 6: Enrich the event log with control-flow violations.** At this phase, we enrich the event log with the diagnostics obtained from the control-flow alignment. The enriched event log is shown in Fig. 6.21. As is shown the artificial event  $e_6^S$  is added where the activity *Demp. administration* was skipped.





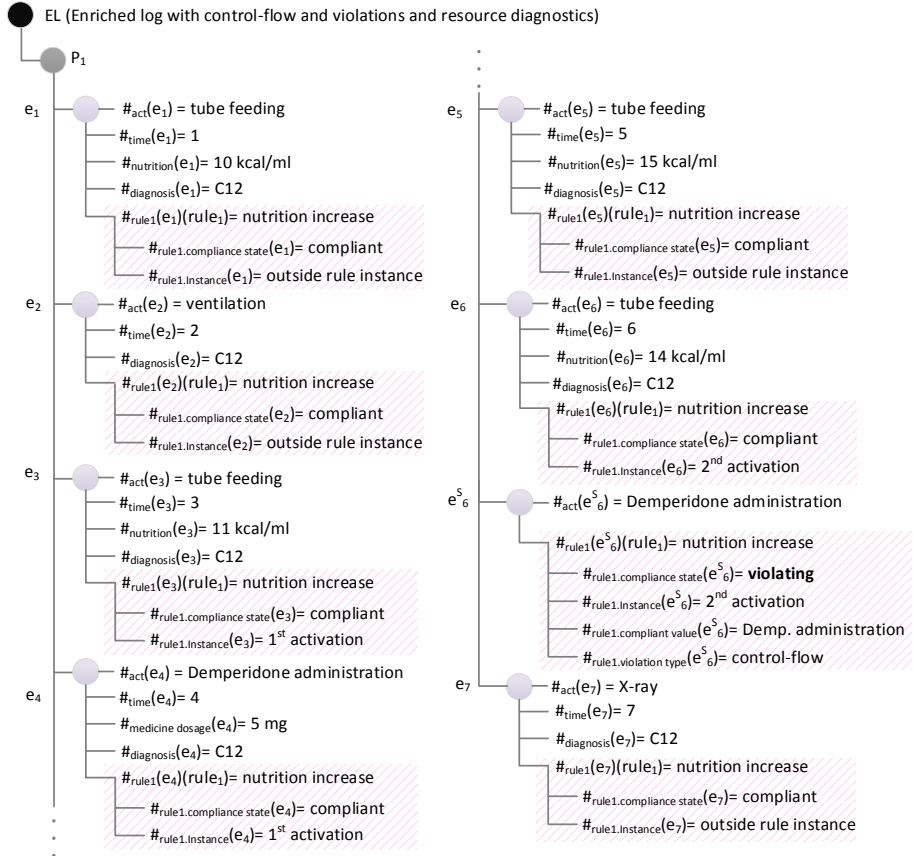


Figure 6.21: The enriched event log with diagnostics obtained from the control-flow alignment.

## 6.5 Diagnostics in Data and Resource-Aware Compliance Checking

So far, we discussed different groups of data-aware and resource-aware compliance rules and the appropriate checking procedure for each group. For the first group of rule, the *checking attribute* was simply a copy of the *activity name*. However, for the second group of rules, a checking attribute was created such that it refines the control-flow perspective based on data attributes. Deciding whether a data-aware rule is of the first category or of the second category of rules will lead to a specific choice of the checking technique and consequently on the diagnostics we obtain.

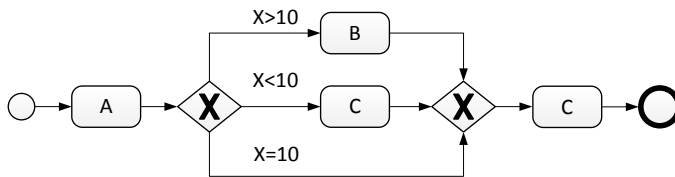


Figure 6.22: A process model in BPMN describing correct occurrences of activities  $A, B, C$ , and  $D$ .

Suppose a process model describes the execution of activities  $A, B, C$ , and  $D$  as is shown in Fig. 6.22 and a sequence of activities is given as:  $\sigma = \langle (A, x = 10), (B, x = 10), (D, x = 10) \rangle$ . According to the model of Fig. 6.22, activity  $B$  may occur with a value for attribute  $x$  to be greater than 10. Hence, the sequence of activities  $\sigma$  violates the behavior specified in Fig. 6.22. In this case two explanations may be provided about this violation: 1) activity  $B$  should not have been executed or 2) the value of attribute  $x$  should have been greater than 10. Picking any of these explanations as the more “precise explanation” imposes an assumption on the rule that is checked. The first explanation assumes that the priority is with the data-flow, i.e., value of attribute  $x$ , while the latter explanation assumes that the control-flow is correct (i.e., execution of activity  $B$ ). This brings us back to the definition of the rules in two different categories of data-aware and resource-aware rules. By choosing one or the other we follow a different checking procedure and finally we may end up with different diagnostics. In this example, if we define the rule as: “the value of attribute  $x$  must be greater than 10 at activity  $B$ ”, then the second explanation is more precise. However, if the rule is defined as ;“activity  $B$  may only be executed if attribute

$x$  is greater than 10”, then the first explanation is more precise. Note that in either way a violation is detected. Therefore, it is very important to precisely define a rule and pick the checking approach accordingly.

Alternatively one can use the *balanced conformance checking* [44] approach and consider data-flow and control-flow together. However, then the cost function in computing an alignment between a log and a data-aware behavior plays an essential role. That is, by adjusting the cost function we can prioritize the data-flow or control-flow. Balanced conformance checking is computationally expensive, because the search space for computing an alignment with the least cost between a log and data-aware Petri net grows exponentially in the length of process instances and number of variables. Besides performance issues, this approach has some drawbacks in case it is used in context of compliance checking as follows:

- Adjusting the cost function requires domain knowledge and technical knowledge to balance the two so that one gets the desired diagnostics.
- In case of considering rules separately as individual atomic patterns, this approach may give imprecise diagnostics. If the cost function is not chosen precisely then the condition of the rule may be ignored. For example for a rule of the second type -where the data-flow assumed to be correct- based on the cost of violations, the technique in [44] may detect a violation on data-flow rather than the control-flow. That is, for this technique both control-flow and data-flow have equal priority and only the total cost of violations is the influential factor on the diagnostics produced not the rule definition.

Note that we assume each **compliance rule** has a precise condition to be checked. A violation from a compliance rule can have severe consequences (e.g. legal consequences), therefore, a violation is only meaningful if we precisely define the condition of the rule. In case, we consider multiple rules together using balanced conformance checking, the alignment obtained from balanced conformance checking leaves interpretation of the results open to the user. Whether a detected violation is false positive or a real violation should be decided by the domain expert per violation.

The diagnostics obtained using data-aware and resource-aware compliance checking are influenced by the qualities described in Sect. 5.6 for diagnostics obtained from temporal compliance checking as well. These qualities include:

*minimizing the total cost of violations globally, minimizing the total value of deviation, and choice of the 'best' control-flow or data-aware alignment.*

## **6.6 Specifying a Set of Data and Resource-Aware Compliance Rules as Atomic Patterns or as a Composite Compliance Model**

We discussed various compliance rules and the approaches for checking each group of rules. We have discussed the quality of diagnostics obtained using each approach. Next, we will briefly explain, depending on how rules are formalized (atomic versus a composite model), diagnostic information can differ.

As with other types of compliance rules (control-flow and temporal), it is possible to model a set of compliance rules as a composite model. However, there are some limitations that one should be aware of. If compliance rules to be combined are of the same category, and overlap in activities or underlying conditions of rules to be combined, it is better to specify them as a composite compliance model. However, if rules are of different categories, it is not always possible or easy to specify them as a composite model. As described in Sect. 6.4, the second category of rules are always formalized as two different patterns; one capturing the data condition that triggered the rule and one specifies the control-flow condition of the rule. Accordingly in the checking procedure for the rules of the second category, we first build the data-aware alignment and then control-flow alignment. Whereas the first category of rules are formalized as one pattern and the during the checking, we first build control-flow alignment and then data-aware alignment (these approaches are visualized in Fig.6.2, and Fig. 6.3). Therefore it is not possible to combine rules from the two categories in one composite model.

## **6.7 Case Study Constraints and Results**

Our data-aware and resource-aware compliance checking techniques have been implemented in ProM, available from [www.promtools.org](http://www.promtools.org), and was applied in two different case studies on real-life logs. We briefly discuss the implementation in ProM and then provide details on the case studies.

### 6.7.1 Implementation

The presented technique is supported by two plugins in the *ComplianceFramework* package of ProM. The *Data and Resource Compliance Checking-Rules Restricting Attributes* plug-in (see Fig. 6.23) checks the compliance of event logs against rules of the first category and takes an event log and a data-aware Petri net as input. The *Data and Resource Compliance Checking-Rules Restricting Activities* plugin (see Fig. 6.24) checks the compliance of event logs against rules of the second category and takes an event log, a data-aware Petri net, and a classical Petri net as input. Both return diagnostics about deviations in the form of an alignment. The resulting alignments provide diagnostics by showing data and resource violations and control-flow violations projected over the original log. Figures 6.25, 6.26, 6.27 show different functionalities of the two plugins.

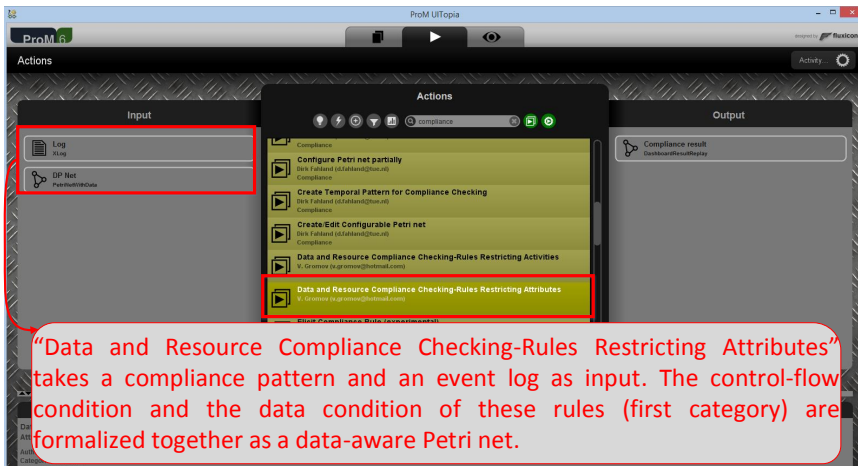


Figure 6.23: The *Data and Resource Compliance Checking-Rules Restricting Attributes* plug-in checks the compliance of an event log against the first category of compliance rules.

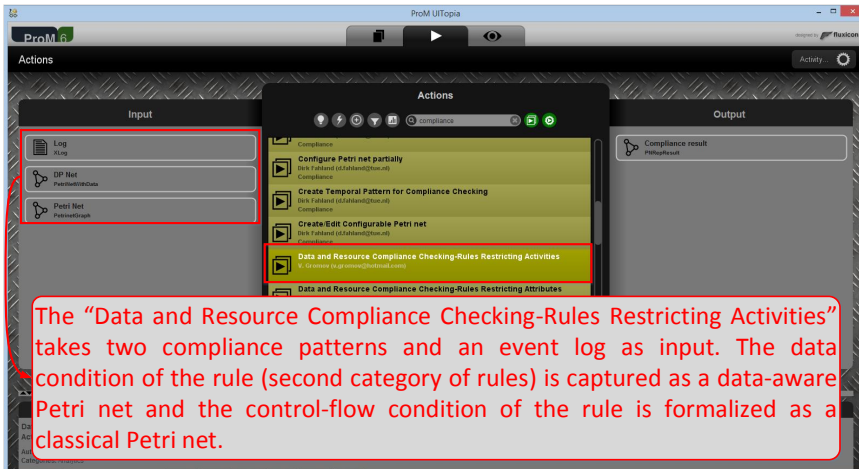


Figure 6.24: The *Data and Resource Compliance Checking-Rules Restricting Activities* plugin checks the compliance of an event log against the second category of compliance rules.

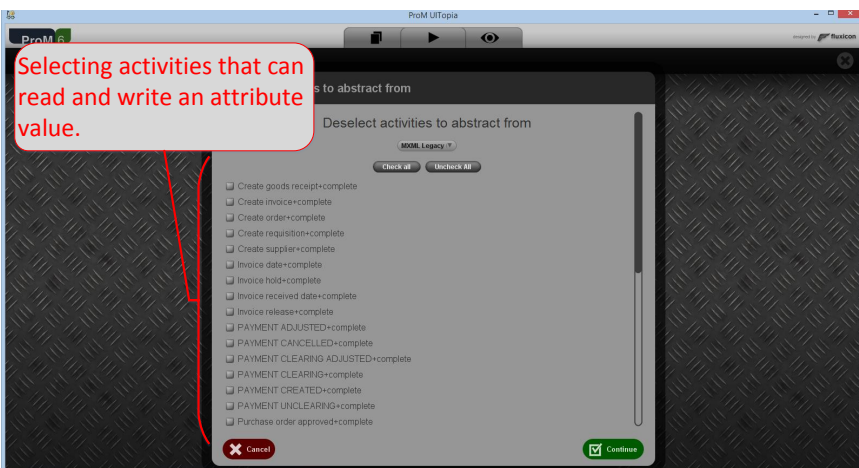


Figure 6.25: During the log preparation step, the activities that can read and write an attribute value are selected.

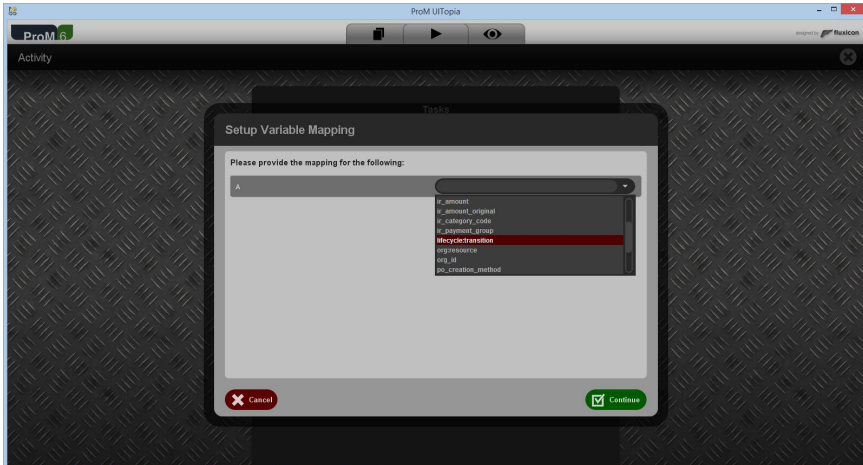


Figure 6.26: Mapping attributes of the compliance pattern to event log attributes.

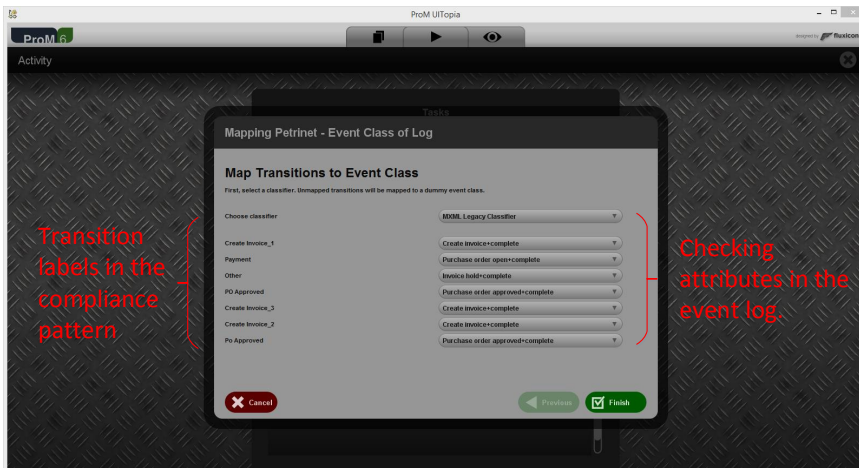


Figure 6.27: Mapping transition labels of the compliance pattern to the values of the checking attribute in the event log.

We applied our approach and toolset in two case studies: one for checking compliance of a personal loan process within a global financial organization and one for analyzing the process of patient treatment in ICU department of a Dutch hospital against a medical guideline.

### 6.7.2 Case Study 1: Constraint and Results

The event log related to the first case study is taken from the BPI challenge of 2012 and it has 13,087 traces and 262,200 events distributed over 36 activities<sup>2</sup>. The events have timestamps in the period from 1-Oct-2011 to 14-Mar-2012. The overall process can be summarized as follows: a submitted loan application is subject to some automatic check. The application can be declined if it does not pass any checks. If the automated checks passed, then applications are approved and activated (the activation of an application implies the payment of the loan value to the applicant). The applications may also be cancelled and some applications do not have a decision yet.

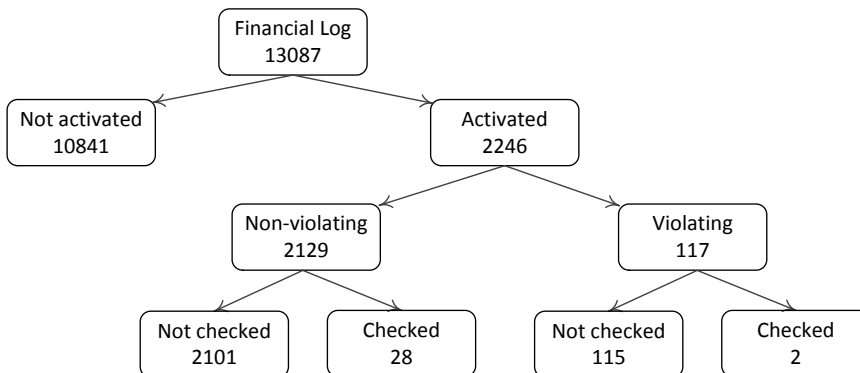


Figure 6.28: Classification of applications based on their compliance result.

A compliance rule restricting this process states: “loan applications with requested amount less than 5000 ( $< 5000$ ) or 50,000 and more ( $\geq 50,000$ ) must not be approved/activated”. This compliance rule is of the second type of data-aware compliance rules. We found 117 deviations from this compliance rule

<sup>2</sup>This event log is publicly available at:  
 dx.doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f



for the 13,087 traces executed. Figure 6.28 illustrates the violating applications. Out of 13,087 applications, 2246 cases were approved and activated while 10,841 were either declined, cancelled or undecided.

Furthermore some cases are considered suspicious and apart from regular checks, a special check for fraud was performed. It is interesting to observe that for two of the violating cases also the fraud check is performed.

Using the technique of this chapter, we were able to check a data-aware rule in a real operation setting. We detected all violations and got diagnostics about the violating activities. Further analysis can be done on detecting possible common characteristics in violating cases that can differentiate them from compliant ones. We will discuss our approach for this type of analysis in Chap. 7.3 and will apply it on a real-life example in Chap. 9.

### 6.7.3 Case Study 2: Constraints and Results

In this case study we investigated the compliance of an event log taken from ICU department of a Dutch hospital with a medical guideline restricting tube feeding nutrition of patients. The event log has 1207 traces; each trace recorded the treatment that a patient received in ICU department. The compliance constraint states: “If gastric tube feeding cannot be increased then use *Demperidone* or *Metoclopramide*. The starting dosage is usually (12·50ml) and it is recommended that the increase follows the pattern (12·50ml, 12·100ml, 12·120ml)”.

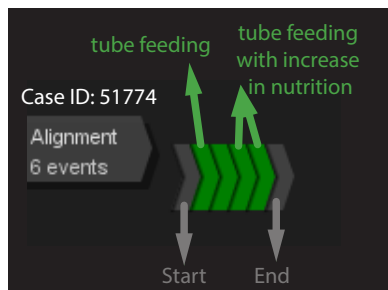


Figure 6.29: An example of a compliant case where nutrition was increased via tube feeding.

The guideline is not precise here, hence we check two different data-aware rules: (1) The nutrition must increase, else *Demperidone* or *Metoclopramide* must be administered to the patient (the running example presented in this

chapter is a modified version of this rule). (2) the increase must follow the pattern (12·50ml, 12·100ml, 12·120ml).

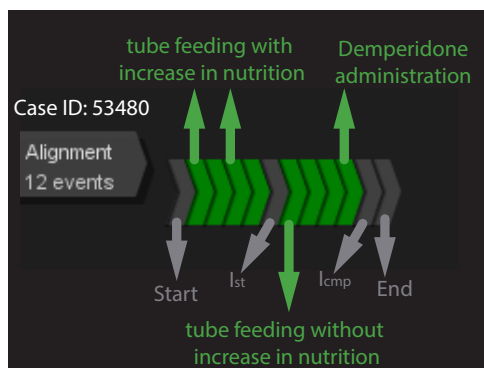


Figure 6.30: An example of a compliant case where *Demperidone* is administered to the patient when nutrition was not increased via tube feeding.

**First results:** We first checked if the nutrition has been increased for the respective patients and if not, we checked whether they received either *Demperidone* or *Metoclopramide*. We observed that from 1207 patients treated in ICU, 209 received tube feeding nutrition. In addition we found that *Metoclopramide* has not been administered for these patients; only one patient has received this medicine and it has been independent from tube feeding nutrition. For 72 patients, the tube feeding nutrition had increased without any problems. An example case is shown in Fig 6.29: patient 51774 received tube feeding while nutrition was increased.

56 patients received *Demperidone* when nutrition was not increased. Case 53480 shown in Fig. 6.30 is of this group. As can be seen, an instance of this rule is activated when nutrition did not increase the tube feeding, and this event is followed by a *Demperidone* administration.

We observed in total 81 violating cases. We identified several patterns for these violations. In some of the violating traces, we observed that although nutrition has increased, patients received *Demperidone* (an example is shown in Fig 6.31). It could be that *Demperidone* was administered to these patients independent from nutrition and for other purposes.

Another group of violations is related to patients that did not receive *Demperidone* although nutrition was not increased for them. An example of these

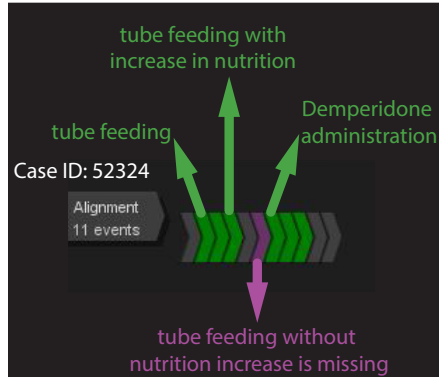


Figure 6.31: An example of a case where *Demperidone* is administered although nutrition was increased via tube feeding.

cases is illustrated in Fig. 6.32.

**Detailed analysis of violations:** We checked these traces further and found that this violation occurs in two situations. In one of these two situations, there were *real violations* because patients never received *Demperidone* despite the nutrition was not increased. However, for another group of patients we observed several iterations of tube feeding nutrition with an increase inside every iteration. For example a patient has received nutrition following two times the pattern ( $12 \cdot 50ml, 12 \cdot 100ml, 12 \cdot 120ml$ ). That is we see the increase in every occurrence of the pattern, yet the second occurrence of the pattern starts again from ( $12 \cdot 50ml$ ).

**Analysis of the second part of the rule:** We also investigated if the recommended pattern of increase has been followed or not. We found out that for less than 20% of patients the recommended pattern of the guideline is followed. From the remaining patients, we identified 3 groups of patients. One group of course were the 56 patients for whom the nutrition has not been increased. For the second group of patients, the nutrition was increased with the pattern ( $12 \cdot 50ml, 12 \cdot 100ml, 12 \cdot 140ml$ ). The third group mostly followed the pattern ( $500ml, 1000ml, 1500ml, 1700ml$ ). These patients received *Demperidone* once in every 24 hours, unlike the other groups that received *Demperidone* every two hours a day.

Applying the technique presented in this chapter, we were able to automatically check compliance for all the cases in the event logs. The technique works

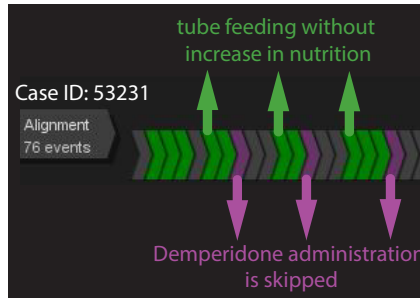


Figure 6.32: An example of a violating case where *Demperidone* is not administered to the patient although nutrition was not increased via tube feeding.

on large event logs because we can focus on events relevant to a specific compliance rule and abstract from all other events. Our technique identified and located the deviations by visualizing for each trace the difference between admissible and observed behavior in the alignment. The extent of the deviation is reported in two ways: (1) The alignment visualizes observed and expected values of deviating events. (2) We compute statically information about the number of deviations per case, in total, etc.

## 6.8 Related Work

This chapter is based on the paper [131].

Existing work in data and resource compliance checking mainly focuses on design time verification. In [15], authors incorporate data in specification of compliance rules. These rules are expressed using a query language which is an extended version of BPMN-Q and they are formalized by mapping them onto PTL. The approach is used to visualize violations by indicating execution paths in a process model causing them.

In [112] the control-flow is modeled and for any other process perspective a data object is defined. Later an object state change becomes explicit in the process model and then the process model is used to generate life-cycles for each object type used in the process. The consistency between business process models and life-cycles of business objects is checked. Apart from the fact that this approach focuses on verification of models, it is not discussed how violations are represented to the user and if further diagnostics about violations are

provided.

Other approaches such as [42, 119, 132] enforce compliance of business processes through the use of compliance patterns. These patterns include some of the data and resource-aware compliance constraints such as four-eyes principle. However, specific checking techniques are not discussed. Further on, the work in [64] addresses the verification of process models for data-aware compliance constraints. The authors do not apply a particular modeling language for specifying compliance rules, but introduce a general notion for data-aware conditions. For checking data-aware constraints, abstraction strategies are introduced to reduce the complexity of data-aware compliance checking and to deal with state explosion issues. This is achieved by abstracting from concrete state of data objects. The approach automatically derives an abstract process model and an abstract compliance rule.

Process mining techniques [134] offer a means to more rigorously check compliance and ascertain the validity of information about an organization's core processes.

The classical data-aware conformance checking [33] that we have applied in our approach, allow for aligning event logs and process models for multi-perspective conformance checking but it has some limitations. It can only check the compliance rules of the first category we discussed earlier. In addition if a deviation is observed for an activity that is restricted with several data attributes at the same time, the classical data-aware conformance checking can only indicate the deviation but not the specific data attribute(s) causing the deviation; resulting in less precise diagnostics. Moreover, using classical data-aware conformance checking, we cannot focus only on specific rules and abstract from other activities that are not restricted by respective compliance rule. That is, we need to provide a data-aware Petri net that captures the behavior of the whole process. Consequently, we need to know exactly how a process must be executed and we need to model all possible compliant process paths (i.e., sequence of activities and the values for data attributes they may have) that can be observed in a process. This task is not trivial, especially in some domains such as health care where the processes are very flexible. In this case we loose the flexibility offered by our compliance checking approaches. In our work we have tried to address the limitations mentioned by extending the classical data-aware conformance checking.

## 6.9 Concluding Remarks

In this chapter, we provided an approach for data and resource-aware compliance checking of behavior recorded in execution logs. We showed a collection of different compliance rules involving data and resource constraints and we discussed how to formalize them using data-aware Pteri nets. In addition, we provided two generic techniques to check rules involving multiple perspectives. Our technique separates control-flow, data and resource compliance checking as much as possible. It produced integrated diagnostics about both control-flow and data or resource related violations by enriching the log with diagnostic information between checking steps. In particular, our technique is capable of showing combined diagnostic information about violations of a compliance rule in a process instance. We showed this technique to be feasible for compliance rules on data dependencies between two or three data attributes. More complex rules are possible but require additional pre-processing and data-aware alignment steps. A more scalable technique is subject of further research.

We provided an implementation of our techniques in the *ComplianceFramework* package of ProM. The software has been tested on synthetic logs and two case studies involving real-life logs from a financial institute and an ICU department of a hospital. The results are encouraging: we were able to uncover various violations and no performance issues were encountered.

In the next chapter we explore violations that can be detected using various compliance checking techniques including: control-flow checking (Chap. 4), temporal checking (Chap. 5), and data-aware and resource-aware compliance checking (presented in this chapter). We will use the context of violations to analyze whether we can detect the cause of violations and whether we can differentiate compliance activities versus violating ones using the information available in a dataset.



# Chapter 7

## Compliance Diagnostics

In Chapters 4, 5, and 6 we presented techniques to detect compliance violations from different process perspectives. We demonstrated by the help of several case studies that the techniques presented are able to provide diagnostics about the rules that are violated, the violating activities, violating cases and the frequency of violations. We could also locate exactly where in a process a violation has occurred and which rule instances are violating. Moreover, in case of a violation, we could specify how an activity should have been executed to make the process compliant. This information helps us detect all violations in a precise manner. However, understanding the compliance level of a process and why a violation occurs and under which circumstances remain unclear. To be able to improve compliance, it is important to understand non-compliance and to understand non-compliance, it is necessary to have an overview of the compliance level of a process, prioritize violations based on their severity, and finally provide means to investigate the causes of violations.

This chapter focuses on understanding non-compliance. In this chapter, we give an overview of the compliance of a process. We present an approach for reporting compliance violations on different abstraction levels. At first, we aggregate violation data from different process perspectives and present a complete picture about all the deviations that occurred in a process. Then we rank deviations based on their business impact and frequency and suggest where a deeper analysis is recommended. In order to give more insights into the root-cause(s) of violations, we use data analysis techniques and identify relations between violations and their context information available in the event log. We would like



to investigate a specific problem to get in depth insight and develop hypotheses about why it happened. We present the obtained diagnostics in two ways: (1) by showing interactive tables and diagrams presenting process data and (2) by presenting a list of violations that are ranked based on their importance in non-technical terms such that these can be easier understood by business users.

Figure 7.1 illustrates how the content of this chapter is organized in different sections. In Sect. 7.1, we provide a motivating example to elaborate on problems we are addressing. In the motivating example, we analyze the compliance of a business process against several compliance rules. We model these rules combined as a composite model. We introduce a synthetic event data for this process. This chapter builds on the result of compliance checking techniques that were discussed in previous chapters. Using data-aware alignments, we detect all violation combined and use these diagnostics for analyzing violations and understanding them. This example is used throughout the chapter to introduce and discuss the details of our techniques. A general overview about the approach is introduced in Sect. 7.2. The checking procedure and the inputs are discussed in Sect. 7.3. In Sect. 7.4 and Sect. 7.5 we discuss how we obtain statistics about deviations and sort them. Our solution for investigating the root-cause(s) of deviations is presented in Sect. 7.6. The implementation of our technique and results of a case study are presented and discussed in Sect. 7.7. We conducted a survey to evaluate the understandability of the results generated with our technique by business users. The outcome of this survey is presented in Sec. 7.8. We discuss related work in Sect. 7.9 and conclude this chapter in Sect. 7.10.

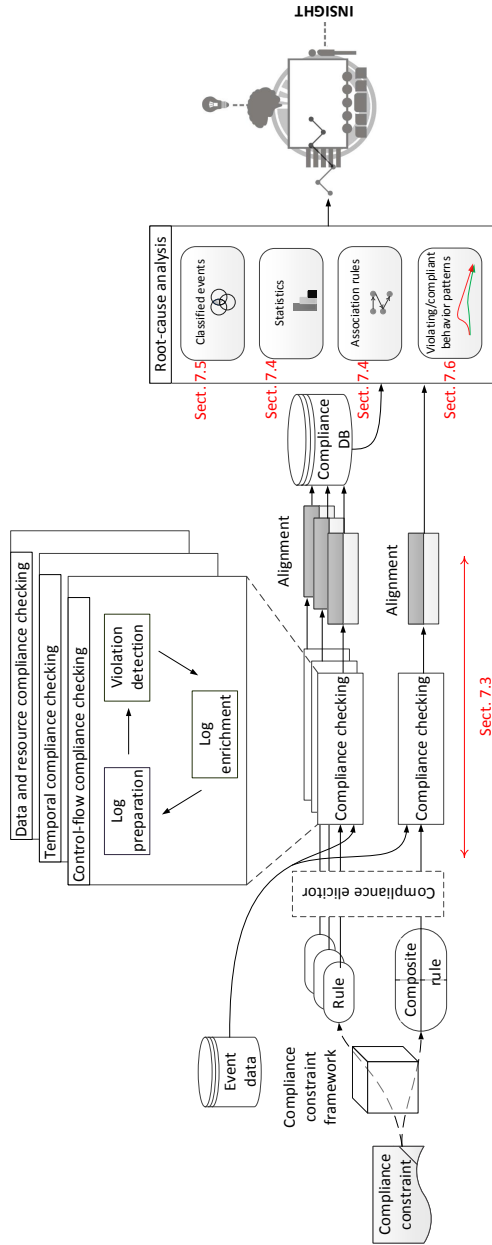


Figure 7.1: Thesis road map gives the mapping of the sections in Chap. 7 on to our compliance analysis approach.

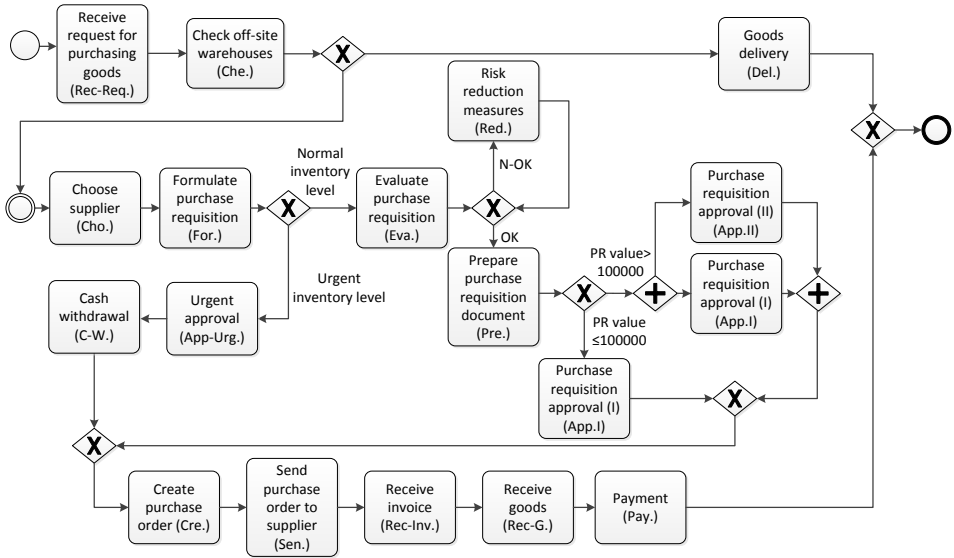


Figure 7.2: A process model for procurement.<sup>1</sup>

## 7.1 Motivating Example

In this section we introduce a motivating example which we use throughout this chapter to explain the problems we are addressing and our solutions to them. The example is built based on a real purchase process in a manufacturing company. Throughout the paper we will present synthetic executions of this process that illustrate compliance violations of a realistic business case. The process model, expressed in BPMN in Fig. 7.2 describes the procurement process in our example.

This process starts with activity receive request for purchasing goods (*Rec-Req.*). This happens whenever the warehouse department realizes that the inventory level of a material falls below a certain limit. Afterwards availability of the goods on demand is checked in off-site warehouses (*Che.*); if the goods are available they get delivered (*Del.*), otherwise the activity choosing supplier (*Cho.*) is triggered. After formulating the initial purchase requisition (*For.*),

<sup>1</sup>The acronyms, that are used to shorten the activity names are shown within brackets, there are acronyms that are used to shorten the activity names.

the standard procedure requires the evaluation of the requisition (*Eva.*) by an expert. This is in case the inventory level of the good on demand is not in an urgent state.

The choice of the supplier always carries a risk which is calculated based on a set of factors such as the past performance of the supplier, legal background, capital structure, and etc. Consequently the risk is classified as high or low. If risk is high and not acceptable (N-OK), the procurement expert follows some risk reduction measures (*Red.*). These measures may lead to a low risk or the risk may remain high. The procurement expert must decide based on the new information to continue with the risk reduction measures (*Red.*) or accept the risk. Once risk is accepted, a purchase requisition document can be prepared (*Pre.*).

If the purchase requisition value exceeds the threshold of 100,000 euros, then two approvals (*App.I*) and (*App.II*) are required by two different agents having directorate role. Otherwise, if the value is 100,000 euros or less one approval (*App.I*) is sufficient. Based on an approved purchase requisition, a purchase order is created (*Cre.*), and it is sent to the supplier (*Sen.*). The process continues with receiving invoice (*Rec-Inv.*). After receiving goods (*Rec-G.*) by the warehouse, the payment (*Pay.*) is done and the process ends. In exceptional cases when the inventory level drops to an even lower level (urgent), standard checks for the purchase requisition document may be omitted and the urgent approval (*App-Urg.*) will be executed. Urgent approval is usually followed by cash withdrawal (*C-W.*). In this case, the payment (*Pay.*) activity is limited to updating books and recording journal entries.

In addition to the general procedure of the procurement process, to comply with regulations and prevent fraud, the company must abide some compliance rules as well. Here we list six of the compliance rules<sup>2</sup> that the procurement process must comply to:

- *Rule 1:* Payment must always be done after receiving goods.
- *Rule 2:* If during the choice of a supplier risk is classified as high, at least once activity risk reduction measures must be executed.
- *Rule 3 (Four-eyes principle):* Purchase requisitions having a value of more than 100,000 euros must be approved by two signatures.

---

<sup>2</sup>Generic form of these rules can be found in the rule repositories discussed in Chap. 4, Chap. 5, and 6.

- *Rule 4:* Urgent approval may only be executed in critical situations when the inventory level is urgent.
- *Rule 5 (Authorization level):* Purchase requisitions may only be approved by agents carrying directorate role.
- *Rule 6:* Goods on demand must be supplied from company warehouses (off-site) unless the inventory level is urgent or there are no goods available in any of the company warehouses.
- ...

These compliance rules are just a subset of typical rules such a procurement processes has to comply to. Executions of processes are recorded in the form of *event logs*. For example assume that the procurement process has been executed 3714 times and a particular type of fraud has occurred. The problem relevant for the process owner is to know whether in the past each of the 3714 cases satisfied all of the above rules or not, and more specifically, when a case did not satisfy one or more of the rules, to provide detailed diagnostics about the deviations.

The diagnostic information needed to assist practitioners extends beyond the detection of deviations. Recall from Chap. 3, practitioners want to understand deviations from different angles, asking questions such as:

- *How many times each of the compliance rules listed above are violated in total?*
- *What type of violations occurred more than others?*
- *Can we relate a certain type of violation for instance violations related to four-eyes principle or authorization level to specific contextual information (e.g. to a specific supplier or material)?*
- *Can we find out based on the information available in the event log, who were the main human actors when a violation takes place?*
- *Is there an indicator available in the event log that can be used to predict future violations?*

## 7.2 Overview of the Approach

Figure 7.3 presents our proposed approach for helping organizations analyze the compliance of a process from different perspectives, report on compliance violations, and investigate reasons for compliance violations in order to answer the questions presented in Sect. 7.1.

This approach has three main components: *Compliance Checking* checks compliance of a process from different perspectives. The compliance checking component of the approach has been discussed intensively in the previous chapters. Note that so far we mainly discussed compliance checking of individual atomic patterns. In this chapter we specify a set of compliance rules as a composite compliance model. Using data-aware alignment, we check whether an event log adheres to the constraints of the composite compliance model or not and detect the violations and their type.

Violation data are aggregated to provide different statistics in the *Compliance Dashboard* component (Fig. 7.3-bottom left). The statistics are summarized at different abstraction levels enabling users to obtain detailed data and vice versa. It is important for business analysts to identify key insights in large amounts of event data. Hence, we present a sorted list of violations based on their importance. In this list, we also point out for every type of violation, the context information available in the event log that has a meaningful relation with that violation. We use association rule mining to quantify the relevance between a specific violation and its contextual information.

The “*Get Problem Insight*” component of the approach (Fig. 7.3-bottom right) allows us to study specific violations and investigate the conditions that lead to that violation; for this we use decision tree learning techniques. We will discuss root-cause analysis components (*Compliance Dashboard*, and *Get Problem Insight*) of the approach in detail using the procurement process example. However, first, we briefly discuss compliance checking for the example.

## 7.3 Compliance Checking

In this section we explain the composite compliance model that specifies the procurement process of the motivating example (Sect. 7.1) combined with all the compliance rules from different perspectives. Then, we illustrate how alignments reveal compliance deviations on this example.

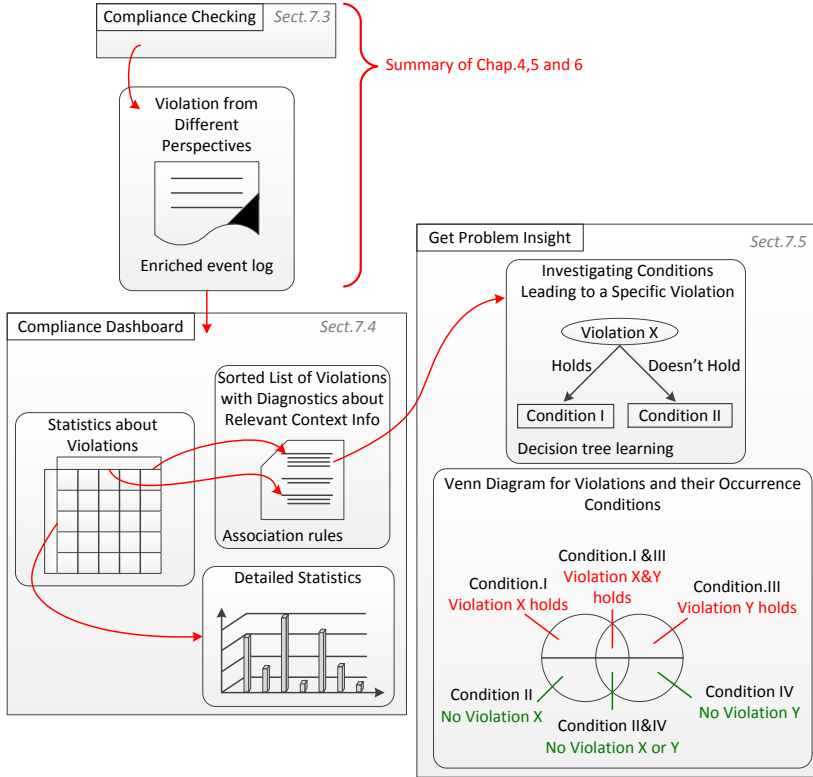


Figure 7.3: Overview of the approach.

### 7.3.1 Prescribed Behavior as a Composite Compliance Model

The data-aware Petri net *DPN-procurement* shown in Fig. 7.4 models the procurement process as that modeled in Fig. 7.2. Note that in this example we do not specify compliance rules separately. *DPN<sup>procurement</sup>* is a composite compliance model that specifies the presence of events and their ordering together with event attributes and their admissible values. Table 7.1 gives an overview of the attributes used in this process. In addition, due to readability reasons, we are not showing the variables and set statements. We only show write statements and guards.

Among others, the model specifies the compliance rules of the procurement

process discussed in Sect. 7.1 as indicated by the shaded parts in Fig. 7.4. All firing sequences of  $DPN^{procurement}$  correspond to process executions that comply with these compliance rules. Next, we choose two process fragments in  $DPN^{procurement}$  to elaborate on how this net describes a compliance rule.

*Rule 1* is a typical control-flow rule specifying that payment (*Pay.*) must be preceded by receive goods (*Rec-G.*). This behavior is formally expressed in Fig. 7.4 in the shaded part labeled (C). *Rule 3* and *Rule 5* are focused on attributes: *Rule 3* states that the four-eyes principle has to be applied on the

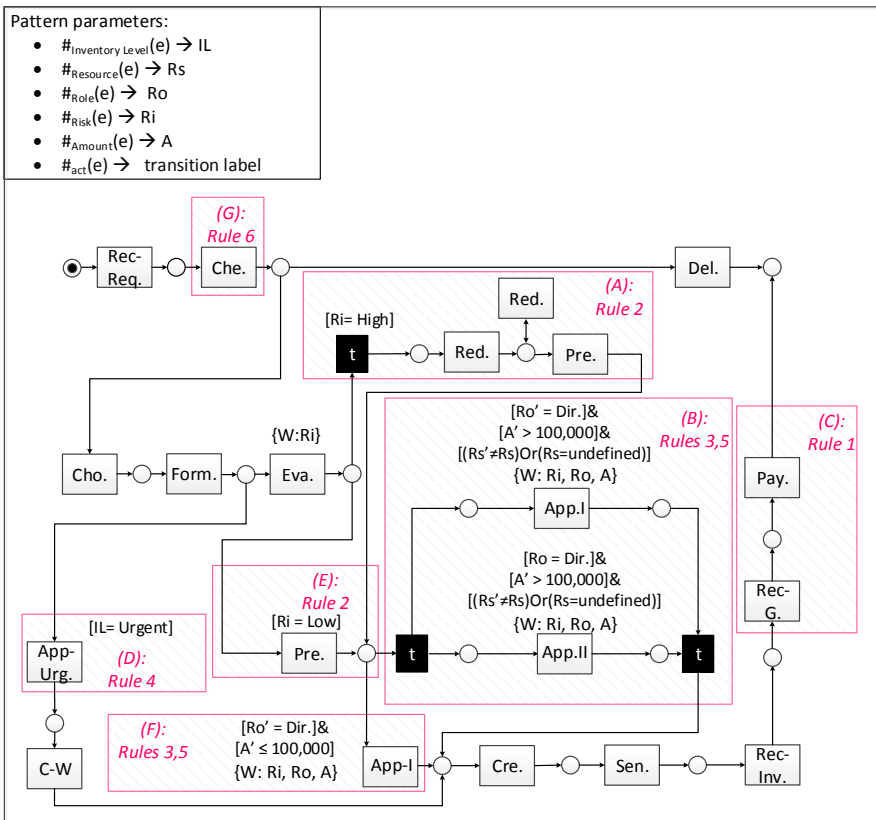


Figure 7.4: The data-aware Petri net  $DPN^{procurement}$  specifying admissible executions of procurement process example.



Attribute Name	Description
Inventory Level (IL)	Describes the state of the inventory level of the material in request that is available in the warehouse.
Resource (Rs)	Describes the name of the agent that executed an activity.
Role (Ro)	Describes the organizational responsibility of the resource that executed an activity.
Risk (Ri)	Describes the risk calculated for a supplier.
Amount (A)	Describes the amount value of a purchase requisition (PR).
Material (Mt)	Describes the material which is being purchased.
Supplier (S)	Describes the supplier.
Date (D)	Describes the date an activity was executed.

Table 7.1: Event attributes of the synthetic log obtained for the procurement process.

*Approval* if the amount is above 100,000 euros, and *Rule 5* demands that the approval has to be carried out by agents having *director* role in the company. This is expressed in the shaded part (B) where transitions have additional annotations that specify attribute values of events. Labeled transitions *App.I* and *App.II* represent respectively events with activity names *purchase requisition approval (I)* and *purchase requisition approval (II)*. The statement  $[Ro' = Dir.]$  in the guard of transition *App-I* and *App.II* specifies that these events must be executed with value *director* of their attribute *role*; expressing the restriction in *Rule 5*. Similarly the statement  $[A' > 100,000]$  in the guard of transitions *App.I* and *App.II*, specifies that both of these activities must be executed for cases with value more than 100,000 of their *amount* attribute; enforcing a part of *Rule 3*. *Rule 3* limits the execution of the two approvals such that execution of one will limit the resources that are allowed to execute the next one. To express this condition, we use variable *Rs*. The statement  $[Rs' \neq Rs) Or (Rs = undefined)]$  in the guard of both transitions then expresses that events with activity names *App.I* and *App.II* must be executed with different resources. This statement evaluates to true if  $Rs = undefined$  implying the corresponding event is the first one that writes in *Rs*, or if  $Rs' \neq Rs$  indicating that the value for *resource* attribute of the latter event must be different from its previous one.

### 7.3.2 Compliance Checking Based on Alignments

Table 7.1 shows (a part of) a possible event log for the procurement process. Violations are detected by aligning the event log and the data-aware model shown in Fig. 7.4 (see Chap. 2 for detailed explanation on data-aware alignment).

Figure 7.6 shows a part of the alignment obtained by aligning the log behav-

ior of Fig. 7.5 to the process instance runs described by the model of Fig. 7.4.

For simplicity we only show parts of the alignment that are related to the subnet (B) that captures *Rule 2*, *Rule 3* and *Rule 5*. The *model-only move* indicated in red, refers to the skipped event *prepare purchase requisition document (Pre.)*. The next violation indicates an *incorrect synchronous move*. The log event  $e_8^L$  has resource *Shiva* whereas the process instance run event  $e_8^R$  requires resource *Arash*.

This violation can be detected based on the statement  $[Rs' \neq Rs) Or (Rs = undefined)]$  annotated at *App.I* and *App.II* which together express that the two approvals must be executed with different agents in one process instance. This

L	<b>event ID</b>	$e_5^L$	$e_7^L$	$e_8^L$	$e_9^L$	...
	<b>process instance</b>	$\#_{pi}(e_5^L)=p_1$	$\#_{pi}(e_7^L)=p_1$	$\#_{pi}(e_8^L)=p_1$	$\#_{pi}(e_9^L)=p_1$	
	<b>activity name (act)</b>	$\#_{act}(e_5^L)=Eva.$	$\#_{act}(e_7^L)=App.I$	$\#_{act}(e_8^L)=App.II$	$\#_{act}(e_9^L)=Cre.$	
	<b>date (D)</b>	$\#_D(e_5^L)=01-05-09$	$\#_D(e_7^L)=04-05-09$	$\#_D(e_8^L)=10-05-09$	$\#_D(e_9^L)=11-05-09$	
	<b>amount (A)</b>	$\#_A(e_5^L)=110000$	$\#_A(e_7^L)=110000$	$\#_A(e_8^L)=110000$	$\#_A(e_9^L)=110000$	
	<b>resource (Rs)</b>	$\#_{RS}(e_5^L)=Ross$	$\#_{RS}(e_7^L)=Shiva$	$\#_{RS}(e_8^L)=Shiva$	$\#_{RS}(e_9^L)=Sina$	
	<b>role (Ro)</b>	$\#_{Ro}(e_5^L)=expert$	$\#_{Ro}(e_7^L)=director$	$\#_{Ro}(e_8^L)=director$	$\#_{Ro}(e_9^L)=director$	
	<b>risk (Ri)</b>	$\#_{Ri}(e_5^L)=low$	$\#_{Ri}(e_7^L)=low$	$\#_{Ri}(e_8^L)=low$	$\#_{Ri}(e_9^L)=low$	

Figure 7.5: Parts of the event log L showing four events from process instance  $p_1$ .

L	<b>event ID</b>	$e_5^L$		$e_7^L$	$e_8^L$	$e_9^L$	...
	$\#_{pi}(e^L)$	$p_1$		$p_1$	$p_1$	$p_1$	
	$\#_{act}(e^L)$	Eva.		App.I	App.II	Cre.	
	$\#_D(e^L)$	01-05-09		04-05-09	10-05-09	11-05-09	
	$\#_A(e^L)$	110000	>>	110000	110000	110000	
	$\#_{RS}(e^L)$	Ross		Shiva	Shiva	Sina	
	$\#_{Ro}(e^L)$	expert		director	director	director	
	$\#_{Ri}(e^L)$	low		low	low	low	
R	<b>event ID</b>	$e_5^R$	$e_6^R$	$e_7^R$	$e_8^R$	$e_9^R$	...
	$\#_{pi}(e^R)$	$p_1$	$p_1$	$p_1$	$p_1$	$p_1$	
	$\#_{act}(e^R)$	Eva.	Pre.	App-I.	App-II.	Cre.	
	$\#_D(e^R)$	01-05-09	02-05-09	04-05-09	10-05-09	11-05-09	
	$\#_A(e^R)$	110000	110000	110000	110000	110000	
	$\#_{RS}(e^R)$	Ross	Katy	Shiva	Arash	Sina	
	$\#_{Ro}(e^R)$	expert	clerk	director	director	director	
	$\#_{Ri}(e^R)$	low	low	low	low	low	

Figure 7.6: Alignment A resulted from aligning event log L and process instance run R of process model  $DPN^{procurement}$ .

statement is evaluated to *true* for execution of the first activity (*App.I*), but it is evaluated to *false* for *App.II* since both log events are executed by the same agent *Shiva*; violating the four-eyes principle.

## 7.4 Violation Statistics

In the previous chapters, we presented individual alignments and discussed the diagnostic information about compliance and non-compliance at the event level and at the process instance level. In order to get insights regarding the compliance status of the entire process, we will aggregate the diagnostic information from the event level to activity level. In the following, we present several statistics and a dashboard view to explore these statistics interactively.

Figure 7.7 shows part of the *Compliance Dashboard* with statistics for the activities in the procurement process. The table shown in this figure contains the list of activities and statistics of different violation types for every activity. As is shown in this table, activity check off-site warehouses (*Che.*) was skipped 1447 times. Based on the company internal policies, this activity may be skipped in *urgent* situations (although not modelled, as it is not the standard procedure). In this case the procurement department proceeds with buying the requested goods directly without following the standard procedure. This could be an explanation for occurrences of this deviation. Note that activity urgent approval (*App-Urg.*) has indeed been executed many times (1136 times). However, the “accepted” deviation (i.e., skipping check off-site warehouses followed by the urgent approval) can be costly because the standard approvals are skipped. Hence, there is a room to investigate about the high number of urgent cases.

Another deviation observed refers to the activity purchase requisition approval-I (*App.I*). This activity was executed 314 times with incorrect value for attributes *role* and *amount* which indicates frequent deviations from one or more compliance rules including *authorization level* and *four-eyes principle*. Similarly activity purchase requisition approval-II (*App.II*) was executed 499 times with an incorrect value for attributes *resource*, *role* and *amount*; referring to violations from one or more rules. In addition, we can observe that activity payment (*Pay.*) is executed twice where it was not allowed and two times it was skipped, suggesting that activity payment (*Pay.*) was executed in a wrong situation. The other deviation observed refers to three instances where activity prepare purchase requisition document (*Pre.*) is executed with a wrong value for *risk*.

Activity	# Moves in Total	Attributes with Incorrect Value							Unwanted Events (log-only move)	Skipped Events (model-only move)
		Risk	Resource	Amount	Supplier	Role	Material	Inventory		
Rec-Req.	3714	0	0	0	0	0	0	0	0	0
Che.	3714	0	0	0	0	0	0	0	0	1447
Cho.	3337	0	0	0	0	0	0	0	0	0
Form.	3337	0	0	0	0	0	0	0	0	0
Eva.	2201	0	0	0	0	0	0	0	0	0
Red.	1531	0	0	0	0	0	0	0	0	0
Pre.	2201	3	0	0	0	0	0	0	0	0
App.I	2201	0	0	314	0	314	0	0	0	0
App.II	1921	0	499	499	0	499	0	0	0	0
App-Urg.	1136	0	0	0	0	0	0	0	0	0
C-W	1136	0	0	0	0	0	0	0	0	0
Cre.	3337	0	0	0	0	0	0	0	0	0
Sen.	3337	0	0	0	0	0	0	0	0	0
Rec-Inv.	3337	0	0	0	0	0	0	0	0	0
Rec-G.	3339	0	0	0	0	0	0	0	0	0
Pay.	3337	0	0	0	0	0	0	0	2	2
Del.	377	0	0	0	0	0	0	0	0	0

Figure 7.7: Deviation statistics for procurement process example.

**Obtaining statistics of high-level compliance analysis.**

To provide a highly aggregated view on the violations as those shown in Fig. 7.7 for the procurement process, we first divide an alignment into its elements, i.e., alignment moves. With  $M$  being the collection of all alignment moves, we can filter  $M$  to build different sets of alignment moves, each characterized by a specific deviation type.

The input to computing the statistics is the alignment  $A = \alpha(L, Mod) = (L, R, M, \leq^M)$  that relates  $\log L = (E^L, \#, \leq) \in BH$  to a run  $R = (E^R, \#, \leq)$  from a given model  $R \in Mod$ . For this alignment we can define various sets of basic move types as listed in Table 7.2.

In addition to classifying moves in general, a business user may want to aggregate results for a specific activity to show how many deviations in total and from each type were observed for that activity. Hence, we filter  $M$  for moves referring to a specific activity and different deviations of the activity in question. Such filters and statistics are not restricted to activities but can be defined for arbitrary attributes.

For instance, we would like to know how many times a specific attribute had an incorrect value. Table 7.3 shows some examples of statistics that can be obtained for a given attribute.

These filters on activities and attributes can be combined as the following

Name	Description and Definition
$M_{log\text{-only.move}}$	set of all log-only moves in the alignment. $M_{log\text{-only.move}} = \{(e^L, e^R) \in M \mid (e^R = \gg) \wedge (e^L \neq \gg)\}$ .
$M_{model\text{-only.move}}$	set of all model-only moves in the alignment. $M_{model\text{-only.move}} = \{(e^L, e^R) \in M \mid (e^L = \gg) \wedge (e^R \neq \gg)\}$ .
$M_{syn}$	set of all synchronous moves in the alignment. $M_{syn} = M \cap (E^L \times E^R)$ .
$M_{cor.syn}$	set of all correct synchronous moves in the alignment. $M_{cor.syn} = \{(e^L, e^R) \in M_{syn} \mid \#(e^L) = \#(e^R) \uparrow (dom(\#(e^R)) \setminus \{tID\})\}$ (See Sect. 2.4).
$M_{incor.syn}$	set of all incorrect synchronous moves in the alignment. $M_{incor.syn} = M_{syn} \setminus M_{cor.syn}$ .
$M_{violating}$	set of all moves having at least one type of violation. $M_{violating} = M_{log\text{-only.move}} \cup M_{model\text{-only.move}} \cup M_{incor.syn}$ .

Table 7.2: Sets of different basic move types.

examples show.

- $M_{act}^a \cap M_{log\text{-only.move}}$  are all log-only moves involving activity  $a \in Val$ ,
- $M_{act}^a \cap M_{model\text{-only.move}}$  are all model-only moves involving activity  $a \in Val$ ,
- $M_{act}^a \cap M_{violating}$  are all moves involving activity  $a \in Val$  and having at least one type of violation,
- $M_{syn} \cap M_{act}^a \cap M_{attr.incorrect.value}^x$  are all synchronous moves involving activity  $a \in Val$  while disagreeing on attribute  $x \in Attr$ ,
- $M_{syn} \cap M_{act}^a \cap M_{attr.missing.in.log}^x$  are all synchronous moves involving activity  $a \in Val$  where attribute  $x \in Attr$  is missing in the event log,
- $\{\#_{act}(e^R) \mid (e^L, e^R) \in M_{model\text{-only.move}}\}$  is the set of all activities involved in a model-only move,
- $\{a \in Val \mid |M_{act}^a \cap M_{log\text{-only.move}}| \geq 5\}$  is the set of all activities executed at least 5 times while model did not allow for it, and
- etc.

Name	Description and Definition
$M_{act}^a$	set of all moves involving activity $a \in Val$ . $M_{act}^a = \{(e^L, e^R) \in M \mid ((e^L \neq \gg) \wedge (\#_{act}(e^L) = a)) \vee ((e^R \neq \gg) \wedge (\#_{act}(e^R) = a))\}$ .
$M_{attr}^x$	set of all moves involving attribute $x \in Attr$ . $M_{attr}^x = \{(e^L, e^R) \in M \mid x \in dom(\#(e^R)) \cup dom(\#(e^L))\}$ .
$M_{attr}^{v,x}$	set of all moves having value $v$ for their attribute $x \in Attr$ . $M_{attr}^{v,x} = \{(e^L, e^R) \in M \mid x \in dom(\#(e^L)) \wedge \#_x(e^L) = v\}$ .
$M_{attr.missing.in.log}^x$	set of all synchronous moves where attribute $x \in Attr$ is missing in the event log. $M_{attr.missing.in.log}^x = \{(e^L, e^R) \in M_{incor.syn} \mid x \in dom(\#(e^R)) \setminus dom(\#(e^L))\}$ .
$M_{attr.missing.in.model}^x$	set of all synchronous moves where attribute $x \in Attr$ is missing in the run. $M_{attr.missing.in.model}^x = \{(e^L, e^R) \in M_{incor.syn} \mid x \in dom(\#(e^L)) \setminus dom(\#(e^R))\}$ .
$M_{attr.incorrect.value}^x$	set of all synchronous moves where log and model disagree on the value of attribute $x \in Attr$ . $M_{attr.incorrect.value}^x = \{(e^L, e^R) \in M_{incor.syn} \mid x \in (dom(\#(e^L)) \cap dom(\#(e^R))) \wedge \#_x(e^L) \neq \#_x(e^R)\}$ .
$M_{attr.violating}^x$	set of all moves involving a violating attribute $x \in Attr$ . $M_{attr.violating}^x = M_{attr.incorrect.value}^x \cup M_{attr.missing.in.model}^x \cup M_{attr.missing.in.log}^x$ .

Table 7.3: Different sets of moves related to a specific attribute.

The above expressions are only examples. The cardinality of these sets gives the frequency of the various deviation types in the entire event log.

Referring back to the high-level statistics of the procurement example shown in Fig. 7.7, for instance the number ‘1447’ for activity *check off-site warehouses* (*Che.*) is the cardinality of the set of all alignment moves with the activity name *check off-site warehouse* that were skipped ( $M_{act}^{Che.} \cap M_{model-only.move}$ ).

Fig. 7.7 shows aggregated results for deviations at the activity instance level in the entire event log. In addition to individual activities, a user may want to know how many *process instances* satisfy a compliance rule or violate it. Therefore, we give statistics on process instance level as well using the type of expressions listed earlier:

- $\{\#_{pi}(e^L) \mid (e^L, e^R) \in M_{log-only.move}\}$  are all process instances having a log-only move,
- $\{\#_{pi}(e^R) \mid (e^L, e^R) \in (M_{a,act} \cap M_{model-only.move})\}$  are all process instances having a model-only move involving activity  $a \in Val$ , and
- etc.

For example, we can give statistics about all process instances having at least one move with activity *App.I* executed with incorrect value for attribute *resource*. Hence, we can see in total what percentage of cases contain at least one occurrence of such violation. In our example, statistics at the process instance level are the same as those at the activity instance level since we do not have duplicate executions of one activity in a process instance.

By creating different sets of alignment moves related to any of the attributes in the event log, we can provide above mentioned statistics not only for activities but for attributes as well. For instance Fig. 7.8 (top) shows statistics related to the attribute *material* of the example. On the left side of the table, all the materials observed in the log are listed. The column *# Moves in Total* shows the number of moves that have a specific material. The columns in the middle indicate the number of moves that have an incorrect value for one of the listed attributes: *Risk*, *Resource*, *Amount*, *Supplier*, *Role*, *Material*, and *Inventory* that co-occur with a value in the column *Material*. The columns *Unwanted Events* and *Skipped Events* show the number of moves that have a violation of type *log-only move* or *model-only move* with the respective material. In this table, for example the value 86 in the column *Skipped Events*, is derived from the following expression:

- $M_{attr}^{aluminium,material} \cap M_{model-only.move}$

Material	# Moves in Total	Attributes with Incorrect Value							Unwanted Events (log-only moves)	Skipped Events (model-only)
		Risk	Resource	Amount	Supplier	Role	Material	Inventory		
Rubber	7077	0	123	135	0	135	0	0	0	137
Aluminum	6487	0	116	128	0	128	0	0	1	86
Glass	12964	3	196	215	0	215	0	0	0	600
Steel Plate	7188	0	128	145	0	145	0	0	0	137
Cooper	11308	0	177	190	0	190	0	0	1	489

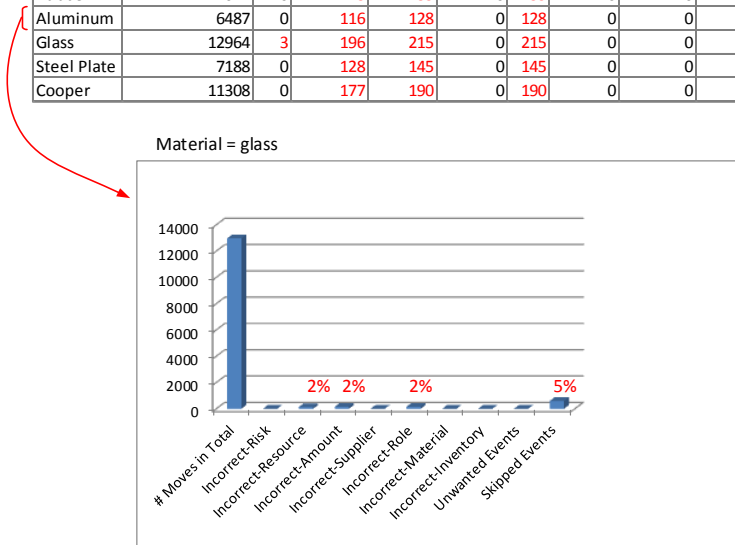


Figure 7.8: Statistics for attribute *material*(top) and detailed statistics for *material* = *glass*.

In another view in *compliance dashboard*, we can get detailed data about a specific attribute value. For instance, Fig. 7.8 (down) shows the statistics calculated for attribute *material* having the value *glass*. This bar chart shows that in 5% of the cases that the *material* requested is *glass*, we observe violations of type *model-only move*. In 2% of the cases, value *glass* of attribute *material* coincides with incorrect values for attributes *role*, *amount*, and *resource*.

## 7.5 Identifying and Ranking Problems

When the number of attributes and activities grows, focusing on important deviations by solely reading statistic tables and corresponding bar charts introduced in Sect. 7.4 will not be easy. Moreover, if there exists a meaningful relation between a deviation and its context information available in the event log, this relation cannot be highlighted using the aggregated statistics presented in the



previous section.

To guide users in identifying important and frequent patterns of violations, we prepare a *list of problem statements*.

### 7.5.1 Generating Lists of Problem Statements

Each problem statement indicates a deviation type and its relation with the contextual information of the violation at the event level or process instance level.

As context of a violation we consider *event-level attributes* of the violating

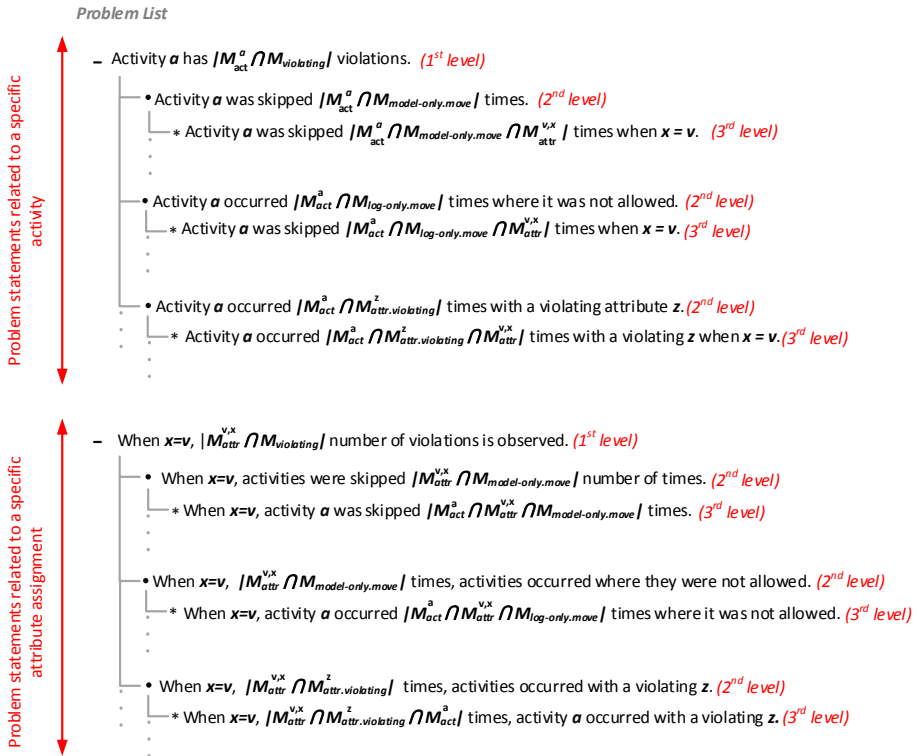


Figure 7.9: The structure of *problem lists*.

event and *trace-level attributes* which describe the information of the entire process instance in which the violation was observed. Technically, we make all trace-level attributes event-level attributes by copying them to each event.

Figure 7.9 shows how we structure the problem list hierarchically. In the problem list, deviations that are related to a specific activity or attribute value or combination of both will be grouped and listed as two categories of problem statements. Every problem statement in this list refers to a set of deviations that have some characteristics in common. For instance all the deviations of type ‘log-only moves’ for ‘activity *a*’. The sets of deviations are built using the expressions discussed in Sect. 7.4. We use these sets then to generate the problem statements in the problem list.

Since one of the main goals of compliance checking is to enable practitioners and business users to get insight and understand the compliance of their processes, the problem list is presented in a textual format. As is shown in Fig. 7.9, the problem list is presented as text. Thereby the text should avoid technical terms where possible.

The *problem list* is ranked based on the severity of the deviations and the strength of the relation found between the deviations and their contextual information. We will discuss ranking of problem statements in detail in Sect. 7.5.2 and Sect. 7.5.3.

Next we will explain how we build different sets of deviations and their corresponding problem statement in a problem list.

**Problem statements related to a specific activity.** As is shown in the problem list structure in Fig. 7.9, first-level problem statements in this category report the total number of violations observed for each activity. Second-level problem statements specify the type of violation, and a third-level problem statement reveals an association between the observed violation and its contextual data.

Figure 7.10 illustrates a Venn diagram based representation showing sets of deviations. These sets show how different levels of problem statements related to a specific activity are built. As is indicated both in Fig. 7.10 and Fig. 7.9, first-level problem statements in this category are generated as the intersection of moves involving activity *a*, and set of all violating moves. To compute the list of all first-level violations, we compute this intersection for each activity *a* in the log.

The second-level problem statements of this category are intersections of moves involving activity *a* and moves having a specific violation type (for instance *model-only moves*). The third level-problem statements are built as the intersection of moves involving activity *a*, moves having a specific violation type



houses was executed.

- *Second-level: Activity check off-site warehouses* was skipped 1447 times.
  - \* *Third-level: Activity check off-site warehouses* was skipped 1288 times when *amount* was more than 100000.
  - \* *Third-level: Activity check off-site warehouses* was skipped 1412 times when *inventory level* was above threshold.
  - \* *Third-level: Activity check off-site warehouses* was skipped 600 times when *material* was glass.
  - \* *Third-level: Activity check off-site warehouses* was skipped 488 times when *material* was copper.
  - \* *Third-level: Activity check off-site warehouses* was skipped 573 times when *resource* was Sara.
  - \* *Third-level: Activity check off-site warehouses* was skipped 476 times when *resource* was Chloe.
  - \* *Third-level: Activity check off-site warehouses* was skipped 398 times when *resource* was Anna.
  - \* ....
- *First-level: ...*

As is shown in the above problem list, the first-level problem statement related to *check off-site warehouses* activity reports in total 1447 number of violations for this activity. The second-level problem statement specifies the type of the violation observed for this activity together with its frequency; and the third-level problem statements reveal the associations between the observed violation and its contextual data. As activity check off-site warehouse is constrained only with one compliance rule (*Rule 6*), we see only one type of violation for this activity in the problem list, hence, only one second-level problem statement is reported for this activity. From the third-level problem statements of this activity, we only show problems statements related to four attributes *amount*, *inventory level*, and *resource*. Similarly, the problems statements of different levels for every violating activity is prepared.

The textual format of the problem list facilitates focusing on important problems and finding relations between deviations and the contextual information available in the log. For instance from the third-level statements listed in the example problem list, the second sentence reveals an unexpected association between the *inventory level* being *above threshold* and *skipping check off-site*

warehouses. We previously discussed that ‘skipped check off-site warehouses’ may only be justified if inventory level is urgent. We also observe that, three resources *Sara*, *Chloe*, and *Anna* are often being in charge when the violations of this type occurred. Similarly the coexistence of the described violation with the values more than 100,000 for attribute *amount* and values *glass*, and *copper* for attribute *material* is highlighted.

**Problem statements related to a specific (attribute,value) assignment.** In addition to describing compliance problems in terms of violating activities and their context, we propose to also group problems by their context to get an overview of problematic contexts. This gives rise to the second category of problem statements as is shown in the lower part of Fig. 7.9. These categories are based on a specific (attribute,value) assignment rather than activities.

These problem statements are built based on the intersection between set of all moves having  $v \in Val$  for their attribute  $x \in Attr$  and different sets of violating moves. Figure 7.11 illustrates how sets of deviations referring to different

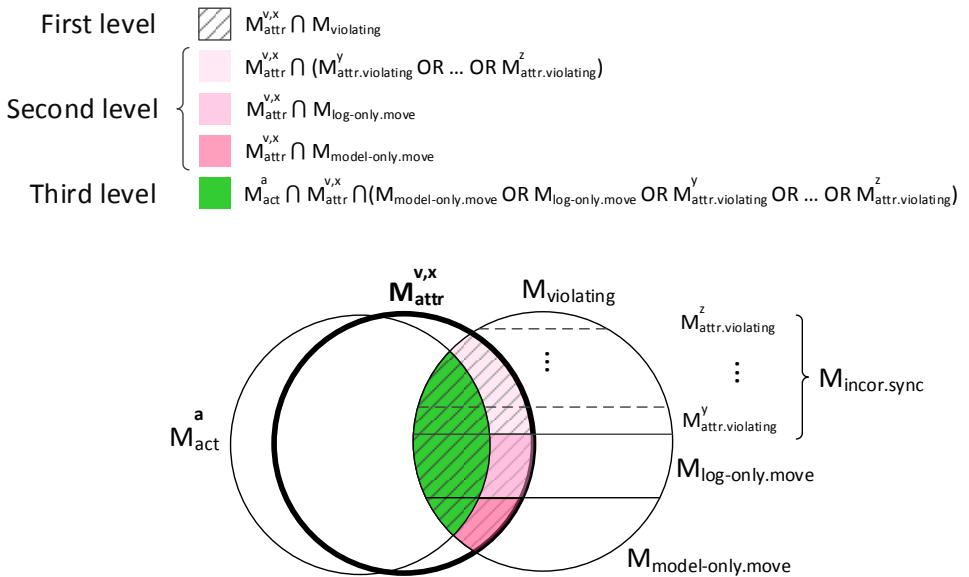


Figure 7.11: Venn diagram representation of how *attribute specific* sets of deviations referring to different problem statement levels are built.

problem statement levels in this category are built. The first-level problem statements of this category indicates the total number of violations observed when attribute  $x$  has value  $v$ . The sets, related to these problem statements, are generated as the intersection between all the moves having value  $v$  for their attribute  $x$ , and set of all violating moves. The second-level problem statements specify the type of violations observed and their frequency. Therefore, a second-level problem statement is generated as the intersection of all the moves having value  $v$  for their attribute  $x$  and moves with a specific violation either: *log-only move*, *model-only move*, or *moves with an incorrect attribute*.

Third-level problem statements of the first category and the second category are the same, they are built as the intersection of all moves having value  $v$  for their attribute  $x$  with all the moves involving activity  $a$  and having a specific violation type either.

The following list of problem statements shows all violations and their type that were observed when *material* was *glass* in our example:

- *First level*: When *material* was *glass*, 1229 violations were observed.
  - *Second level*: When *material* was *glass*, 600 times activities were skipped.
    - \* *Third level*: When *material* was *glass*, 600 times activity *check off-site warehouses* was skipped.
  - *Second level*: When *material* was *glass*, 196 times activities were executed with an incorrect *resource*.
    - \* *Third level*: When *material* was *glass*, 150 times activity *purchase requisition approval-II* was executed with an incorrect *resource*.
    - \* *Third level*: When *material* was *glass*, 65 times activity *purchase requisition approval-I* was executed with an incorrect *resource*.
  - *Second level*: When *material* was *glass*, 215 times activities were executed with an incorrect *amount*.
    - \* *Third level*: When *material* was *glass*, 65 times activity *purchase requisition approval-I* was executed with an incorrect *amount*.
    - \* *Third level*: When *material* was *glass*, 150 times activity *purchase requisition approval-II* was executed with an incorrect *amount*.
  - *Second level*: When *material* was *glass*, 215 times activities were executed with incorrect *role*.

- \* *Third level: When material was glass, 65 times activity purchase requisition approval-I was executed with an incorrect role.*
- \* *Third level: When material was glass, 150 times activity purchase requisition approval-II was executed with an incorrect role.*
- *Second level: When material was glass, 3 times activities were executed with an incorrect risk.*
  - \* *Third level: When material was glass, 3 times activity prepare purchase requisition document was executed with an incorrect risk.*
- *First level: ...*

The *problem list* at this stage can become very long specially if there are many activities and attributes present in the log. To improve its comprehensibility, all the statements in the list that refer to an empty set of moves will be removed. Moreover we rank the problem statements to make the more important problems appear earlier in the list. Next, we will explain the two ways that we rank the problem list.

### 7.5.2 Ranking Problem Statements by Domain Knowledge

We use domain knowledge to rank problem statements in the list. For this, we propose to use a user-configurable *severity* score. The score is calculated for each activity or attribute based on the importance given to a certain attribute or activity by the *user* multiplied with the frequency of the observed deviation for that activity or attribute. For instance if deviations about activity *payment* or attribute *amount* are of more importance, the *severity* metric for *payment* will increase and the deviations related to *payment* and attribute *amount* may end up in an earlier position in the problem list. For example for the list:

- Activity check off-site warehouses was skipped 1412 times when **inventory level** was above threshold.
- Activity check off-site warehouses was skipped 1288 times when **amount** was more than 10,000.

If the user chooses severity weight of attribute **inventory level** as 1 and **amount** as 4, then the entry with attribute **amount** will be shown at an earlier position than the entry with attribute **inventory level**.

### 7.5.3 Ranking Problem Statements by Context Relevance

The type of problem statements explained previously may reveal important information about the coexistence of a deviation and a particular attribute value. For instance, the statement ‘When *material* is *glass*, 600 times activity *check off-site warehouses* was skipped’ indicates that the deviation ‘*skipped check off-site warehouses*’ and the (attribute,value) assignment ‘*material is glass*’ coincide 600 times; hypothesizing a relation between them.

We can consider such a relation as an association rule of the form  $X \Rightarrow Y$ , with  $X$  being the *antecedent* of the rule and  $Y$  being the *consequent* of the rule. The *antecedent* or *consequent* of the rule can be a violation e.g. ‘*activity check off-site is skipped*’, or an (attribute,value) assignment e.g. ‘*material is glass*’. Many of such rules can be found after checking compliance of a dataset. We can leverage various metrics proposed in association rule mining literature to evaluate the relevance of a rule and thereby the relevance between a violation and an (attribute,value) assignment.

To evaluate an association rule from business perspective, it is important to understand (1) the direction of the association and (2) how much more often, the antecedent and consequent of the rule co-occur than expected if they were statistically independent. The (1) is important if we want to use association rules for predicting possible violations in the future. For instance, if a specific (attribute,value) assignment implies a violation, we can predict that future executions of activities with that (attribute,value) assignment are probably violating. The latter is important because it help us detect infrequent but important associations.

As mentioned before, several interest measures are defined in literature. In the following we compare some of them w.r.t. the two requirements we have. We will discuss support [12], confidence [12], Lift (interest) [22], and we finally choose the *Conditional Probability Increment Ratio* (CPIR) metric defined in [152] to quantify the relevance between violations and their contextual information available in the event log. First we establish some notation.

Let  $I = \{i_1, i_2, \dots, i_n\}$  be a set of distinct elements called *compliance elements*. An example of a *compliance element* could be (*check off-site warehouses,skipped*). A *compliance element*  $i \in I$  is a pair  $(x, v)$  with  $x \in Attr$ , and  $v \in Val \cup \{\text{skipped, not allowed, executed, violating attribute}\}$ .

In the following, we will discuss finite (sub)sets  $A, B \subseteq I$  of compliance elements;  $|A|$  is the *size* of  $A$  (i.e., the number of elements in  $A$ ). Further, we call a set  $\mathcal{C} = \{A_1, A_2, \dots, A_k\} \subseteq 2^I$  of sets of compliance elements, a *compliance element collection* of size  $k$ .



Intuitively,  $\mathcal{C}$  corresponds to a, say, second-level problem statement  $p$  and each  $A_i \in \mathcal{C}$  corresponds to the (attribute,value) pairs of third-level problem statements under  $p$ . We define

$ce : M \rightarrow \{(x, v)\}$  is a function that maps a move  $m \in M$  to its *compliance elements*.

- For  $m = (e^L, \gg) \in M_{\log\text{-only.move}}$ :

$$ce(m) = \{(\#_{act}(e^L), \text{not allowed})\} \cup \{(x, \#_x(e^L)) \mid x \in \text{dom}(\#(e^L)) \setminus \{act\}\}.$$

- For  $m = (\gg, e^R) \in M_{\text{model-only.move}}$ :

$$ce(m) = \{(\#_{act}(e^R), \text{skipped})\} \cup \{(x, \#_x(e^L)) \mid \exists m' = (e^L, e^R) \in M \wedge$$

$$e^L \not\gg \wedge m' \leq^M m \wedge \#_{pi}(e^L) = \#_{pi}(e^R) : \forall x \in \text{dom}(\#(e^L)) \setminus \{act\}\}$$

That is, if an event is skipped,  $ce(m)$  maps move  $m = (\gg, e^R)$  to the (attribute,value) pairs of an event ( $e^L$ ) that precedes move  $m'$ . In our implementation we pick the  $e^L$  that occurred last before the model-only move  $m$ , and

- For  $m \in M_{y,attr.violating}$ , let  $A_{attr.violating} = \{y \in Attr \setminus \{act\} \mid (y \in \text{dom}(\#(e^L)) \wedge y \notin \text{dom}(\#(e^R))) \vee (y \in \text{dom}(\#(e^R)) \wedge y \notin \text{dom}(\#(e^L))) \vee (y \in (\text{dom}(\#(e^L)) \cap \text{dom}(\#(e^R))) \wedge (\#_y(e^L) \neq \#_y(e^R)))\}$ , be the attributes of move  $m$  where  $e^L$ , and  $e^R$  differ in some form. Then:

$$ce(m) = \{(\#_{act}(e^L), \text{executed})\} \cup \{(y \in A_{attr.violating}, \text{violating})\} \cup$$

$$\{(x, \#_x(e^L)) \mid x \in \text{dom}(\#(e^L)) \setminus \{act\}\}.$$

The choice of the compliance elements  $ce(m)$  of a model-only move may require adjustments or careful log preparation to deliver meaningful results. When an event is missing, there is no event-level attribute for the skipped event to be considered as context of the violation. The challenges and possible solutions for dealing with missing events were discussed already in Sec. 5.2. We can use trace-level attributes and contextual data available for other events in the same process instance (for instance neighbouring events) as a context of the skipped event. In order to do so, we have two decisions to make: (1) we need to choose the event(s) that can be considered as context of the missing event, and (2) we need to choose attributes that need to be considered relevant for analyzing the skipped event. Domain knowledge is required to make aforementioned choices. However, next we elaborate on consequences of the choices we make in these two decisions.

- **Choice of events.** Neighbour events (predecessor and/or successor) of an skipped event, are the closest context to be considered. Yet, domain knowledge and definition of the compliance rule can help us choose different relevant event(s) as a context for an skipped event. As mentioned earlier, use attributes of the direct proceeding event of a skipped event as its context. However, our approach can be extended to allow users to choose the relevant event(s) as the context of a skipped event. In case a number of events are chosen, we should make sure that they do not have common event-level attributes with different values.
- **Choice of attributes.** Suppose we have chosen the relevant event(s) to be considered as the context of a skipped event. We also need to choose the attributes of these events that should be considered as a context for the skipped event. Trace attributes can easily be included as the context of a skipped event since they have a fixed value throughout a process instance. If event-level attributes hold a fixed value for a sequence of events, they should be included in context of a skipped event. For instance attributes such as *amount* or *resource* are as such. There are some attributes which are very local to a specific event, for instance *dose of a medicine* that is administered to a patient, or *characteristics of the machine* that produced an event. In case of such attributes, it is better not to include them in the context because they are irrelevant for the skipped event and most probably they will not add much information. Yet, domain knowledge can also be very useful in choosing relevant attributes.

No matter which event(s) and which of their attributes are included in context of a skipped event, we should be aware that the possible associations we find between these context data and the the specific violation are actually between the violation and attributes of the relevant event(s).

#### Computing relevance of an association rule.

An alignment  $A = (L, R, M, \leq^M)$  yields the compliance element collection  $\mathcal{C}(M) = \{ce(m) | m \in M\}$ . We take the intersection of move sets used for the problem lists, and then compute compliance elements of these moves. Each third-level problem statement, such as “*activity a was skipped n times when x = v*”, gives rise to a possible association rule over compliance elements, such as  $R \equiv (a, not\ allowed) \rightarrow (x, v)$  or  $(x, v) \rightarrow (a, not\ allowed)$ . To test whether a rule  $X \rightarrow Y$  holds over alignment  $A$ , we check different measures.

---

$A_1 = \{a,b,d\}$
$A_2 = \{a,b,c,d\}$
$A_3 = \{b,d,f\}$
$A_4 = \{b,c,d,f\}$
$A_5 = \{a,f\}$
$A_6 = \{d,f\}$
$A_7 = \{a,b,e\}$
$A_8 = \{c,f\}$
$A_9 = \{e,f\}$
$A_{10} = \{a,b,c,d,e\}$

---

Table 7.4: Sets of compliance elements in compliance element collection  $\mathcal{C}$ .

### Support.

For a *compliance element collection*  $\mathcal{C} \subseteq 2^I$  and a set of *compliance elements*  $A \subseteq I$ , the support of  $a \in I$  is the fraction of sets in  $\mathcal{C}$  containing  $a$ :  $\text{supp}(a) = \frac{|\{A \in \mathcal{C} \mid a \in A\}|}{|\mathcal{C}|}$ .

For instance if  $I = \{a, b, c, d, e, f\}$  and  $\mathcal{C} = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}\}$ , with Table 7.4 showing compliance elements in different sets in  $\mathcal{C}$ , then  $\text{supp}(a) = \frac{5}{10}$ , and  $\text{supp}(b) = \frac{6}{10}$ .

Similarly the support of a rule of the form  $a \rightarrow b$  is the fraction of sets in the compliance element collection  $\mathcal{C}$  containing both  $a$  and  $b$ , i.e.,  $\text{supp}(a \rightarrow b) = \frac{|\{A \in \mathcal{C} \mid a \in A, b \in A\}|}{|\mathcal{C}|}$ . In the example of Table 7.4:  $\text{supp}(a \rightarrow b) = 0.4$ .

As support counts the occurrences of an association rule, it is often called a frequency constraint. We can say a rule with a support greater than a specified minimum support threshold is a *frequent* rule, hence can be considered important. The disadvantage of support is the so called *rare item problem*. If the antecedent and the consequent of a rule are infrequent, they will be pruned, although they may produce potentially valuable rules. In addition, *support* is not directed, i.e.,  $\text{supp}(a \rightarrow b) = \text{supp}(b \rightarrow a)$ .

### Confidence.

Confidence is defined as the probability of seeing a rule's consequent under the condition that the dataset ( $\mathcal{C}$ ) also contains the antecedent. For instance,  $\text{conf}(a \rightarrow b) = \frac{\text{supp}(a \rightarrow b)}{\text{supp}(a)} = \frac{0.4}{0.5} = 0.8$ .

Association rules need to satisfy a minimum confidence constraint to be considered important. Unlike support, confidence is directed and gives different

values for the rules  $a \rightarrow b$  and  $b \rightarrow a$ . Confidence is sensitive to the frequency of the consequent in  $\mathcal{C}$ . In particular consequents with higher support will automatically produce higher confidence as well. For instance a resource that executed most of the activities will be found guilty for violating ones as well.

### Lift.

Lift of an association rule is the ratio between the confidence of the rule and the support of its consequent. For instance, in Table 7.4  $lift(a \rightarrow b) = \frac{0.8}{0.6} = 1.3$ . Lift measures how many times more often antecedent, and consequent occur together than expected if they were statistically independent. Lift is not directed, i.e.,  $lift(a \rightarrow b) = lift(b \rightarrow a)$ .

### Conditional probability increment ratio.

Wu et al. have proposed the *conditional probability increment ratio* (CPIR) [152] that assesses whether two entities (in our case two compliance elements) are positively or negatively related.

For instance, in Table 7.4  $CPIR(a \rightarrow b) = \frac{supp(a \rightarrow b) - supp(a)supp(b)}{supp(a)(1 - supp(b))} = \frac{0.4 - (0.5 * 0.6)}{0.5 * (1 - 0.6)} = 0.5$ .

If  $supp(rule) > supp(consequent)supp(antecedent)$ , then consequent and antecedent are positively dependent and they occur more often than expected if they were statistically independent and in case CPIR is larger than a user set threshold the relation is considered important.

If  $supp(rule) < supp(consequent)supp(antecedent)$ , then the consequent and the antecedent are negatively dependent. Negative associations may be interesting, for instance if we find out that a certain *violation* seldom occurs for a specific *material* or absence of a specific *resource* usually implies that a particular *violation* does not hold.

In our example of Table 7.4,  $CPIR(a \rightarrow f) = \frac{0.1 - (0.5 * 0.6)}{0.5 * (1 - 0.6)} = -1$ . In this case  $a$  and  $f$  are negatively dependent, or  $a$  and  $\neg f$  are positively dependent, then  $a \rightarrow \neg f$  indicates that whenever  $a$  is present we expect  $f$  to be absent.

Although negative associations can reveal important information, the interpretation of negative associations are more difficult than positive associations domain and conclude whether they convey a *meaningful* business insight (event for domain experts). Hence, we are more interested in finding association rules with *positive* CPIR.

*Positive* associations with a CPIR more than a fixed upper threshold, and *negative* associations smaller than a fixed lower threshold are kept to guide

users to focus on more important violations.

Unlike lift, CPIR is a directed measure, since it also uses the information of the absence of the consequent. As we mentioned earlier, this is important if we want to use association rules with higher CPIR value for predicting possible violations in the future.

In essence we can use any of the interest measures discussed above such as support, confidence, Lift, and many more including all-confidence [89], collective strength [11], or conviction [22], however we utilized CPIR in our approach. In addition to the direction of an association, utilizing CPIR, we can also find infrequent and unexpected rules.

We use alignment moves to build compliance element collections. Given an alignment  $A = (L, R, M, \gg^M)$  and the compliance element collection  $\mathcal{C}(M)$ , we test the relevance of a rule of the form  $a \rightarrow b$  over  $\mathcal{C}(M)$  using its CPIR value:

$$CPIR(a \rightarrow b, \mathcal{C}(M)) = \frac{supp(a \rightarrow b, \mathcal{C}(M)) - supp(a, \mathcal{C}(M))supp(b, \mathcal{C}(M))}{supp(a, \mathcal{C}(M))(1 - supp(b, \mathcal{C}(M)))}$$

### Filtering Association Rules

Recall that we aim at presenting the user a list of relevant compliance violations. In Sect 7.5.1, we produced lists of hierarchical problem statements, where the third level problem statements combine a particular kind of violation with a context attribute. To give the user only relevant problem statements, we prioritize and filter this list as follows. For each third level problem statement “violation  $V$  occurred  $K$  times when  $C$ ” we generate rules “ $context \rightarrow violation$ ” (in short  $C \rightarrow V$ ) and “ $violation \rightarrow context$ ” (in short  $V \rightarrow C$ ) and compute their CPIR values over all compliance elements of the alignment. If the CPIR value is below a user-chosen threshold, we remove the problem statement in the list. The remaining problem statements can further be ordered by their CPIR value to show stronger rules more prominently.

Table 7.5 shows the associations we found between violation *activity check off-site was skipped* and attribute *inventory level* together with their computed CPIR value and direction. As can be seen each of the problem statements have two CPIR values depending on the direction of the association.

In the problem list we obtained for our procurement example, in total we found 380 positive associations. If we increase the minimum CPIR for positive associations from 0 to 0.5, then we will obtain about 45 associations. In our example we found several negative associations. However, we could not give

Violation	Frequency	CPIR	Direction
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>above threshold</i>	1412	0.962	V → C
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>above threshold</i>	1412	0.048	C → V
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>threshold</i>	5	0.423	V → C
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>threshold</i>	5	0.029	C → V
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>Urgent</i>	30	0.242	V → C
Activity <i>Che.</i> was skipped when <i>IL</i> is <i>Urgent</i>	30	0.026	C → V

Table 7.5: CPIR value and direction of some of the associations between ‘*activity check off-site was skipped*’ and its context.

a meaningful business interpretation for them. Therefore, we only considered positive associations and discarded the negative ones.

## 7.6 Investigating Specific Problems

In Sect. 7.5, we used the CPIR values of association rules “*context* → *violation*” or “*violation* → *context*” to rank problems in their importance. The context information was limited to concrete (attribute,value) pairs. However, the context of a violation may be larger and include multiple values for the same attribute (e.g. price or age) and also comprise multiple attributes (e.g. product and resource). To identify such richer context of violations, we translate the problem of root-cause analysis of a specific deviation to the problem of building a decision tree [134] to find out how compliant and non-compliant events differ from each other w.r.t. the information available in the event log.

### 7.6.1 Decision Trees for Compliance Violations

In decision tree learning [134], we choose one *response variable* and some *predictor variables* and classify instances of data to possible values of the response variable. In our example, we choose the deviation *skipped check off-site warehouses* as the response variable and some other attributes available in the event log such as material and inventory level as predictor variables. Figure 7.12 illustrates a possible decision tree. The tree describes under which values for

material and inventory level, the activity check off-site warehouses faces a violation.

Each non-leaf value (also the root value) is labeled with a predictor variable. The outgoing arcs are labeled with values (singleton values or intervals) of the predictor variable. Each path from the root node to a leaf node describes a *complete* assignment of values to all predictor variables. Each leaf node describes the value of the response variable under the chosen assignment of the predictor. In Fig. 7.12, activity check off-site warehouses is skipped when *inventory level = above threshold* and *material = copper* or *material = glass*; activity check off-site warehouses is not skipped for all other assignments to the predictor variables.

In the context of classifying moves of an assignment, the tree can also be read differently. The root node of the tree of Fig. 7.12 represents all the 3714 moves involving activity check off-site warehouses. The outgoing arcs of any non-leaf node partition this set of moves based on the value of the predictor

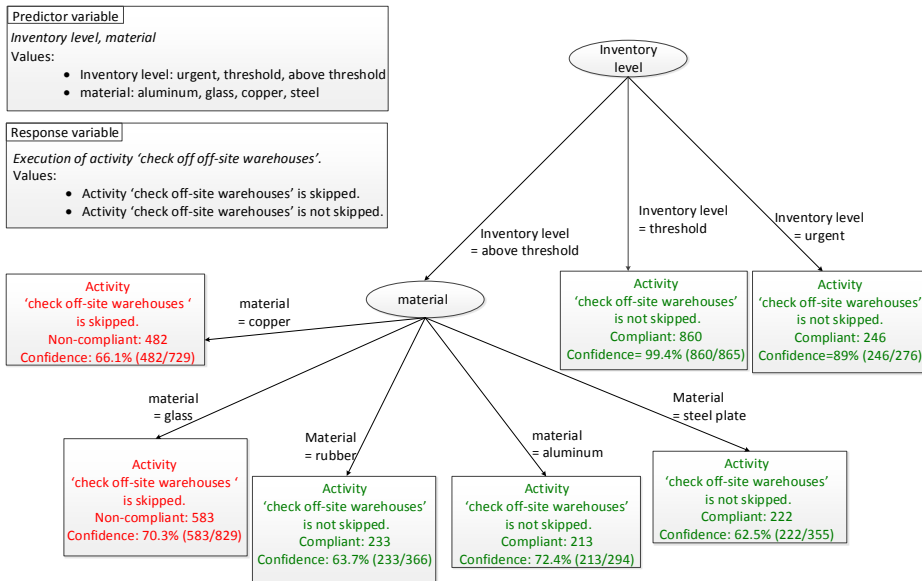


Figure 7.12: A decision tree classifying moves to those skipped *check off-site warehouses* and those did not, considering inventory level and material as predictor variables.

variable of the non-leaf node. For instance in Fig. 7.12, the edges leaving the node “inventory level” partition the 3714 moves into 276 moves where inventory level was *urgent*, 865 moves where the inventory level was *threshold*, and 2573 moves with inventory level *above threshold* that are further partitioned by the material variable.

### 7.6.2 Building Decision Trees from a Problem Statement

The decision tree explained in Sect. 7.6.1 provides context conditions that distinguish whether moves that involve activity *check off-site warehouses* have been skipped or not. Next, we discuss how to build such a decision tree for any of the problem statements in the problem list of Sect. 7.5.

Each problem statement refers to violations in a particular class  $M_{cl}$  of moves characterized by either (1) a particular activity  $M_{a,act}$ , (2) a particular attribute-value combination  $M_{v,x,attr}$ , or (3) both  $M_{a,act} \cap M_{v,x,attr}$ . Where the problem statement identifies only the violating moves in  $M_{cl}$ , the decision tree shall consider and distinguish *all compliance statuses* (executed, skipped, not allowed, violating attribute) of all moves in  $M_{cl}$ .

Thus, we build the decision tree learning problem as follows. The *compliance status* is the response variable  $x_0$ , the user chooses the predictor variable(s)  $x_1, \dots, x_k$  from the available attributes in the log. Each move  $m \in M_{cl}$  defines an *observation instance*  $oi(m) = (v_1, \dots, v_k, c)$  where  $(x_i, v_i) \in ce(m)$ ,  $i = 1, \dots, k$  are the values of the context attributes of move  $m$ , and  $c$  the compliance status of  $m$  derived as follows:

- $c = \textit{skipped}$  if  $(a, \textit{skipped}) \in ce(m)$  and  $M_{cl} = M_{act}^a$  or  $M_{cl} = M_{act}^a \cap M_{attr}^{v,x}$
- $c = \textit{not allowed}$  if  $(a, \textit{not allowed}) \in ce(m)$  and  $M_{cl} = M_{act}^a$  or  $M_{cl} = M_{act}^a \cap M_{attr}^{x,v}$
- $c = \textit{violating}$  if  $(x, \textit{violating}) \in ce(m)$  and  $M_{cl} = M_{attr}^{v,x}$  or  $M_{cl} = M_{act}^a \cap M_{attr}^{v,x}$
- $c = \textit{executed}$  otherwise.

The set  $oi(M) = \{oi(m) | m \in M\}$  of all observation instances can be encoded as a table and given as input to a decision tree learning algorithm. Various algorithms exist to build a decision tree that best describes the values of response variables (compliance of a move) based on values of predictor variables (context attributes). The most popular listed also in [151] include CHAID [62],



ID3 [94], CART [21], and C4.5 [95]. In our approach, we use C4.5 [95] to build decision trees but this could be replaced by any other method.

This way, the user can simply click on a problem statement in the list and learn a decision tree for a chosen set of context variables. The choice of variables for building a decision tree is discussed in the next section. In case the problem statement refers to both an activity and an attribute-value combination, we automatically compute decision trees for all three move classes  $M_{act}^a \cap M_{attr}^{v,x}$ ,  $M_{act}^a$ , and  $M_{attr}^{v,x}$  as the tree for most restrictive class only  $M_{act}^a \cap M_{attr}^{v,x}$  may not give enough insights.

In case the user chooses a timestamped attribute  $z$  as context attribute, the absolute time value  $v_z$  of  $z$  might not produce enough discriminative power. In these cases, we allow to pick an *anchor* activity  $a$  in the log whose timestamp value  $v_a$  is then used to produce the *relative timestamp*  $v_z - v_a$  as value for  $z$  in the observation instance, see [54] and [130].

The learned decision tree usually gives an approximation of the classification as the predictor variables typically do not capture the complete context of the response variable in the given data. The reliability of the classification is indicated by confidence values in each leaf node and is computed as follows. For classifying moves to a value of response variable (i.e., violating or compliant), the classification technique employed divides the data into a training set and a test set. The decision tree is learned using the training dataset. Then, the data in the test set is used to check how precise the trained tree can predict to which value of response variable, an observation is assigned based on the values of its predictor variables. The ratio between the correctly classified moves and the moves that are wrongly classified represents how precise a trained decision tree is. For example, in Fig. 7.12, from the 276 moves with inventory level being urgent, 246 were classified correctly as *compliant* and (276-246) were wrongly classified. Consequently the *confidence measure* is 89% for this leaf node. Similarly, the confidence measure for other leaf nodes is computed.

### 7.6.3 Using Decision Trees to Analyze the Cause of a Violation

The resulting decision tree reveals patterns (i.e., combination of values of predictor variables) under which a particular violation holds or does not hold. These patterns not only can be used for predicting future violations but we can use them to understand violations better. Next, we showcase the steps that can be taken to first of all detect possible compliant and violating patterns and

also to analyze them further to better understand cause(s) of violations.

### Choosing variables

The choice of predictor variables influences whether one is able to get insights about violations. In practise we may have many context attributes for a violation and consequently many combination of attributes can be chosen to learn a decision tree. An option would be to use a feature selection technique [45, 90, 154] to choose groups of attributes which are dependent. However, domain knowledge should also be considered as an important source of information for choosing interesting combination of predictor variables.

Even in absence of domain knowledge we can use the important and unexpected associations identified in the problem list (see Sect. 7.5) for choosing a set of predictor variables. For instance in the problem statements listed for *activity check off-site warehouses* (shown in Sect. 7.5), we found interesting associations between *activity check off-site warehouses is skipped* and attributes *inventory level*, and *material*. Therefore, we used these two attributes as predictor variables in the decision tree shown in Fig. 7.12. As is shown in this figure, *activity check off-site warehouses is skipped* when inventory level is above threshold and material is glass or copper and this activity is not skipped for other values of material and inventory level. Such violating and compliant patterns can be used for predicting possible future violations and the conditions that a particular violation most probably will hold or not.

### Use insights and combine it with other knowledge.

The insights obtained from this decision tree could then be used for further analysis. Suppose the analyst knows that the company always orders copper and glass from the same supplier. Given that variables primarily coincide with these two materials, one could investigate the relation of the supplier with violations in another decision tree analysis.

In general, results of one analysis may trigger a “creative process” for doing further analyses. The follow-up step can be another decision tree analysis with choice of new predictor variables or other types analysis. For instance, another way to help collecting/gaining insights is to compare compliant and non-compliant behavior.

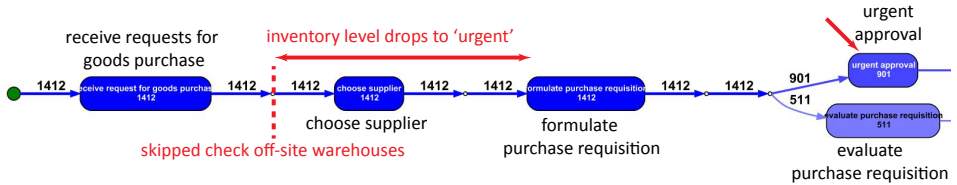


Figure 7.13: Part of the process model discovered from violating cases with inventory level above threshold.

### Comparing compliant and violating patterns.

After detecting the violating and compliant patterns (learned by a decision tree), we can filter them and analyze them further. For instance we can filter out all the cases that contain a violating pattern and discover the process model that describes their behavior best using any of the process mining algorithms [23,56,70,148]. Similarly we can mine the process model that describes the compliant pattern. Comparing the two models helps us understand how violating and compliant patterns differ in behavior.

Based on the decision tree shown in Fig. 7.12 of our example, we can filter out cases that have a violation of type “skipped check off-site warehouses” and have an “inventory level=above threshold”. This allows to discover the process model that describes these cases for example using the Inductive visual Miner technique [70]. Suppose the resulting model (partially shown in Fig. 7.13) shows us that activity *urgent approval* has been executed in 901 cases of the 1412 violating cases with ‘inventory level=above threshold’.

In contrast to the case attribute “*inventory level = above threshold*”, execution of activity *urgent approval* implies that ‘inventory level’ has the value *urgent* at the time this activity is executed (assuming all of these cases are compliant with *compliance rule 4*). This observation would suggest that the value of attribute ‘inventory level’ changed in each of those cases and dropped to *urgent* level somewhere between *check off-site warehouses* and *formulate purchase requisition* (illustrated in Fig. 7.13), where the decision for following an *urgent approval* procedure is taken. Knowing that urgent approval procedure is risky and it must be followed only in critical situations, this observation gives us a concrete point in the process that needs to be improved to make it compliant.

‘*Get Problem Insight*’ component of the *Compliance Framework* (Fig. 7.3), enables us to do the type of analysis we showcased above for the procurement example. The results of this type of analysis in our approach are presented as

a decision tree or a Venn diagram. Providing these diagnostics help auditors to hypothesize about deviations. Note that obtaining these diagnostics is not possible by solely using current techniques in compliance checking [24, 33, 85, 97, 109, 111, 130, 135].

## 7.7 Implementation and Case Study Results

The presented technique has been implemented in the form of two plugins of the *ComplianceFramework* package in ProM 6.6. The “*Compliance Dashboard*” plugin (see Fig.7.15 for a screenshot) takes a log and a process model as input, it checks compliance of the event log w.r.t. the model, and it returns *statistics tables*, *bar charts* and the *problem list*. The *statistics tables* provide the possibility to get information about deviations from different perspectives and as a whole. The *bar charts* gives a detailed view on deviations and the *problem list* highlights important deviations and their relation with the context information.

By running the second plugin “*Get problem insight*”, the user can pick a specific violation for a deeper analysis by choosing the respective *problem statement* from the *problem list*. Based on the selected *problem statement*, the user needs to choose different attributes available in the event log for further analysis. The result of this analysis is a decision tree as described in Sect. 7.6.1 together with a Venn diagram elaborating on the conditions that lead to a specific deviation. Several figures in this section illustrate the output of each plugin.

### 7.7.1 Setup of the Experiment

We have applied our technique and its implementation to several real life event logs with the goal to test its functionality in detecting dependencies between contextual information of a violation and the violation and its usability for business users. Here, we focus on a case study we did in collaboration with *Price-waterhouseCoopers* (PwC) for checking compliance of a purchase process in a manufacturing company. Our results shows that we are able to detect violations of different types and find several associations between these violations and their contextual information. Our analysis also reveals patterns that differentiate violating activities and compliant ones w.r.t. different violations. In addition our analysis shows that business users are able to read and interpret the root-cause analysis results as intended.

We collected an event log of 77004 cases containing 1036865 events for this process. Each case contains the events related to a purchase entry. In this event

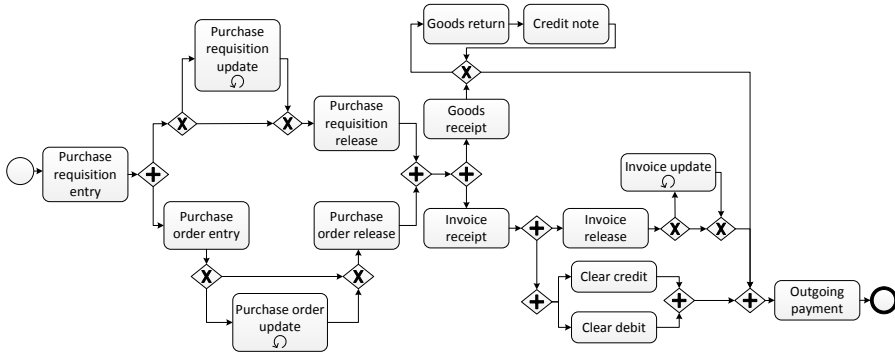


Figure 7.14: The compliant procurement process analyzed for experimental results.

log, executions of this process between Jan. 2012 to Jan. 2013 are recorded. We modelled this process and relevant compliance rules in terms of a data-aware Petri net. To ease the presentation, a simplified model (in BPMN) describing this process is shown in Fig. 7.14.

The process starts with a *purchase requisition entry*, which may be updated several times. After the purchase requisition entry, a *purchase order* is created based on it. A purchase order can be *updated* multiple times as well. When a purchase order is *released*, it is considered to be approved. *Goods receipt* and *invoice receipt* may be executed in parallel. Each purchase order must have at least one goods receipt and invoice receipt. After invoice receipt, clearing the balance sheet should be done eventually by executing *clearing debit* and *clearing credit*. If some of the goods are returned due to quality issues, activity credit note is executed to fix some possible errors in invoices and goods receipt. The process should end with the *outgoing payment*. We checked the control-flow of this process as presented in Fig. 7.14.

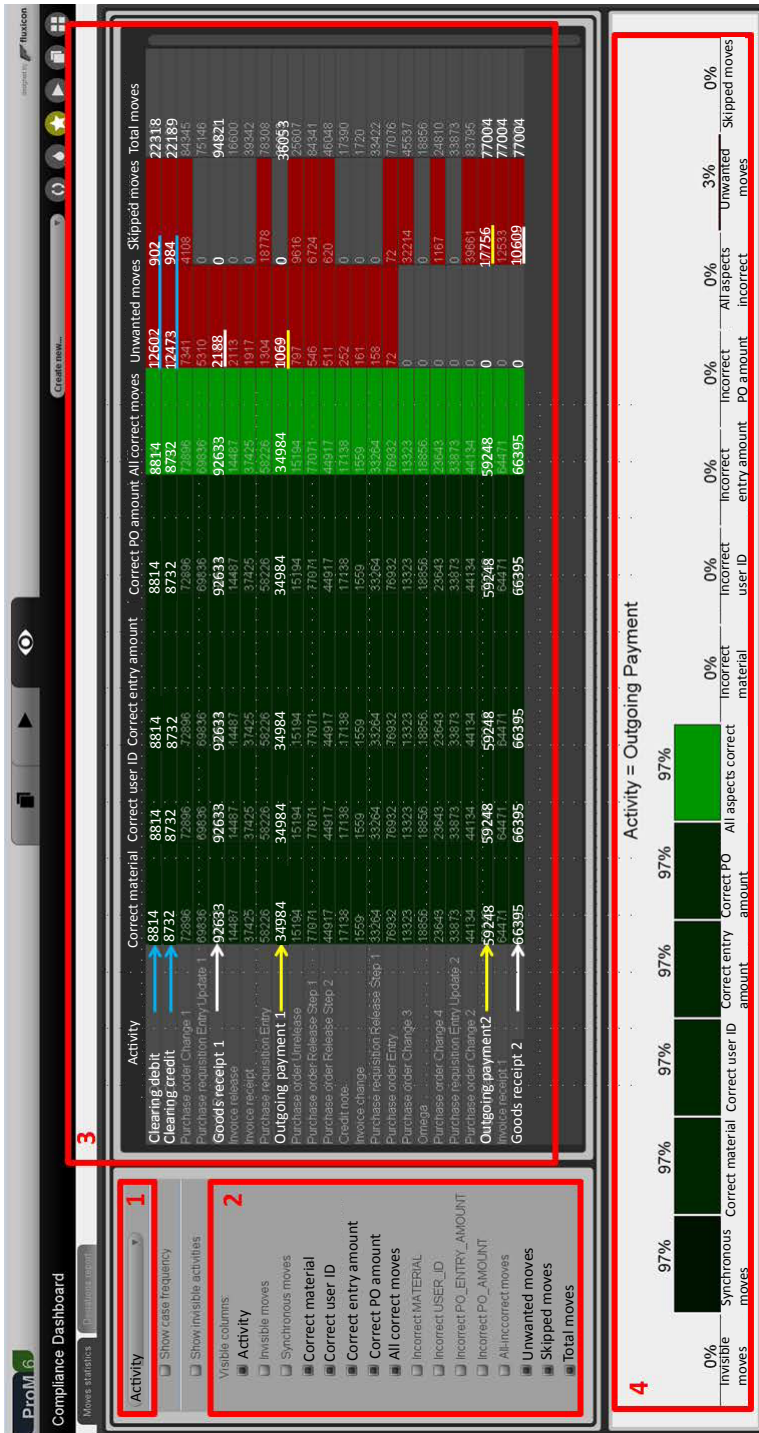


Figure 7.15: Activity deviations statistics table of experimental result.

## 7.7.2 Obtained Diagnostics

After checking compliance of the event log w.r.t. the model of Fig. 7.14. We visualized the resulting alignment in the compliance dashboard. Figure 7.15 shows the statistics overview showing activity deviations and details of the statistics related to activity *outgoing payment*.

The combo box in Fig. 7.15 (indicated with red rectangle 1) shows the attribute for which the statistics are presented. We have chosen attribute *Activity*, hence the statistics table is computed for every activity. If the user chooses a different attribute *X*, the table will show statistics related to attribute *X*. We can choose the violating and non-violating attributes (in red rectangle 2) that we would like to see in the statistics table (rectangle 3). Several attributes are selected including: *correct material*, *correct user ID*, *correct entry amount*, *correct PO (purchase order)*, and *amount* (See Fig. 7.15 rectangle 2). In addition we chose to see some move types. Here, we selected violation types *unwanted moves*, and *skipped moves* to be shown in the table. With this configuration of the dashboard shown in Fig. 7.15, the activity names are shown in the left side of the statistics table and every row contains numbers of different violation types related to each activity.

**Observations related to activity *outgoing payment*.** The statistics table in Fig. 7.15 (rectangle 3), shows that activity *outgoing payment* (marked with yellow arrows)<sup>3</sup>, was executed 1069 times where it was not allowed and it was skipped 17,756 times. The lower part of Fig 7.15 (rectangle 4) shows detailed statistics for activity *outgoing payment*. Some of the skipped *payments* are due to unfinished cases and some because they occurred earlier in the process than they were supposed to.

The compliance dashboard allows us to filter cases having a specific violation. We filtered cases containing both type of violations (unallowed *outgoing payments* and skipped *outgoing payments*). The filtered cases can be exported into an event log and then be analyzed on their own. For instance using ProM Fuzzy Miner [56], we obtain a model containing the fragment shown in Fig. 7.16. As can be seen, *outgoing payment* has occurred before activities *purchase order change*, *goods receipt*, and *invoice change* (indicated by numbered red edges). These sequences (i.e., occurrence of *outgoing payment* before these activities) are not allowed based on the specified behavior.

**Observations related to activity *goods receipt*.** Activity *goods receipt* (marked with white arrows in Fig 7.15 rectangle 3) was executed 2188 times where it

<sup>3</sup>executions of activity *outgoing payment* is divided into two parts

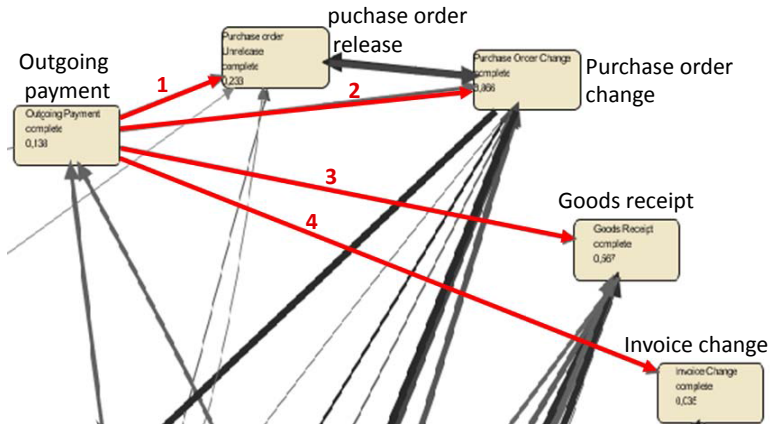


Figure 7.16: Parts of the model describing cases containing violating activity *outgoing payment*.

was not allowed and the activity was skipped 10,609 times<sup>4</sup>. If we filter traces containing unallowed executions of *goods receipt*, then we see the violating pattern shown in Fig. 7.17. This pattern reveals the occurrence of activity *credit note* before *goods receipt* (the edges lead to execution of activity *goods receipt* are numbered and colored in red). However, activity *credit note* should be executed to correct the purchase order **after** some parts of the purchased good are returned to supplier due to quality reasons. This is an interesting violation to look into, because it shows that the purchase order value changed before goods receipt.

**Observations related to activities clearing credit and clearing debit.** Another observation from the statistics table in Fig.7.15 is related to the activities *clearing debit* and *clearing credit* (marked with blue arrows). They were executed respectively 12,602, and 12,473 times while not allowed and they were skipped 902, and 984 times. When we filtered violating cases with skipped *clearing credit*, we observed that this activity was never executed for these cases whereas *clearing debit* was executed for some of them. *Clearing credit* and *clearing debit* are executed to balance accounts receivables and accounts payable after activity *invoice receipt*. These two activities must occur together. Therefore, the occurrence of one without the other one is not allowed and requires

<sup>4</sup>Executions of activity *goods receipt* is shown in two parts in the table.



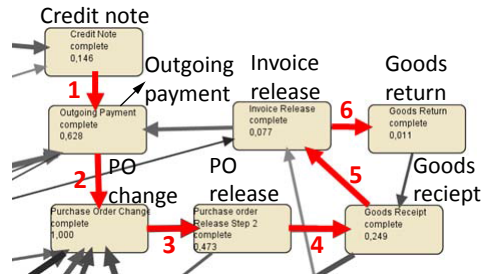


Figure 7.17: Part of the model describing cases containing unallowed executions of activity *goods receipt*.

further investigation.

For those cases that contained unallowed occurrences of *clearing credit* and *clearing debit*, we observed that these activities have occurred before *invoice receipt* (sequences that were not allowed). This pattern can be seen in Fig 7.18 (numbered red edges show the incorrect sequence of activities in the detected pattern).

**Observations related to resources using problem list.** This analysis also resulted in a long *problem list* of violations shown partially in Fig. 7.19. The list shows that in total 13,457 violations were observed for activity *clearing credit*. Out of all violations observed for this activity, 984 times activity *clearing credit* was skipped. The *problem list* raises the hypothesis that *purchase order amount* may be related to the observed violation; because we can see that most of the violations of this type occurred for values less than 120,000.

We know from the process that 552 different resources participated in executing the process under analysis. According to the *problem list* shown partially in Fig. 7.20, it seems that from the total number of resources (552), some specific resources are frequently linked to violations.

**Focused analysis on resources using Get Problem Insight.** We continued our analysis by investigating the resources that were involved in violations related to activities *clearing debit*, *clearing credit*, and *outgoing payment*. The results of the decision tree learning are shown shown in Fig. 7.21 (left). We found with a relatively good confidence that *Resource 14* and *Resource 37* have been present in considerable number of violations where execution of activity *clearing debit* was not allowed. In this analysis we classified violating and compliant executions of activity *clearing debit* based on attribute resource. That is, the

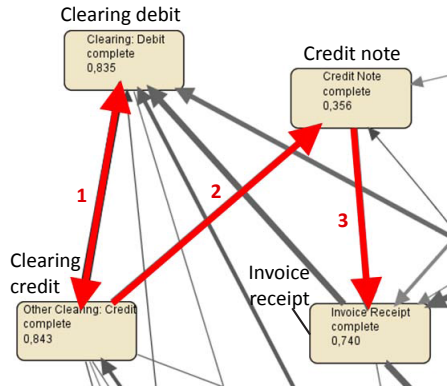


Figure 7.18: Part of the model describing violating cases where activities *clearing credit* and *clearing debit* occurred before *invoice receipt*.

chosen *response variable* is the execution of activity *clearing debit*, the target classes are *execution of clearing debit was not allowed*, and *execution of clearing debit is allowed*, and the *predictor variable* is resource.

The same analysis for unallowed executions of activity *clearing credit* shown in Fig. 7.21 (right), reveals that *resource 37* is present in large number of violations of this type as well. Furthermore we can see that many compliant executions of both activities are done by *Resource 6*. Our root-cause analysis for activity *outgoing payment* did not show meaningful connections between violating executions of this activity and a specific resource.

## 7.8 Improving the Compliance Framework Based on User feedback

We have tested the functionality of *Compliance Framework* several times and improved it based on the results of these tests using both real-life logs and artificial event logs where we had a priori knowledge about violations and correlations between violations and contextual information available in the logs. For details of these evaluations, see [54]. The functionalities of the *Framework* include finding violations of different types, providing statistics about them, and enabling root-cause analysis on violations. Besides the functionalities of the *Framework*, one of the main goals in this chapter is addressing the needs of

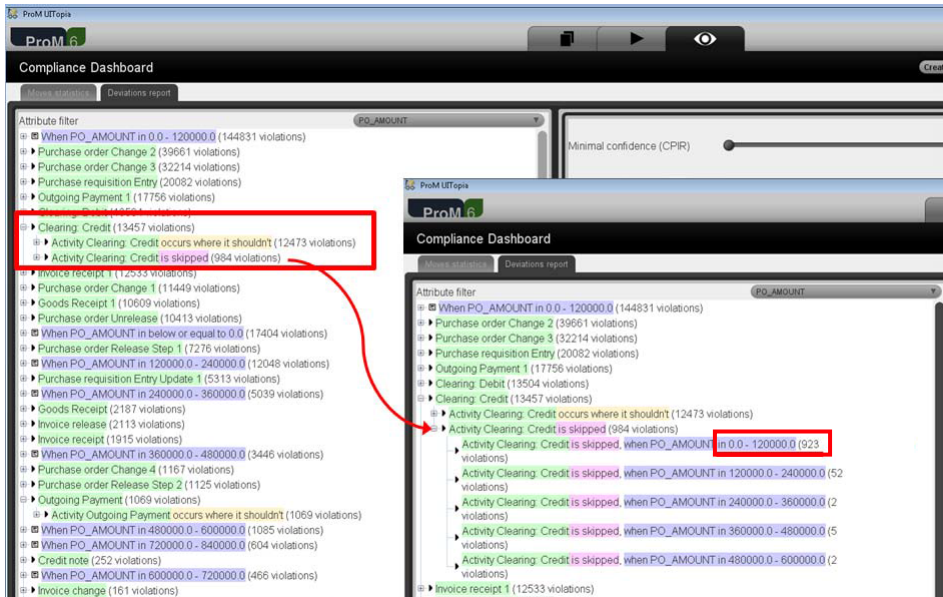


Figure 7.19: Part of the *problem list* obtained for the example log in our case study.

business analysts and auditors. Therefore, together with four of the compliance experts from PwC working in assurance and forensic sections, we did a survey to check if the output visualization of plugins was understandable and clear. This study consisted of several workshops in participation with compliance experts from PwC, and process mining researchers at TU/e.

In these workshops we first provided a business process model together with several compliance rules specified in that model that we wanted to analyze. Furthermore, we gave a short (about 15 minutes) introduction to the plugins and our approach. The participants were provided with an event log related to the process, the model, and a set of questions about the details of the diagnostics that can be obtained after applying the two plugins. These diagnostics were related to details of the compliance status of the process that could not be answered simple inspection of the log, but could be answered using the techniques presented in this chapter. Participants were unfamiliar with the event log and the technique prior to the workshop and no further details about the log or the process were provided.

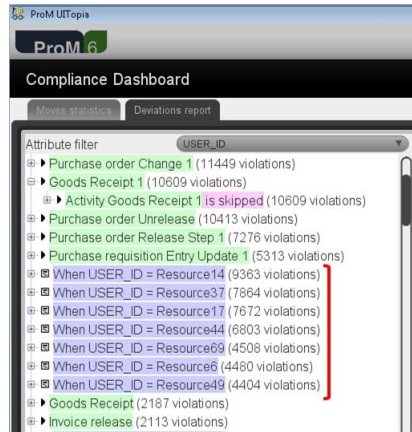


Figure 7.20: Resources frequently linked to violations.

We prepared a questionnaire with 10 questions. Each question was asking about a specific piece of diagnostics generated by the tools. In the following we present some of the questions listed in the questionnaire:

- List the violating activities.
- Which *material* is likely to have connection with the identified violations?
- What are the common attributes and their values present in violations related to both *Rule 1* and *Rule 2*?
- List the resources that executed activities with a violating *amount* attribute.
- In your opinion which of the attributes *purchase order amount* and *entry amount* has a stronger relation to violations of *Rule 3*?

We asked the participants to do the analysis and answer the above questionnaire based on their findings. The event log contained three fraud scenarios and the questions were designed to test if the diagnostics offered by the techniques is clear enough to be understood by our participants. Participants needed to run both “*Get Compliance Dashboard*” and “*Get Problem Insight*” plug-ins to answer the questions.

Answers	Questions									
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
correct	100%	100%	82%	73%	100%	54.5%	100%	100%	64%	73%
partly correct			18%							
incorrect				9%		27.3%			36.4%	27.3%
no answer				18%		18%				

Table 7.6: Result of the survey on the visualization of the results in the *Compliance Framework*.

Table 7.6 shows how participants answered each question. The full report of the survey and its results are available in [54].

All participants answered 5 out of 10 questions correctly. For the third question, 9 out of 11 participants provided correct answer. 18% of the participants did not answer questions four and six. 3 out of 11 gave an incorrect answer to questions ten and six. 18% participants gave answers to the question three which were partially correct.

In addition to the questionnaire, we asked the participants to give us suggestions for improving the presentation of the results. Some of the participants criticized the data representation. This included the way numerical ranges were expressed and the color scheme used for their representation. Another observation in the feedback was related to the interface. It was claimed that there

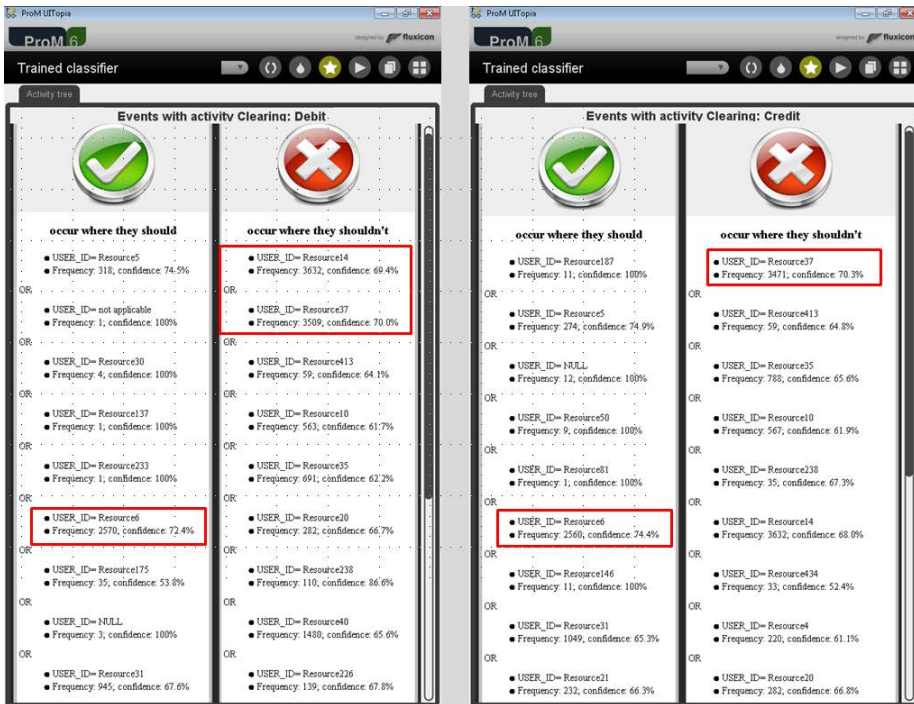


Figure 7.21: Root-cause analysis of unallowed executions of activities *clearing debit* and *clearing credit* wrt attribute *resource*.

was too much information on each screen and that it was difficult to choose the correct attributes for the 'Get problem insight' plugin. The feedback was used to improve the data representation and the interface. Such improvements include changing the way to express numerical ranges or showing IDs of the cases for each item in the problem list. Another improvement was to extend the user's capabilities in refining the list by adding a filter that allows them to keep only assignments of the selected attribute. The revised version of the plugins are available now in ProM 6.5.1.

## 7.9 Related Work

Our approach is based on a joint effort with PwC [54] and has three main components: (1) compliance checking and analysis, (2) providing statistics about deviations in different abstraction levels and from different process perspectives, and (3) guiding users in reasoning about the deviations by identifying relations between violations and their underlying context information.

Related work on compliance checking and analysis has been discussed intensively in Chapters 4, 5, and 6.

Regarding the second component of this paper, i.e., providing diagnostics for compliance checking, fewer works are available [104,122]; mostly in the area of Service Oriented Architecture (SOA). They introduce Key Compliance Indicators (KCI) and report about compliance status of business services computing these indicators.

In order to enable root-cause analysis of deviations (third component), we advocate the idea of discovering relations between violations and their underlying context. This creates the need (1) to measure the strength of the relationship between a violation and the underlying context, and (2) to find a way to discover what distinguishes violating events from non-violating ones. Both problems are solved using data-mining approaches. The approach proposed in [104] also uses decision trees to identify the root cause of problems, but it classifies process instances, while the approach in this paper classifies alignment moves. Also [104] does not employ any way of discovering association rules to measure the strength of the relationship between a violation and the underlying context; to this end we leveraged the approach in [152]. Regarding reporting on diagnostics, [104] displays list of activities for each compliance rule that can be drilled further down to the context level. At the bottom level, a user can observe various context data regarding the violation. Our approach allows the user to explore not only violations related to a specific activity, but

also violations related to specific values of context attributes. This allows a user to discover common context patterns across different violations and hence get better understanding of the observed problems.

In [59] a technique to get insights into context-related information of event logs is proposed. Although, the proposed approach allows a user to get useful context information, it does not focus on violations. Instead it aims to present the context data of an event log in general. There are more papers on the visualization of process related information. For example, [32, 58] focus on visualizing the execution of event logs but providing few diagnostics.

## 7.10 Concluding Remarks

In this chapter we provided an approach to quantify compliance levels of a business process recorded in execution logs, provide diagnostics about violations and guide users to reason about the root-cause(s) of deviations.

The technique in this chapter is based on the assumption that the enriched event log contains diagnostic information about all compliance rules of interest that should be considered together by creating a composite compliance model. We checked the compliance of an event log against the composite model using data-aware alignments. The violation data were aggregated from different process perspectives to give an overview about the process compliance. A set of statistic tables, bar charts, and a problem list were used to report on violations in different abstraction levels. The problem list provided a means for non-technical users to focus on important violations. We discussed how to rank violations based on their negative impact and based on their context relevance. We discussed several metrics for quantifying the relation between a violation and its contextual information using CPIR metric. To analyze a specific violation, we used decision learning techniques and classify violating and compliant patterns w.r.t. a particular violation.

We have implemented our approach using ProM. The software has been tested and applied on real-life data. Moreover, we studied the understandability of the diagnostics. The results are encouraging. We were able to uncover various deviations and reason over the root-cause(s) of the deviations.

The next chapter addresses the situation where compliance constraints are formalized as several atomic compliance rules into one enriched event log which then can be analyzed with the techniques of this chapter.





# Chapter 8

## Integrating Compliance Results for a Precise Analysis

As is shown in our compliance analysis approach in Fig. 8.1, the last step in compliance checking is enriching the original event log with the diagnostics obtained during compliance checking. The enriched event log then can be used to analyze the root-causes of violations and obtain deep diagnostics about non-compliance. This information can be used for compliance improvement and taking corrective measures to reduce the negative impact of non-compliance.

Chapters 4, 5, and 6 discussed various approaches for checking compliance rules from different perspectives. When modelling a set of compliance con-

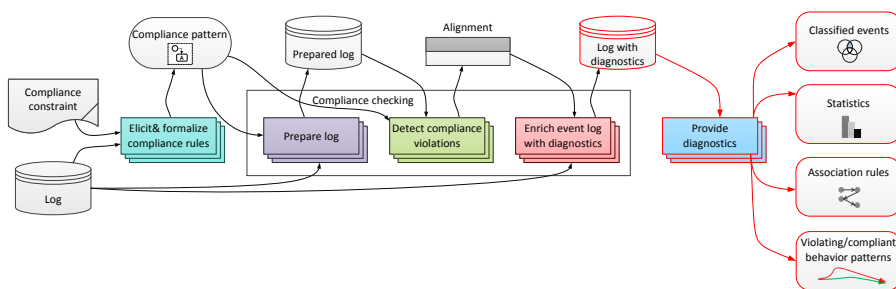


Figure 8.1: Compliance Analysis Overview.

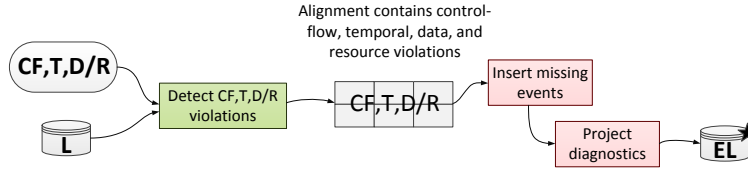


Figure 8.2: Generating an enriched event log with diagnostics based on one alignment result.

straints as a composite compliance model, the result of compliance checking includes violations related to all these constraints in a single alignment. Chapter 7 showed how to use the diagnostics obtained by such an overall alignment to analyze the violations further for root-cause analysis.

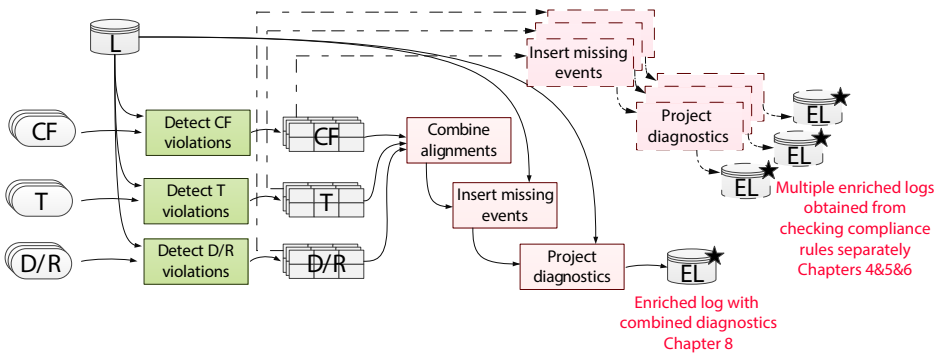


Figure 8.3: Generating an enriched event log with diagnostics obtained from several alignments.

When formalizing compliance rules as separate atomic patterns, the compliance checking results are distributed over several alignments obtained from checking each rule individually. Therefore, we need to integrate diagnostics from different alignments to get the complete picture of compliance in a process. Figures 8.2 and 8.3 illustrate the differences between the two approaches. Figure 8.2 shows how a single alignment contains all diagnostics after checking, when a set of compliance rules are modelled using a single composite compliance model. Figure 8.3, on the other hand, visualizes an extra step for com-

---

binning alignments during the log enrichment when a set of compliance rules are modelled as a set of atomic patterns. We discussed in Chapters 4, 5 and 6, how to generate an enriched log from the result of checking an atomic rule individually. In this chapter we discuss how to combine several checking results in one single enriched log. The reasoning over when a set of rules should be modelled as a composite model or as a set of atomic patterns has been discussed in chapters 3, 4, 5, and 6.

To combine the diagnostics obtained from different checking results, we collect all the obtained diagnostics in a database called *compliance database (compliance DB)*. This chapter discusses how we collect and combine compliance data in the *compliance database*. This database not only provides us with a platform for collecting and combining compliance data: it also enhances the flexibility to explore and query data to build various (even complex) filters based on multiple criteria. In addition, we have the possibility to view data from different dimensions such as compliance rules, violations, processes, process instances, events, and attributes. These various subsets of data then can be used for analyzing the root-causes of violations, detecting violating and compliant patterns, comparing different patterns, and produce various statistical reports on compliance data.

Figure 8.4 illustrates how the content of this chapter is organized in different sections. Various sources of compliance data that must be integrated, and the compliance database structure and its components are discussed in Sect. 8.1. Exploring compliance data from various perspectives and some sample queries are explained in Sect. 8.2. Section 8.3 describes our approach for combining compliance diagnostics into an enriched event log. In Sect. 8.4, we propose a framework of tools to populate the compliance database and export data from it. Section 8.6 concludes this chapter.

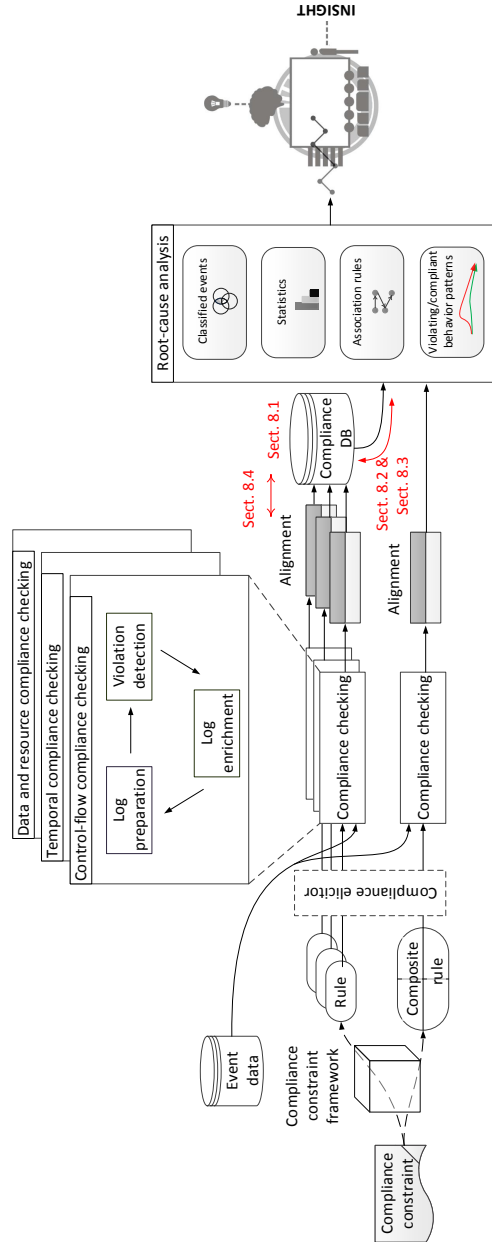


Figure 8.4: Thesis road map gives the mapping of the sections in Chap. 8 on to our compliance analysis approach.

## 8.1 Integrating Compliance Data Using Compliance Database

The compliance DB should provide a platform to integrate all compliance related data. Furthermore, the data model of compliance DB should connect different sources of compliance data so that we can explore data from different angles. Figure 8.5 visualizes different sources of compliance data. Compliance rules and event data are the inputs of our compliance analysis approach and the result of checking is stored in alignments. Recall from previous chapters that during compliance checking, the event log usually goes through a lot of preparation steps depending on the rule and the particular checking technique employed (details of log preparation steps are explained in chapters 4, 5, and 6). Hence, the prepared event log is also another source of compliance data. The compliance database collects and combines all these data sources and defines the connections between them. The information stored in this database is the input for generating an enriched log with combined diagnostics. We can query the information we need to be shown in the enriched log from the database and start the log enrichment process.

Next, we first describe compliance data sources. Later we explain the data model of compliance database and discuss its components by example.

### 8.1.1 Sources of Compliance Data

As is shown in Fig. 8.5, *compliance rules profile*, *event log*, *prepared log*<sup>1</sup>, and *results of compliance checking of different rules (enriched log)* are integrated in the compliance database. A *compliance rule profile* contains information about a rule that is checked. This information includes the compliance constraint that a rule originates from, rule name, rule description and its type. Each of these data sources gives rise to several entities in the relational model of the database structure. Before we go into details of the relational data model, we first discuss the entities of each data source and how they are related to each other. Figure 8.6 shows all entities in our data model grouped by the different data

---

<sup>1</sup>Note that we include *prepared log* into compliance DB as one of the sources because such logs may contain information that is not present in alignments or original logs. Recall from Chapters. 4, 5, and 6 that during preparation of event logs, we may shorten original logs. In this case, the positions of abstracted events are kept in the prepared log. We may even insert artificial events during log preparation (e.g. when aligning of enriched event log obtained from control-flow alignment with the generic temporal pattern). Such events can have attributes that are not present in the original log.

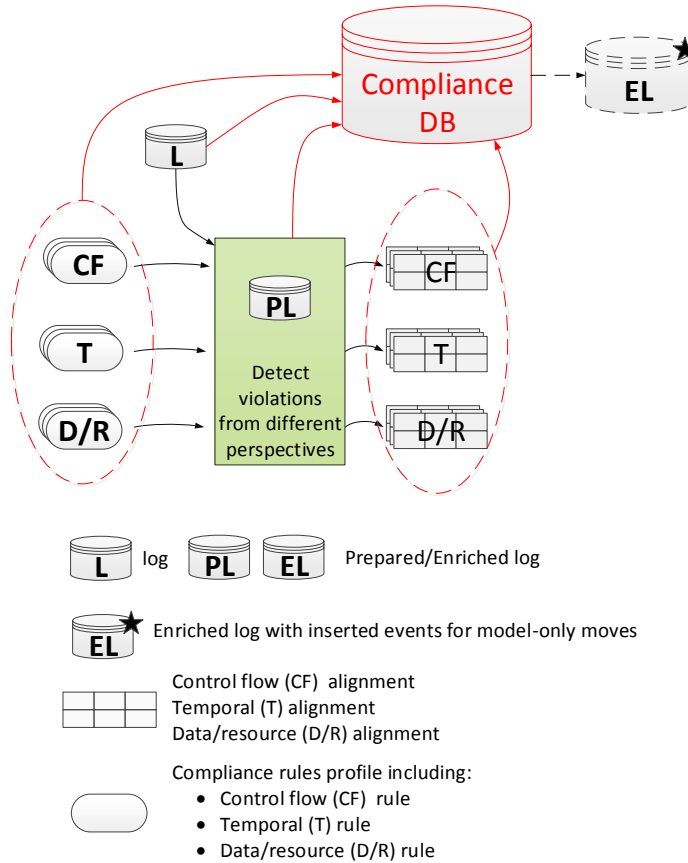


Figure 8.5: Sources of compliance data.

sources. Each cluster in this graph holds data of one of the four data sources. The nodes corresponds to data entities in the data model of the compliance database and the arcs indicate relations between theses entities.

First, we give an overview on the general structure of the data model and its elements. In Sect. 8.1.2, we will describe the entities and relations in more detail and reason how this structure allows us to integrate all data in a way that allows for meaningful queries supporting root-cause analysis.

The main data entities in the ‘event data’ cluster includes *log*, *trace*, and

*event*. Note, *attribute* node is in between ‘prepared log’, and ‘event data’ clusters and involves both. This entity contains all different attributes used in an event log or prepared log. The attribute values in log and prepared log are stored in separate entities including: *log-attribute value*, *trace-attribute value*, *event-attribute value*, *prepared log-attribute value*, *prepared trace-attribute value*, and *prepared event-attribute value*.<sup>2</sup> The *attribute* node also connects compliance rule profile cluster to other clusters. The compliance rule profile cluster includes all entities related to rules including: the *compliance rule* and the *compliance constraint* that a compliance rule originates from. Every compliance rule measures the compliance of some process property. For example a rule may confine attribute *time*, *resource*, etc. The rule may be related to different attributes in different processes. Therefore, the entity *rule-attribute* relates the compliance rule profile to the attribute in the log that is confined by the rule.

The ‘compliance checking result’ cluster, consists of the entities *alignment*, *move*, and *violation*. This cluster is connected to other clusters via *move* node. The compliance checking result cluster is connected to the compliance rule profile via the relation between *move* node and *compliance rule instance*, and between *move*, and *compliance rule*.

---

<sup>2</sup>Note, attributes usually do not overlap between the aforementioned entities. For example trace level attributes are different than log level attributes. Therefore separate their values in different entities but the attribute entity which contains only attribute IDs stays the same.



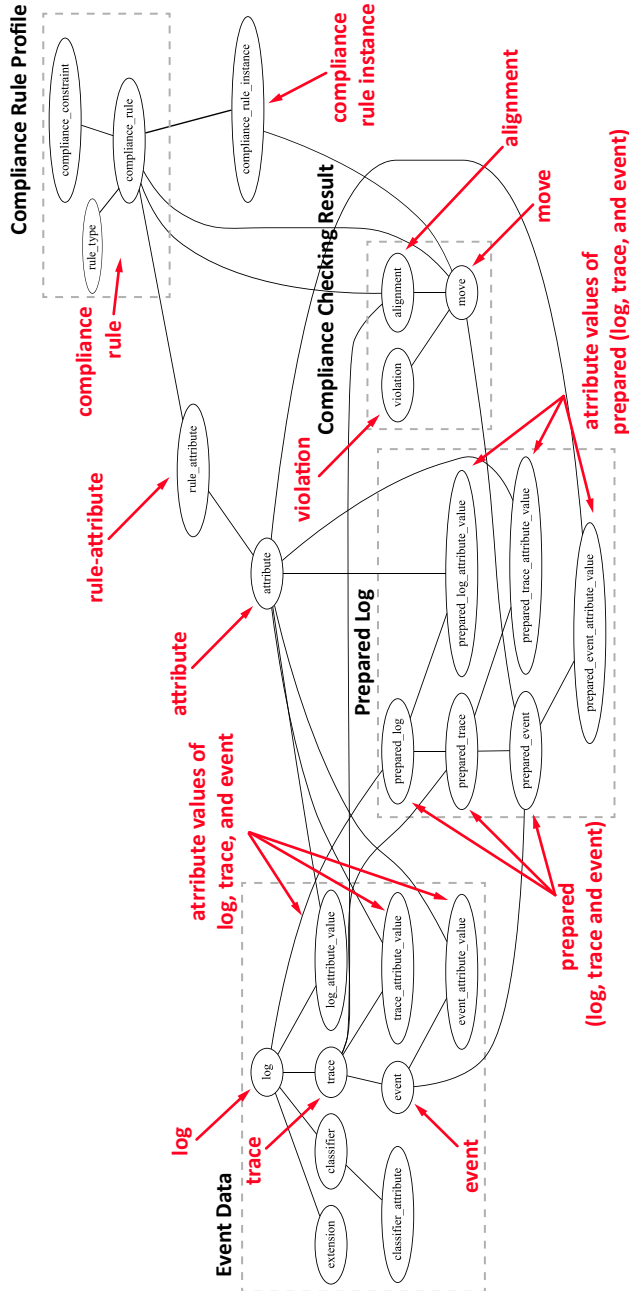


Figure 8.6: Different clusters of compliance data.

### 8.1.2 Compliance Database and Its Components

Figure 8.7 shows a simplified relational data model for the compliance DB. In this section, we explain the important entities and their relations. The model is based on the clusters introduced in Fig. 8.6. Figures 8.8, 8.10, 8.12, and 8.14 that follows throughout next chapters, zoom into different parts of Fig. 8.6.

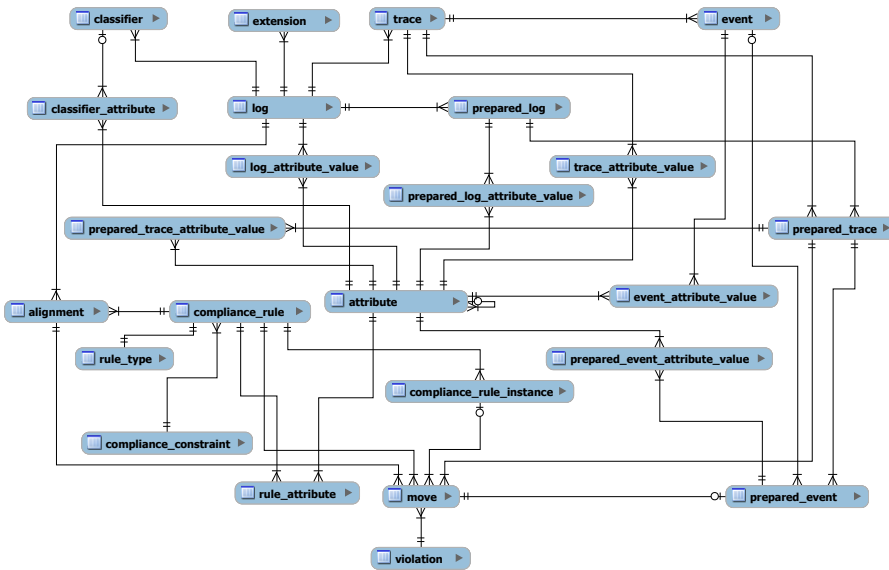


Figure 8.7: Relational data model of the compliance database.

**Compliance rule profile.** Figure 8.8 shows the data entities related to *compliance rule profile* and their relations. *Compliance\_rule*, *rule\_type* and *compliance\_constraint* (marked in red) are the three entities in the compliance rule profile cluster, other entities shown are part of other clusters that are related to the compliance rule profile. As is shown in Fig. 8.8, a compliance constraint can have one-to-many compliance rules.

Assume the accounting rule *two-way match*.<sup>3</sup> This rule originates from *Sarbanes-Oxley Act (SOX 2002)* (as its respective compliance constraint) and confines *process data* and is of type *data-aware*. Note that a process may be

<sup>3</sup>See Chap. 6 Table 6.1.

checked against many of such rules. Therefore, when integrating and combining different compliance sources, it is necessary to be able to track back to understand the rule that has been checked.

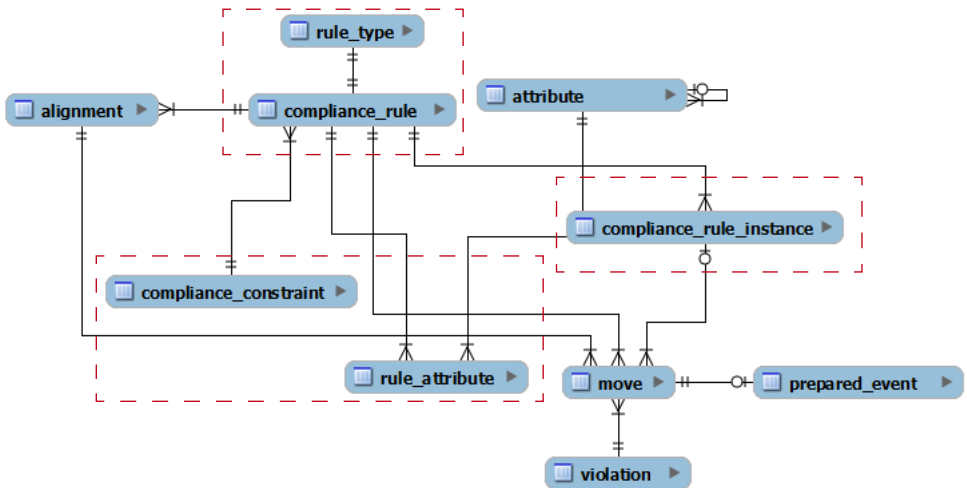


Figure 8.8: Parts of Fig. 8.7 are chosen that show data entities and their relations related to compliance rule profile.

Suppose, there are two compliance constraints that a process must comply to. Each of these constraints is divided into two compliance rules. As we discussed in Chap. 4 and Chap. 3, compliance constraints may be decomposed to several compliance rules. The tables shown in Fig. 8.9 store the information related to these constraints according to the data model of compliance DB.

As can be seen, a compliance constraint may have one or more rules (see the multiplicity *one-to-many* shown in Fig. 8.9 on the association between *compliance\_constraint* and the *compliance\_rule*). Each rule has a *type*, a *description*, and refers to the *constraint* that it originates from. *Rule types* are recorded in another table (*rule\_type*). There are *six* different rule types. As mentioned earlier, each rule has *exactly one* rule type. This is indicated by the multiplicity on association between *compliance\_rule* and *rule\_type*. Each rule confines a process property. The relation between a rule and the attribute describing that process property is captured in a separate table called *rule\_attribute*. For example, *rule\_1* is confining the control-flow of the process. Hence, this rule is paired with the

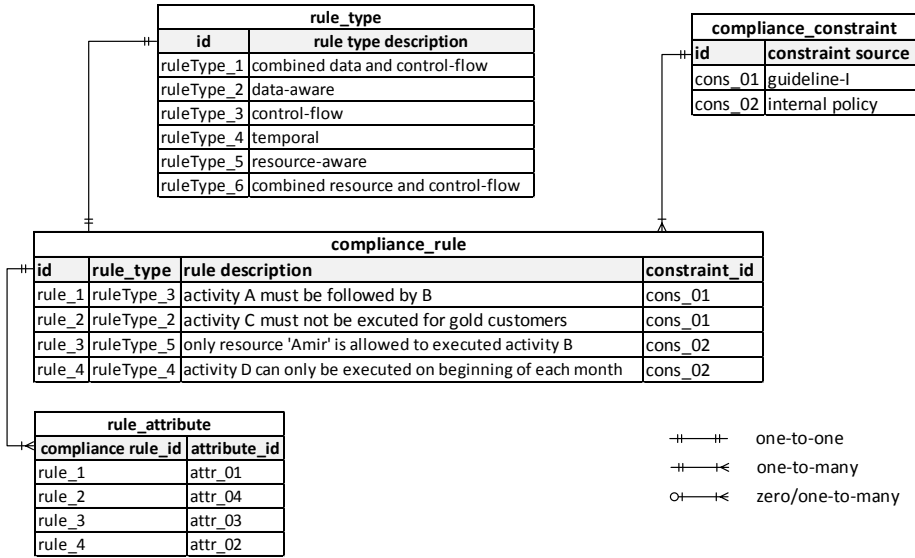


Figure 8.9: An example compliance profile populated with two constraints and their rules.

attr\_01 which records the *activity names*. Figure 8.11 indicates that in table *attribute*, attribute *attr\_01* is the *activity name*). Similarly, *rule\_2* reasons on *client category*, *rule\_3* confines *resource*, and *rule\_4* constrains *date*.

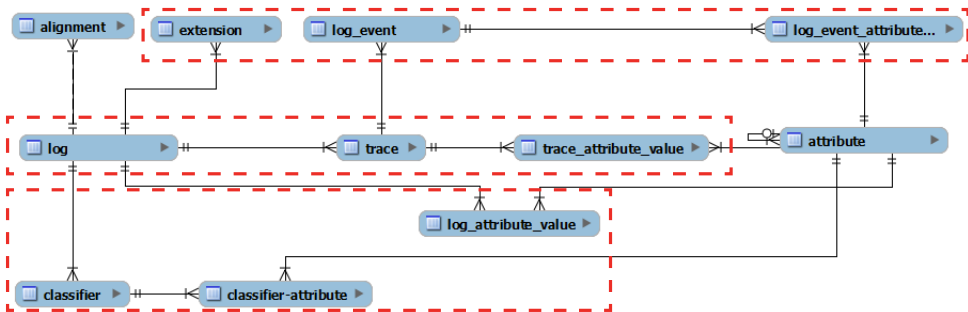


Figure 8.10: Data entities and their relations related to *event data*.

**Event data.** As mentioned before, the event log (as the input of compliance checking) and related entities form another cluster in the data model of the compliance DB. We may record several event logs from the same business process but each event appears in just one trace and log. For example, event logs referring to executions of a business process in different periods of time, or different regions. When analyzing violations, it is necessary to be able to track back to the event log (e.g. the period of time or region) a violation is referring to. Log attributes and their values are stored in *event\_attribute\_value* entity. We can track back from traces to the log they belong and from there query logs with specific attribute values. The data entities and their relations related to the event data cluster in the data structure of the compliance database are shown in Fig. 8.10.

Suppose we checked an event log against the compliance rule profile shown in Fig. 8.11. The *log* table records information about the log. Each log has one or more *traces*. The association between the two tables *log* and *trace* allows to track back each trace to the log it originates from. Similarly, the relation between the *trace* table and *event* allows to track back events to their traces. In our example, we have populated the tables with seven events distributed over two traces. *trace\_1* has four events, i.e., *trace\_1*:  $\langle event\_1, event\_2, event\_3, event\_4 \rangle$  and *trace\_2*:  $\langle event\_5, event\_6, event\_7 \rangle$ .

*Client category (attr\_4)* is a trace level attribute and its value is *gold* at *trace\_1*. This attribute has value *silver* at *trace\_2*. The events also have several attributes: *activity name*, *date (attr\_1)*, and *resource (attr\_3)*. All these attributes are stored in the *attribute* table. The value of attributes at each trace or event are stored in separate tables *trace-attribute value*, and *event-attribute value*. For instance, from table *event-attribute value*, we can see that *event\_1* indicates the execution of activity *A* on *15-Jan-11* by *Amir*. Similarly, the information related to other events are recorded in this table.

**Prepared log.** When we check different compliance rules individually, the log preparation is also done for each rule individually. Figure 8.12 shows the data entities and their relations of the *prepared log* cluster in the data model of compliance DB. Each prepared log refers to exactly one log, but a log may have zero or more prepared logs (see the multiplicity on the association connection between *log*, and *prepared log* tables). Each prepared log has one or more *prepared traces* and similarly each prepared trace has one or more *prepared events*. Prepared trace, and prepared event are related to *trace*, and *event* tables. For instance every trace has zero or more prepared traces and every prepared trace refers to exactly one trace. Similarly an event has zero or more prepared events.

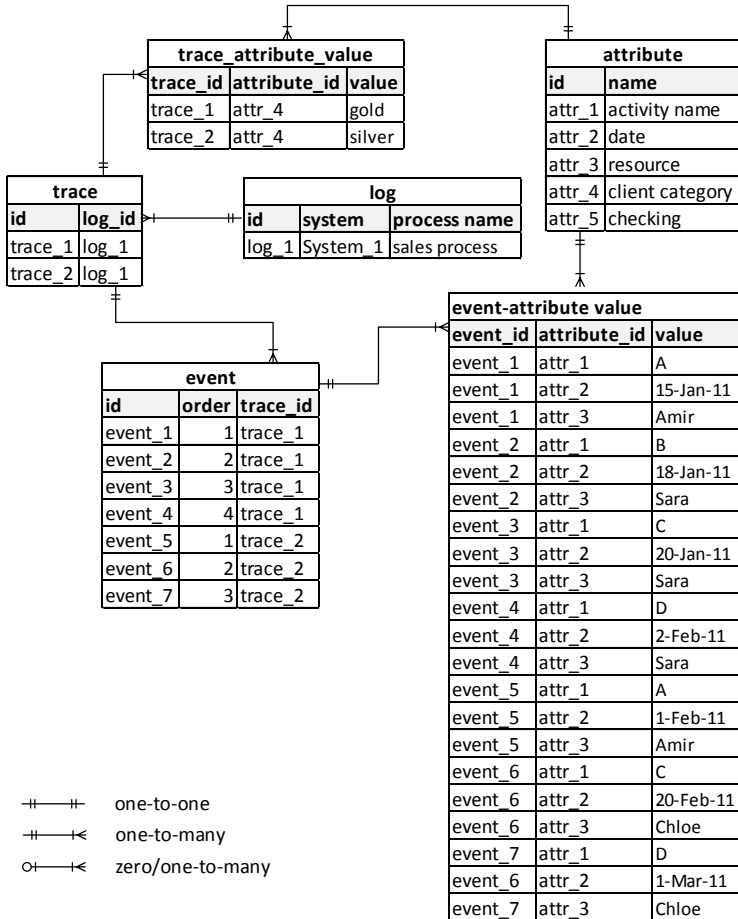


Figure 8.11: Event data tables populated with an example *log*, two *traces*, and their *events*.

However, a prepared event does not necessarily refer to an event. An example of such a prepared event can be the artificial events created for model-only moves during log preparations of temporal compliance checking to mark rule instances (for details on log preparation in temporal compliance checking, see Chap. 5.1).

Figure 8.13 shows parts of the information related to prepared log of our

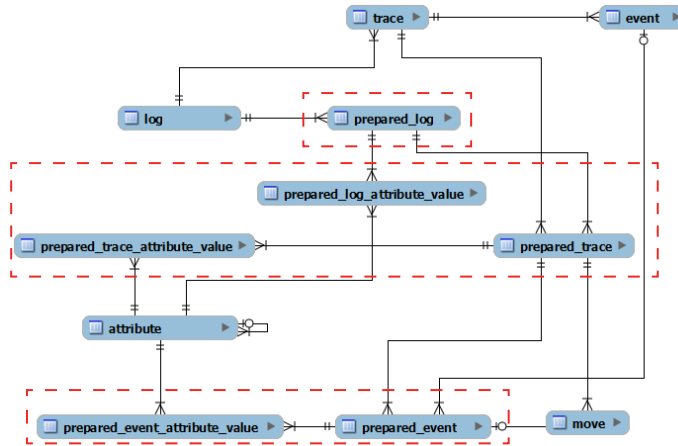


Figure 8.12: Data entities and their relations related to *prepared log*.

example. As it is indicated in the *prepared log* table, four instances of prepared log are stored for the four rules that 'log\_1' is checked against. The foreign key *log\_id* in *prepared log* table shows that all the instances of prepared log originate from 'log\_1'. In total eight instances of prepared trace are stored in table *prepared trace*. The foreign key 'trace\_id' in table *prepared trace* indicates to which trace each instance of prepared trace is referring to. For example 'prepared trace\_11' and 'prepared trace\_12' are the result of the log preparation for 'rule\_1'. 'Prepared trace\_21', and 'prepared trace\_22' are the result of the log preparation w.r.t. 'rule\_2'. Note, in Fig 8.7 each *move* is related to a *compliance rule* and at the same time to a *prepared event*. Hence, we can trace back to the *rule* that a *prepared log* is built for.

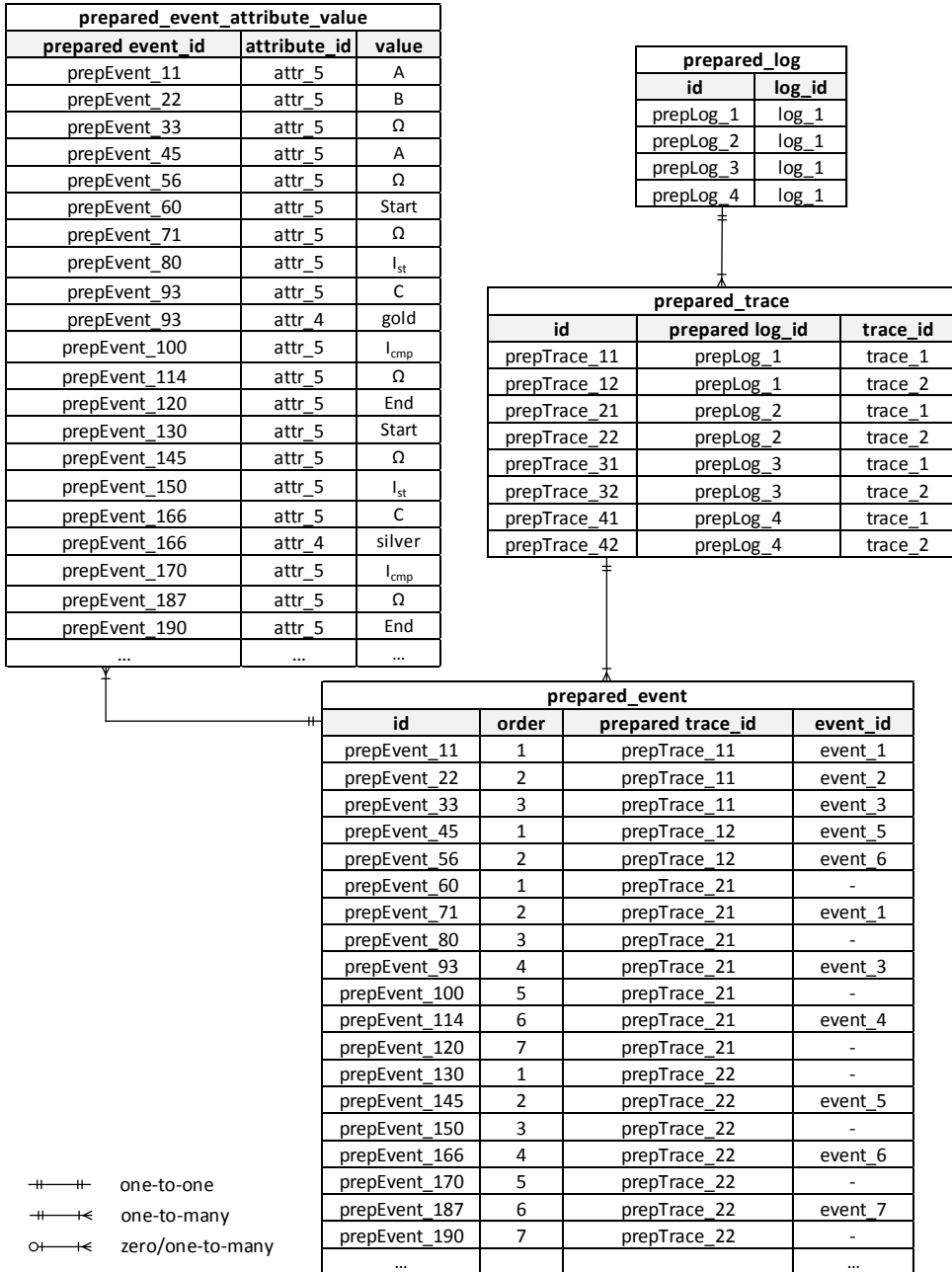


Figure 8.13: Entities and their relations of an example prepared log cluster populated with four prepared logs, eight prepared traces, and their prepared events.



Similarly the *prepared event* table contains the instances of prepared events. For example, 'prepTrace\_11' has three prepared events: 'prepEvent\_11', 'prepEvent\_22', and 'prepEvent\_33'. Note that this trace has one event less compared to original trace 'trace\_1', because during the log preparation, the trace is shortened. However sometimes a prepared trace has more events compared to the original trace it is derived from. For example 'prepTrace\_21' has seven prepared events: *prepTrace\_21*:  $\langle \text{prepEvent}_{60}, \text{prepEvent}_{71}, \text{prepEvent}_{80}, \text{prepEvent}_{93}, \text{prepEvent}_{100}, \text{prepEvent}_{114}, \text{prepEvent}_{120} \rangle$ . From these prepared events, *prepEvent\_60*, *prepEvent\_80*, *prepEvent\_100*, *prepEvent\_120* do not refer to an original event (the 'event\_id' in the table prepared event is empty for these prepared events). During the log preparation for checking 'rule\_2', the original log is enriched with some artificial events to mark the rule instances (see Sect. 6.3 and Sect. 6.4 for how to mark rule instances in a log).

Similar to original traces and events, prepared traces and prepared events have attributes. The value of these attributes at each prepared trace and each prepared event is captured in tables *prepared trace-attribute value* and *prepared event-attribute value*. For example all the prepared events have a value for attribute *attr\_5 (checking)* and values of this attribute at each prepared event are shown in table prepared event-attribute value.

**Compliance checking result.** As explained before, the compliance checking result cluster in the data model has entities that store diagnostic information in alignments. The execution of an event in an event log may be checked against different rules. Therefore, to get a complete picture about the compliance of this activity we need to combine the individual diagnostics obtained in each checking procedure. For example, we would like to be able to query against

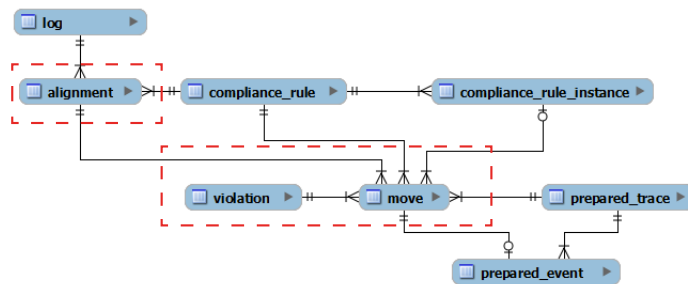


Figure 8.14: Data entities and their relations related to *compliance checking result* cluster.

how many rules the execution of an activity has been checked, and how many violations of which type and from which rules are observed for that specific activity. Therefore, it is necessary to be able to track back to the exact checking result that each diagnostic is produced from. Our data model realizes this as follows.

Figure 8.14 shows the entities and their relations of this cluster. Each *alignment* indicates the result of checking a log against a rule. Hence, the 'log\_id', and 'rule\_id' (as foreign keys) record this information for each *alignment*. Each *alignment* has one or more *moves* and each move refers to exactly one alignment. A move has an attribute *order* that reflects the order of moves in an alignment. It is indicated in each move whether it is inside a rule instance or not. The 'rule\_instance\_id' refers to various activations of a rule. A move therefore refers to zero or one *rule instance*, however, a rule instance has one or more moves. It is also indicated in each move whether it is violating or not. This information is recorded for each move in 'violation\_id' column. Each move refers to exactly one *violation* type, including "no violation (compliant)" and each violation may be connected to zero or more moves. Furthermore, each move may refer to a *prepared event* or not. Synchronous moves and log-only moves refer to exactly one prepared event and model moves have no prepared event. Hence, each prepared event is connected to exactly one move.

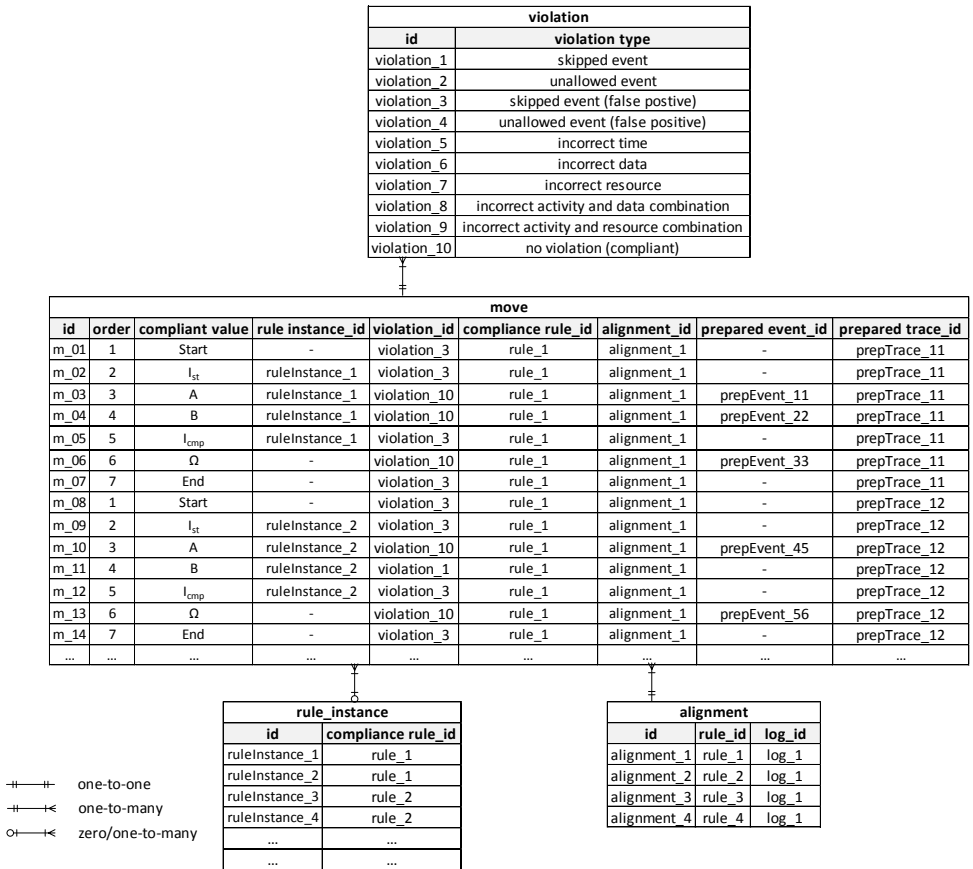


Figure 8.15: Example entities and their relations related to a compliance checking result. Figure above shows four *alignments*, ten types of *violations*, parts of *move* table, and parts of *rule\_instance* table.

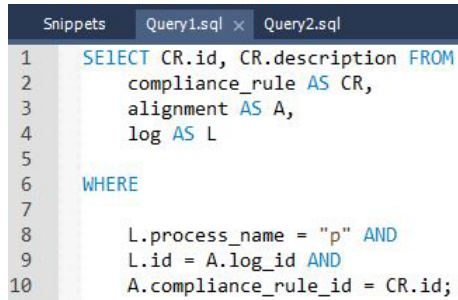
Figure 8.15 shows the entities of compliance checking result cluster populated with our example. Suppose we checked the event log against the example compliance rule profile, the alignment table records four different alignments, each as a result of one checking procedure. As explained before, each alignment is referring to a rule, and a log. For example ‘alignment\_1’ is the result of checking *log\_1* against *rule\_1*. The *moves* related to this alignment are shown partially in table move. The sequence of moves  $\langle m\_01, m\_02, m\_03, m\_04, m\_05, m\_06, m\_07 \rangle$  refers to checking ‘prepTrace\_11’ against ‘rule\_1’ and the moves are ordered accordingly. The first move ‘m\_01’ denotes the start of the trace and ‘m\_07’ denotes the end of the trace. Similarly ‘m\_02’ and ‘m\_05’ denote the start and completion of ‘rule instance\_1’. ‘Rule\_1’ checks whether the execution of activity *A* is followed by *B* and therefore moves ‘m\_03’, and ‘m\_04’ have ‘no violation’. Note, the moves *m\_01*, *m\_02*, *m\_05*, and *m\_07* have violations of type *skipped event* because these moves are not referring to any prepared event. However, these violations are no violations but only technical steps introduced by the compliance patterns, since the moves are not referring to real business activities. The *compliant value* column records the compliant value of the attribute that was expected to occur according to the rule. Therefore, in our example for the moves *m\_01*, *m\_02*, *m\_05*, and *m\_07*, referring to violation of type *skipped event*, the compliant (compensation) value refers to values *Start*, *I<sub>st</sub>*, *I<sub>cmp</sub>*, and *End*.

Table rule instance shows that ‘rule\_1’ has been activated twice, therefore we have ‘rule instance\_1’, and ‘rule instance\_2’ both pointing to ‘rule\_1’. Similarly other rule instances are listed.

In this section, we discussed in detail the structure of the compliance data model. This data model allows us to integrate all compliance data. In addition, it gives us the flexibility to explore and query data for various reporting purposes and follow up root-cause analysis.

## 8.2 Querying Information from the Compliance Database

The design of the data model of compliance DB allows us to query and view compliance data from different perspectives including compliance rule profiles, event logs, and compliance checking results. The relations between different data clusters give us the flexibility to query and analyze data regarding complex criteria. Furthermore, this analysis can be done on different levels such as: compliant versus violating, move or alignment level, event and process instance level, constraint, compliance rule and rule instance level.



```

Snippets Query1.sql x Query2.sql
1 SELECT CR.id, CR.description FROM
2 compliance_rule AS CR,
3 alignment AS A,
4 log AS L
5
6 WHERE
7
8 L.process_name = "p" AND
9 L.id = A.log_id AND
10 A.compliance_rule_id = CR.id;

```

Figure 8.16: An example query in SQL requesting all the compliance rule that a process  $p$  is checked against.

### 8.2.1 Exploring Compliance Data

In the following we list some example queries in textual form (and two example in SQL) to elaborate on the different dimensions of the analysis that is possible to do using compliance database:

- *What are the compliance rules that a process  $p$  is checked against?*
  - Tables involved: *compliance\_rule* (number and description of rules), *alignment* (to choose a log and link the log to the rule that is checked), and *log* (for the choice of an analyzed process). Figure 8.16 visualizes this query in SQL.
- *How is the compliance of process  $p$  in the time period  $[t_1, t_2]$  with respect to all the compliance rules that it has been checked against?*
  - Tables involved: *compliance\_rule* (for the choice of rules and their description), *alignment* (to choose a log and link the log to the rules), *move* (violating versus compliant moves), *prepared\_event*, *event* and *event\_attribute\_value* (to select events that occurred during a specific period of time), and *log* (choice of a specific process). Figure 8.17 visualizes this query in SQL.
- *How is the compliance of process  $p$  w.r.t. to rule  $r$  in different months of year  $y$ ?*
  - Tables involved: *compliance\_rule* (for the choice of a rule), *alignment* (for the choice of checking results), *move* (violating versus compli-

ant moves), *event* and *event\_attribute\_value* (for the choice of a time period), and *log* (for the choice of a specific process).

- How do violating and non-violating events w.r.t. rule  $r$  differ in values of attribute  $x$ ?
  - Tables involved: *compliance\_rule* (for the choice of a rule), *align-*

```

1  SELECT M.id, V.id, V.violation_type, CR.id, CR.description FROM
2
3  log AS L,
4  alignment AS A,
5  compliance_rule AS CR,
6  move AS M,
7  violation AS V,
8  (
9  SELECT M.preparedTrace_id AS pt_id, MIN(M.order) AS min, MAX(M.order) AS max
10 FROM move as M
11 WHERE
12 M.preparedTrace_id IN
13 (SELECT trace.id
14  WHERE
15   event_attribute_value.value <= t_2 AND
16   event_attribute_value.value >= t_1 AND
17   event_attribute_value.event_id = event.id AND
18   event_attribute_value.attribute_name = "time"
19  GROUP BY trace_id
20  ORDER BY event_id
21 )
22 GROUP BY M.preparedTrace_id
23 ORDER BY M.order;
24 ) AS SQ
25
26 WHERE
27 L.process_name = "p" AND
28 A.log_id = L.id AND
29 CR.id = A.compliance_rule_id AND
30 M.violation_id = V.id AND
31 M.preparedTrace_id = SQ.pt_id AND
32 M.order >= SQ.min AND
33 M.order <= SQ.max;
34

```

Figure 8.17: An example query in SQL requesting all the moves within time period  $[t_1, t_2]$  w.r.t. all relevant compliance rules that a process  $p$  is checked against.

ment the (choice of a checking result), *move* (violating versus compliant moves), *prepared\_event*, *event*, and *event\_attribute\_value* (for the choice of events having attribute  $x$ ).

- *How many violations and non-violations are observed w.r.t. rule  $r$ ?*
  - Tables involved: *compliance\_rule* (for the choice of a rule), *alignment* (for the choice of a checking result), and *move* (violating versus compliant moves).
- *Query all the events having a violation of type “incorrect activity and data combination” in process  $p$ ?*
  - Tables involved: *log* (for the choice of a process and a log), *alignment* (for the choice of a checking result), *move* (violating versus compliant moves), *violation* (for the choice of violation type), *prepared\_event* and *event* (for the choice of events).
- *Query all the events related to log  $l$  while specifying their violations type.*
  - Tables involved: *log* (for the choice of log), *alignment* (for the choice of a checking result), *move* (violating versus compliant moves), *violation* (for the choice of a violation type), *prepared\_event*, and *event* (for the choice of events).
- *Query all the events related to log  $l$  while specifying their violations type and the compliance rule they violate.*
  - Tables involved: *log* (for the choice of a log), *compliance\_rule* (for the choice of a rule), *alignment* (for the choice of a checking result), *move* (violating versus compliant moves), *violation* (for the choice of a violation type), *prepared\_event* and *event* (for the choice of events).
- *Query all violating and non-violating moves related to compliance rule  $r$ .*
  - Tables involved: *compliance\_rule* (choice of rule), *alignment* (choice of checking results), *move* (violating versus compliant moves).
- *Query all violations of rule  $r_1$  that are violating rule  $r_2$  as well.*
  - Tables involved: *compliance\_rule* (choice of rules), *alignment* (choice of checking results), *move* (violating versus compliant moves).

- Query all the violations *w.r.t.* rule  $r$  that occurred in month  $m$  of year  $y$ .
  - Tables involved: *compliance\_rule* (for the choice of a rule), *alignment* (for the choice of a checking result), *move* (violating versus compliant moves), *prepared\_event*, and *event*, and *event\_attribute\_value* (for the choice of events).
- Query all violating and compliant process instances of process  $p$  that are violating rule  $r$  while marking the violating vs. non-violating.
  - Tables involved: *log* (choice of process and log), *trace* (choice of traces), *compliance\_rule* (choice of rule), *alignment* (choice of checking result), *move* (violating versus compliant moves), *prepared\_events*, and *events* (choice of events).
- Query all process instances of process  $p$  that share the trace attribute  $x = val$  while specifying the rules they violate.
  - Tables involved: *log* (for the choice of a process), *trace*, and *trace\_attribute\_value* (for the choice of traces), *alignment* (for the choice of checking results), *move* (violating versus compliant moves).
- etc.

As implied by the above mentioned queries, we can report on compliance data from different views and different abstraction levels. The result of these queries is usually a set of events or moves. We can compute various statistics over these subsets of data. The events or moves chosen can refer to a group of dependent events (i.e., all the events related to a process can be chosen) or they may be independent (events can be distributed over various processes and logs). We can conduct various analysis depending on the data subsets that is queried.

### 8.2.2 Analyzing Sets of Independent Events/Moves

The event/moves in the *independent* data subsets are typically selected because they have common characteristics e.g. they all have a common violation type, they all violate a specific rule or group of rules, they all share a common attribute value, etc. That is regardless of the traces and logs they originate from.

For example a query stating: “all events and their attribute values that are violating both rule  $x$  and  $y$ ”. In this case only events are chosen that are violating the two rule. Such group of events/moves can be used for various statistical



analyses e.g. calculating distribution of violations in different periods, calculating percentages of violations among different groups of interest, etc In addition these groups of moves or events can be input for several data analysis techniques such as correlation analysis, or classification (such analyses have been discussed extensively in Sect. 7.5).

### 8.2.3 Analyzing Sets of Dependent Events/Moves

The events/moves in the *dependent* subsets of data are typically used to build sub-logs. Therefore, we do not only choose events but a complete trace that contains an event with characteristics of interest. Consequently, it is possible to create a sub-log from dependent subsets. Such sub-logs can be used for detecting behavioral patterns using various process mining techniques e.g. mining the behavior of traces violating a rule versus the compliant ones. The detected patterns can be compared for differences. In addition, we can enrich these sub-logs with diagnostics obtained and analyze the enriched logs further. The log enrichment is very similar to the log enrichment step we explained in Chapters 4, 5, and 6. However, at this phase the moves used for building an enriched event log are not derived from a single alignment but from multiple alignments. The challenge then is to combine all the diagnostics obtained from different alignments in one enriched event log. In the next section, we will explain our approach for extracting an enriched event log by integrating moves from several alignments.

## 8.3 Generating Enriched Event Logs

In this section, we first briefly review the extraction of an enriched event log built as a result checking one atomic compliance rule (discussed in chapters 4, 5, and 6). Next, we explain how we build and enriched event log by integrating results of checking several atomic rules (the case visualized in Fig. 8.3).

Figure 8.18 illustrates an example enriched event log that was discussed in Sect. 5.2. The original log has been checked against a control-flow rule and a temporal rule. The log shown in Fig. 8.18 is enriched with diagnostics about both of the rules (the diagnostics related to control-flow rule are hachured in pink and the diagnostics related to temporal rule checking are shaded in blue). As can be seen, new attributes are added to the enriched event log that store the diagnostics including the *compliance rule* checked, the *rule instance* that an event refers to, whether the event is violating or not, in case of violation the type of violation and the compliant value are recorded. In addition, we also

add artificial events related to model-only moves (see event  $e_7^S$ ). This event is added in between the events  $e_6^L$ , and  $e_7^L$  and indicates that occurrence of  $e_7^S$  was expected at that position but it was skipped. Note, all this information was obtained from one alignment. Therefore, we exactly know where the model-only move is located and we can say exactly where the artificial event related to this 'missing event' must be added.

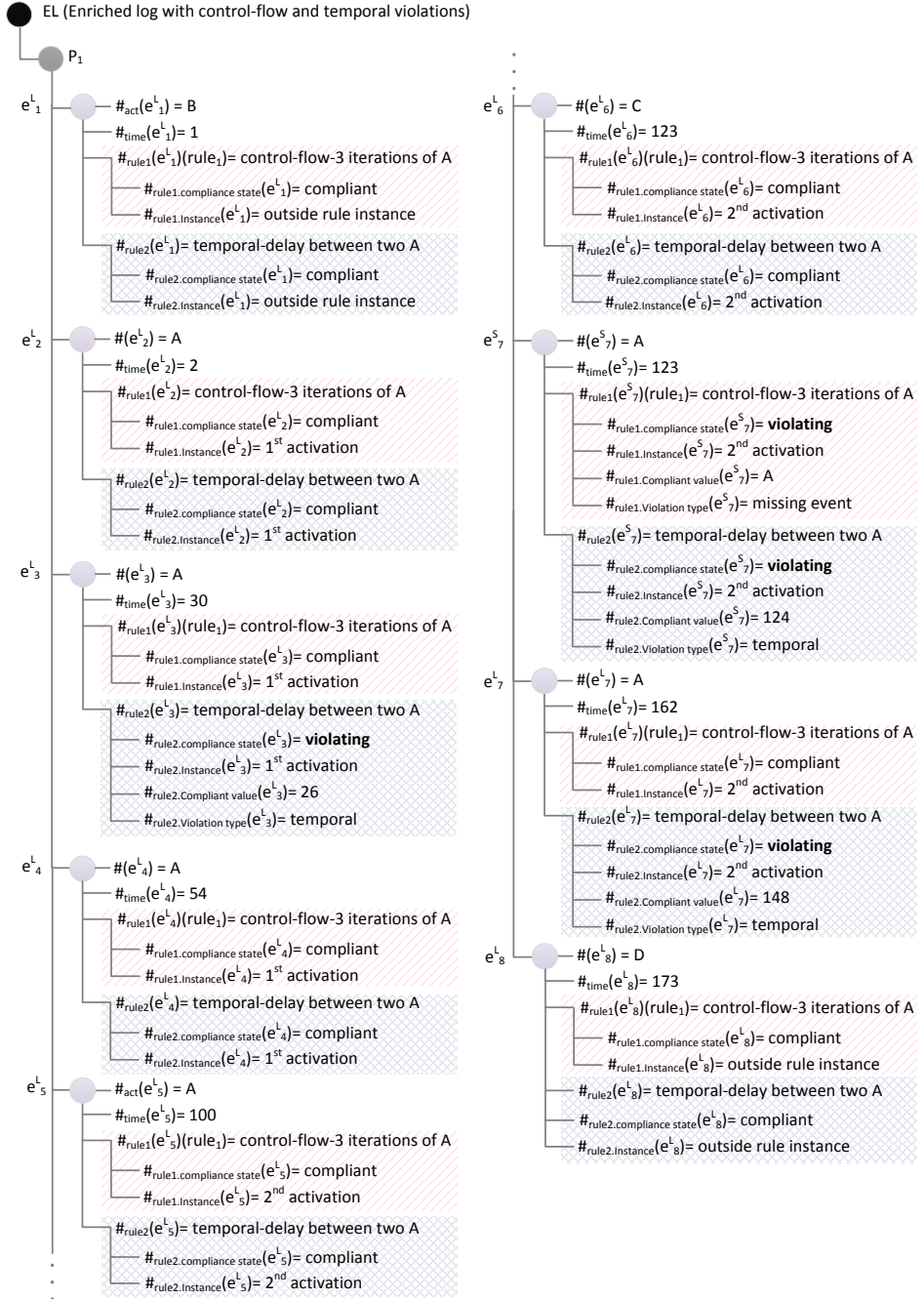


Figure 8.18: The enriched event log with control-flow and temporal diagnostics.

When we check a log against several individual compliance rules (shown in Fig. 8.3), the diagnostics are distributed over several alignments. To build an enriched event log in this case, we need to perform the following steps:

1. Query the rules that a log has been checked against.
2. Query all the alignments that involve the specified log and the chosen rules.
3. Query moves related selected to the alignments.
4. Retrieve all the original events related to the selected moves.
5. Introduce artificial events related to model-only moves (i.e., moves that do not have a corresponding original event).
6. Assign all the moves referring to an event to that event.
7. Enriching events by translating each move to a new attribute of the retrieved/artificial event containing the diagnostics recorded in that move. For example if a move is referring to a control-flow rule  $r$ , we create a new attribute named *rule r* where we store the rule type, the rule instance, and whether it was violating or not and in case of violation, the compliant value.
8. Group enriched events in each trace.
9. Sort enriched events according to the order of the retrieved event in its trace.
10. Insert enriched events that are artificial in each trace according to the location of model-only moves.

Sorting enriched events that are artificial (i.e., artificial events related to model-only moves) however is not straightforward. Suppose three different model moves  $m_1, m_2, m_3$  are detected in three different alignments. All these moves are detected to be in between event  $e_1$ , and  $e_2$  of the original event log. The detected moves need to be translated into three artificial events  $e_{m_1}, e_{m_2}$ , and  $e_{m_3}$ . In this case in the enriched event log, the events  $e_{m_1}, e_{m_2}$ , and  $e_{m_3}$  are partially ordered between events  $e_1$ , and  $e_2$ . By partially ordering these events we make explicit that no ordering exist between these events.

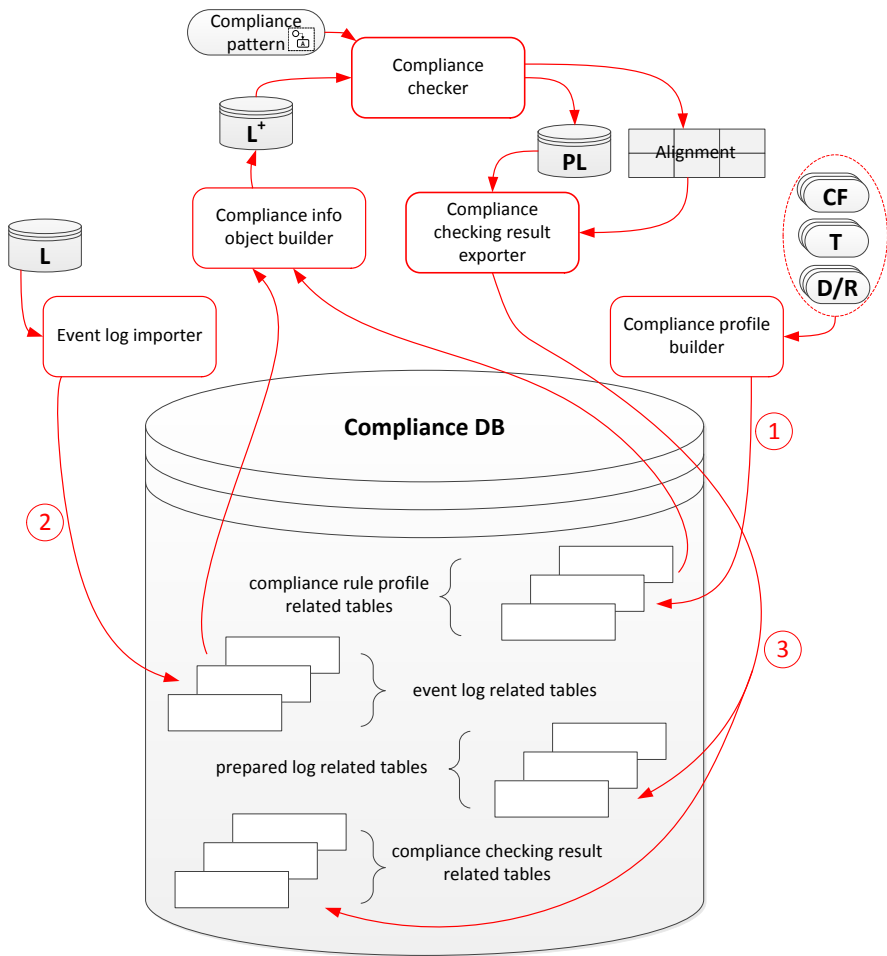


Figure 8.19: Proposed framework for populating compliance database.

## 8.4 Populating the Compliance Database with Relevant Data

In this section we discuss the machinery that is required for populating the compliance DB with different sources of compliance data. Figure 8.19 sketches a framework of tools required for populating the compliance DB.

The first step in populating the compliance DB is (1) the insertion of compliance rule profile. For this, the component *compliance rule profile creator* is required to fill in relevant tables in compliance DB with information about the rules to be checked, their type and description and the constraints they originate from.

Next, (2) the *log importer* component load an event log in XES format into the log related tables within the compliance DB. The component *compliance info object builder* takes an event log and the rule information to be checked from the compliance DB. In this component the connection between the chosen compliance rule to be checked and the event log is created. The output of this component is the event log that contains a “special” attribute that should be checked for the chosen compliance rule. The event log with special attribute together with the formalized compliance rule are inputs of the *compliance checker* component. The compliance checker then writes the result of checking in an alignment and produces the *prepared log*. The *compliance checking result exporter* component then (3) loads the *alignment* and the *prepared log* back to the compliance DB. This process can be repeated for each checking procedure. Note that *compliance pattern* is separate from the compliance DB, because the information about the pattern is irrelevant for the analyses we would like to do afterwards with the data queried from compliance DB. Hence, we do not include information related to the pattern in the database.

## 8.5 Proof of Concept

As a proof of concept, we have realized all the components of the framework shown in Fig. 8.19 by implementing over 30 separate scripts in SQL, Java and R. In this proof of concept, the process of invoking the different scripts is not automated, though invoking these scripts in the right order demonstrates the feasibility of our approach. Table 8.1 gives a summary about number of scripts per component.

In addition, we have populated the compliance DB with a small artificial

Component	# of scripts
Event log importer	10
Compliance rule profile builder	3
Compliance info object builder	4
Compliance checking result exporter	14
Building enriched log	4

Table 8.1: Number of scripts developed per component.

dataset. We recorded *two event logs* and *five compliance rules* of different types in the compliance DB. These rules are derived from *two compliance constraints*. Each of the event logs have *two traces*, so in total *four traces*. These traces have together *35 events*. We have recorded in total *293 attribute values* and *49 distinct attribute values* including all the event attributes values, and trace attribute values for this log. One of the logs was checked against the five compliance rules and resulted in *five alignments* and *19 compliance rule instances*. The alignments have in total *202 number of moves*. Table 8.2 summarizes the dataset used as proof of concept.

# of logs	2
# of traces	4
# of events	35
# of attributes	12
# of attribute values	293
# of distinct attribute values	49
# of alignments	5
# of rule instances	19
# of moves	202

Table 8.2: Summary of the dataset used as proof of concept.

*Twelve* queries were generated to extract subsets of events and moves from the compliance DB. In addition we created an enriched event log with all the diagnostics obtained from checking the five rules.

The dataset was quite small, but allowed us to check whether it is possible to seamlessly integrate data and extract diagnostics. We could integrate data generated from different components of the framework and derive detailed di-

agnostics and explore data from different perspectives. The complete implementation of the framework and an evaluation using bigger datasets and real life logs is a topic of future research.

## 8.6 Concluding Remarks

In this chapter we discussed the necessity for integrating various checking results to obtain a complete picture about the compliance status of a process and enable the root-cause analysis of violations. We detected three clusters of compliance data including *compliance rule profile*, *event log*, and *compliance checking result*. The compliance DB collects and integrates various sources of compliance data. The compliance DB is designed such that we can analyze violations from different dimensions and abstraction levels. In addition, we can track back each violation to the rule and the event log that is checked.

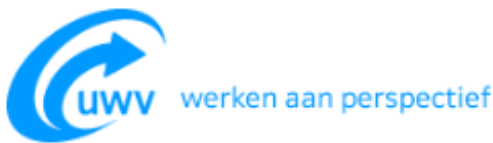
We can query the compliance data based on even complex criteria. The result of these queries are subset of moves or events. These subsets of data can be used to build different reports and statistics, for root-cause analysis, and to build sub-logs.





# Chapter 9

## Case Study within UWV



In this chapter we present the results of a case study where the concepts and techniques developed in this thesis have been applied.

This case study demonstrates the feasibility of our compliance analysis approach and applicability of its techniques and findings for improving compliance in a real business case. Furthermore, it has been chosen to illustrate some of the challenges in analyzing compliance for large scale event logs.

The case study involves the processing applications for unemployment benefits received in Dutch national Employee Insurance Agency (UWV) <sup>1</sup> that is part of the Ministry of Social Affairs and Employment to implement employee insurances and provide labor market and data services. We conducted this analysis in three business units located in different regions in the Netherlands and compared the results. Later, we informed one of the units about the results of our analysis and did the analysis in all three units again to check how diagnostics results we offered improved the process compliance of the office that was informed about the results.

---

<sup>1</sup>In Dutch “Uitvoeringsinstituut Werknemersverzekeringen (UWV)”.

As described above, the case study was conducted using *A/B testing*. Together with UWV, we chose a set of compliance rules and checked them in all the three business units. One of the business units (*A group*) was informed about the results of our analysis and we did not distribute the results to the other two units (*B group*). The case study continued with a follow-up study. We monitored and compared the changes in compliance level of different business units. The results of the analysis show that using our approach, we can provide detailed diagnostics and enable root-cause analysis to improve compliance. The compliance level w.r.t. some of the compliance rules showed up to 23% improvement in *A group*, the same effect was not observed in other business units. We received positive feedback from domain experts about the applicability of the approach and UWV is planning to incorporate our approach of compliance analysis in its daily operations.

The process that we analyzed in this case study is explained in Sect. 9.1. The design and scope of the case study will be discussed in Sect. 9.2. The pilot analysis including definition of the compliance rules, the techniques we used for our analysis, data collection and data preparation steps are elaborated in Sect. 9.3. We show the results of the first study and the feedback we received about the results in Sect. 9.4. In Sect. 9.5, we discuss the details of our root-cause analysis. The results and the improvements we obtained in the follow-up study are discussed in Sect. 9.6. We conclude our report with lessons learned in Sect. 9.8. In Sect. 9.7, we review some relevant case studies reported in the literature.

## 9.1 Process Description

In this case study, we aim at analyzing the compliance of the “*handling unemployment benefits*” process in UWV. If an employee loses his/her job and working is not possible or not immediately possible for him/her, UWV arranges a temporary income for the person. People place applications in UWV for unemployment benefits and annually on average 1.4 million Dutch citizens and their families depend on UWV for their income. Unemployment benefits is a rule-driven process and it is important for UWV to monitor if its operations adhere to relevant compliance rules or not. In addition UWV set several internal policies to keep its clients satisfied with the service they receive from UWV especially because of the impact this process has on people’s lives. Each application for unemployment benefits received by UWV goes through several phases as shown in Fig. 9.1.

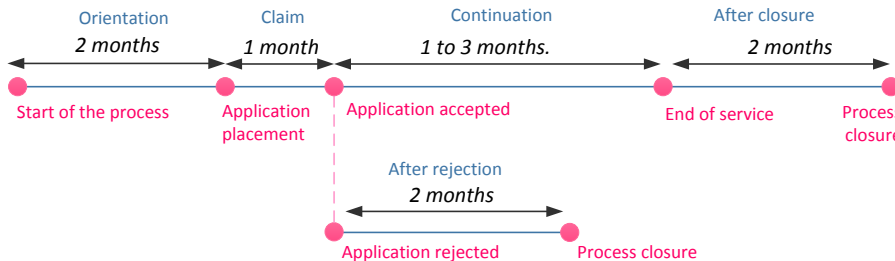


Figure 9.1: Life cycle of an unemployment application in UWV.

Usually before applicants place an unemployment benefits application, UWV supports them by providing precise information regarding their eligibility for unemployment benefits and the procedure they need to follow. Therefore, a client case starts in UWV roughly *two months* before the placement of an application where UWV sends different forms and information to its clients (*orientation phase*). After submission of an application, the *claim phase* starts where UWV officially announces its decision to clients whether they are eligible to receive unemployment benefit or not and if they are eligible what the amount of the benefits will be. This phase may take at most one month. If the application is accepted, the *continuation phase* will start, during which the applicant will receive unemployment benefits. In this phase, UWV supports its clients to find a new job or go back to work. This phase usually takes about *one to three months*. When unemployment benefits end, still the client case will be open in UWV for another *two months*, while the case is in *after closure phase*. In this period, UWV monitors cases for possible follow-ups. If an application gets rejected during the claim phase, *after rejection phase* starts, where UWV monitors rejected cases for possible objections and follow-ups.

Next we will discuss several compliance constraints and specific rules we derived from each constraint that unemployment process must adhere to. For this, we employed the *Compliance Constraint Framework* discussed in Sect. 3.1 to decompose the compliance constraints to specific rules each restricting one perspective of the unemployment process.

**Call clients before high impact decisions requirement.**

The first compliance constraint is taken from internal policies of UWV. This requirement aims at improving customer satisfaction. Based on a customer

satisfaction survey, UWV decided to inform its clients via the phone about its decisions. This is highly appreciated by clients. Especially clients whose applications were rejected for some reason and the rejection decision had been communicated with them before its official announcement, accepted the rejection decision better than those who had not been informed via phone about the decision prior to its official announcement. Even for accepted applications, applicants did object more often when the decision had not been communicated in advance via phone.

Furthermore UWV has asked explicitly its clients in the same survey if they would like to be contacted via phone about the decisions made for their applications. The answers have been mostly positive. Hence, **UWV has set the policy to inform clients about the decisions that have high impact on their applications before the official announcement of the decisions via mail.** In short, a decision is considered high impact when it influences the payment of benefits, the amount and the procedure of payment. This policy requires employees to call clients at most one day before the official announcement of a high impact decision. Together with domain experts in UWV, we decomposed this constraint to *four* specific compliance rules:

- *Rule (1):* There must be a *call* before a *high impact decision* is announced to a client.
- *Rule (2):* The time between the *call*, and the *announcement of a high impact decision* must not exceed one day.
- *Rule (3):* The calls given to clients before high-impact decisions must be registered as *high impact* in the UWV system.
- *Rule (4):* Every call registered as *high impact* must be followed by a *high impact decision*. That is, *regular calls* to clients must not be registered as *high impact*.

Violations of the first two rules (*Rules (1), and (2)*) may lead to unsatisfied clients. Violations of the third and fourth rules (*Rules (3), and (4)*) make it difficult for UWV to monitor its operation and will lead to an inefficient process.

**Decisions must be taken in correct application phase and must be announced correctly.**

The second compliance requirement focuses on the decisions made for an application in different phases. As mentioned earlier, each application will go

through different phases.

The Dutch national law requires that in each of these phases, certain decisions are allowed to be taken about the application of a client. In addition it is required that letters sent to clients to announce the decisions taken in each phase match the content of the decision. This seems to be trivial at the first glance, but this requirement is very important because it enforces that decisions to be made in the correct phase. Moreover, decisions must be announced correctly to the clients to avoid possible confusion.

Although this requirement seems straightforward, nevertheless complying to this requirement is not always easy. Correct phrasing of a decision to be announced to a client such that it prevents any possible confusion for an applicant and considers all the legal consequences may be difficult in some cases. Therefore, to avoid confusion for clients, legal experts in UWV have prepared standard letters for each type of decisions to phrase the decision as precise and clear as possible. These letters are predefined and categorized based on the type of decision message that they contain.

We decompose this compliance requirement to two specific compliance rules as following:

- *Rule (5):* Every letter announcing a *decision* that is taken about an application must match the *application phase*.
- *Rule (6):* Every letter announcing a *decision* must be sent corresponding to the *decision category* it belongs to.

Apart from legal consequences, violation of this rule can lead to extra work in UWV and consequently higher costs.

### **Ending of “weekly benefits” on Mondays.**

This compliance requirement restricts the closure of some benefits to be done only on Mondays. UWV pays unemployment benefits every four weeks to its clients. However, some benefits (for simplicity here we call them “weekly benefits”) are calculated on a weekly basis. These benefit may only end on Mondays. We derive one compliance rule from this requirement stating:

- *Rule (7):* Every *closure* of “*weekly benefits*” may only occur on a *Monday*.

This compliance rule is enforced in the business operation by the underlying information systems. That is, the underlying information systems do not allow for ending of “weekly benefits” on any day rather than Mondays. Violations of this rule may have legal consequences.

### Unemployment applications of clients with multiple applications in UWV must be handled by a specific team.

Sometimes clients of UWV apply for different kinds of benefits including unemployment benefits. These cases can be subject of different regulations and they need a careful processing. Hence UWV assigns these cases to a specific team (in this paper we call it *team T*) to process them. Employees in this team have the expertise to process these applications. We derived the following compliance rule from this requirement, stating:

- *Rule (8)*: Client cases with *multiple applications* must be processed by *team T*.

### Summary of rules.

Table 9.1 lists the compliance rule types that we derived from the constraints. These rules confine different perspectives of a business process including control-flow, process data, process resource and process time. This impacts the specification of these constraints together with the technique we use to check them.

As can be seen in this table *Rule (1)* specifies the sequence of activities. *Rules (2)*, and *(7)* specify when certain activities must be executed. *Rules (3)*, *(4)*, *(5)*, and *(6)* put constraints on data attributes of unemployment benefits process, and *Rule (8)* specifies the resource (team) that is allowed to process specific cases.

Compliance Rule	Rule Type
Rule (1)	Control-flow
Rule (2)	Temporal
Rule (3)	Data-aware
Rule (4)	Data-aware
Rule (5)	Data-aware
Rule (6)	Data-aware
Rule (7)	Temporal
Rule (8)	Resource-aware

Table 9.1: Compliance rule types to be checked.

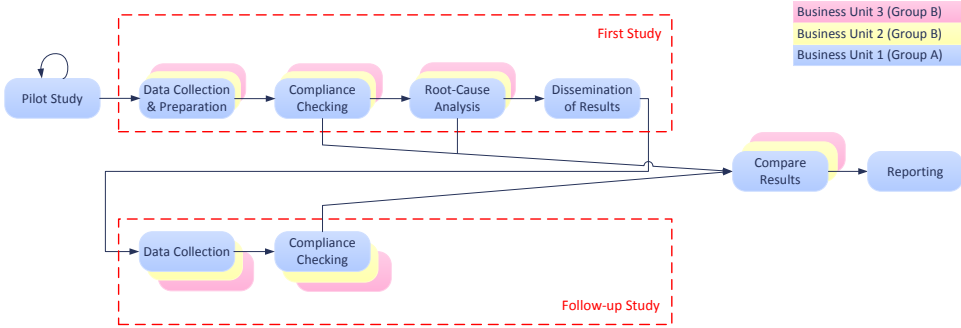


Figure 9.2: *Split testing* (A versus B) design of the case study including three business units and steps taken for each business unit.

## 9.2 Design of the Case Study

UWV has several business units in the Netherlands. We chose *three business units* to run our case study. We conducted the case study using *split (A/B) testing*. These units were divided into two groups with *Business Unit 1* being in the *A group* and *Business Unit 2*, and *3* in the *B group*. We did two rounds of analysis including *first study*, and *follow-up study* for both *A* and *B* groups.

For this, we first conducted a pilot analysis to refine the compliance rules and define our data requirements for the analysis. During the first analysis we collected a data set for the three business units and checked the compliance of the unemployment benefit process against the specified rules in the three business units. Further on we did a root-cause analysis for all the units as well. We disseminated the results and findings of our analysis only to *Business Unit 1 (A group)* to measure the impact of the diagnostics provided using our techniques. These results include overall statistics on violation of different compliance rules, detailed diagnostics about violations of each compliance rule, example of violating and non-violating cases. In addition, we share our root-cause analysis results including patterns that differentiate violations from compliant cases. After about two months, we repeated the analysis for the three business units in the follow-up study and monitored the changes in both groups. Figure 9.2 illustrates different steps we took in each study. In the following sections, we will discuss these steps in detail.



### 9.2.1 Setup

Here we discuss the techniques that we used in different steps of our case study including compliance rule elicitation, compliance checking and root-cause analysis.

#### **Compliance rule elicitation and specification.**

We used the elicitation technique presented in Chap. 4 for formalizing the control-flow of the rules listed in Table 9.1. We modelled the first four rules together as a composite compliance model because they were overlapping in activities. Although the elicitation technique presented in Chap. 4 focus on formalizing compliance rules in separate patterns, we could also use it for formalizing the control-flow perspective of the composite model. From configurable rule repository presented in Appendix C, we configured the pattern shown in Fig. C.6 for the control-flow of the composite compliance model.

We could use the pattern shown in Fig. C.1 for formalizing the control-flow of *Rule (5)* and *Rule (6)* if we would check different category of letters separately. However, we did the checking combined and built two Petri net models for *Rule (5)* and *Rule (6)*. However, the pattern shown in Fig. C.1 was configured and used as the control-flow of the *Rule (7)* and *Rule (8)*.

The data-aware Petri nets related to data-aware and resource-aware rule were built manually and for the temporal rules we used the technique in Chap. 5.

#### **Compliance Checking.**

We used compliance checking techniques presented in chapters 4, 5, and 6 for different rule types including control-flow, temporal, data-aware, and resource-aware compliance rules. By employing these techniques, we detected violations w.r.t. executions of activities and their sequence, the time they were executed, the data related to executions of these activities (e.g. application type, application phase, etc. ) and the resources (team, business unit, etc. ) that executed them.

#### **Root-cause analysis.**

We used the techniques presented in Chap. 7 to analyze the root-causes of identified violations and detect patterns that differentiated violations from compliant executions of activities.

Note that the technique presented in Chap. 7 is based on the result of checking a composite compliance model. However, we can use the same techniques for analyzing the result of checking atomic rules as well. As mentioned before we combined the first four rules in one composite model and the other four as separate atomic rules. For all of them, we use the techniques of Chap. 7 to do root-cause analysis.

### 9.2.2 Case Study Scope

As we discussed earlier, the case study was conducted for three business units. The compliance *Rules (1) to (7)* were analyzed for all the three business units and *Rule (8)* was specific for *Business Unit 1* and therefore was analyzed only in this unit.

From the eight compliance rules considered in this case study, the first *three* rules were of utmost importance for UWV because they were connected to the *improving customer satisfaction* project they have started earlier in the organization. Therefore, throughout the whole project more emphasis was put on these rules.

We did the pilot and the first study over the data taken from a two-month period including November and December 2014. The follow-up study is done over the data of June and July 2015. Figure 9.3 illustrates roughly the timing we spent on each project phase.

We disseminated results of our first study in two parts. First during a meeting where we discussed our results with the management team and domain experts in *Business Unit 1*. We discussed the various statistics we obtained about the violations during this meeting and discussed each in detail. Accordingly, we wanted to refine the compliance rules and our dataset. Consequently, we repeated the analysis. We continued with the root-cause analysis. We distributed the results of root-cause analysis with management team and domain experts in *Business Unit 1* in form of a workshop where we analyzed the results and received feedback about them.

Later we did the follow-up study with a new dataset and monitored the changes in each business unit w.r.t. different compliance rules.

## 9.3 Pilot Study

Pilot analysis of our project included the *precise elicitation of the compliance rules, definition of data requirements, collection of pilot data set and transforming*

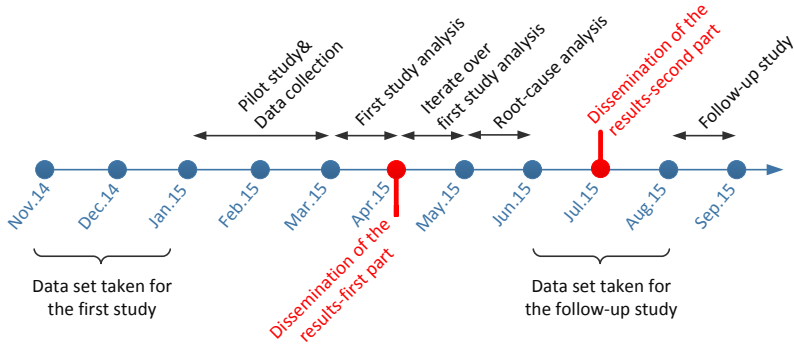


Figure 9.3: Different phases of the case study and their timing.

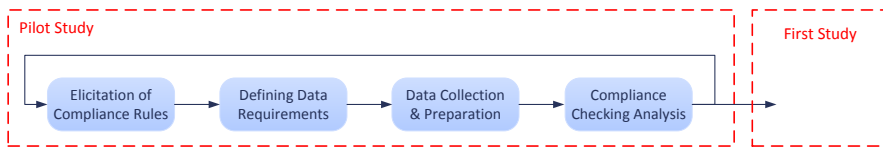


Figure 9.4: Different steps executed during the pilot study.

the data set to the format of an event log in XES structure, and compliance checking analysis. Figure 9.4 shows different steps we went through during the pilot study.

### Compliance rules elicitation and specification.

The first, and the most challenging part of the pilot study was the *elicitation of the compliance rules*. The elicitation of the rules was not straight forward. Due to the abstract definition of the rules in company guidelines, there was room for various interpretations about them. For instance, regarding the first two compliance rules stating: *there must be a call at most one day before a high impact decision is announced to a client*, the guideline did not specify what sort of decisions considered being *high impact*. Consequently together with domain experts in *Business Unit 1* we iterated over the specification of compliance rules. Even after showing the first analysis results, where we shared our results with a larger group than in the pilot study, we needed to change the definition of some compliance rules and repeat the first analysis.

### Defining data requirements, data collection and preparation.

Corresponding to the compliance rules we defined our data requirements. The required data included all the client cases that were relevant for the analysis, i.e., all the clients whose their unemployment benefits applications were received and processed during November and December 2014 in all the three business units. We included all the events that were relevant for the compliance rules in question including the calls given to clients, all the letters that were sent to clients, start of the benefits, closure of benefits, and other relevant events. However, we did not include events that would not be needed to check the compliance of one of the eight chosen compliance rules. In addition, together with experts from UWV, we chose all the attributes of the process and clients that we thought could be relevant for our analysis. Table 9.2 shows the list of process and client related attributes that we included in our dataset.

Client related attributes	Process related attributes
Age	Type of submission
Average working ours per week	Application phase
Entitled for additional allowance	Call registration
Number of contracts from previous jobs	Department contacted the client
Type of contract from the previous job	Letter title
Business sector of the previous job	Decision category
Reasons of unemployment	Day when a benefit was ended
daily wages of the previous job	Team
Living area	Business unit
	Time
	Simultaneous multiple applications

Table 9.2: Two groups of data attributes (process and client related) that we included in case study dataset.

The process related attributes include information about how an unemployment application is processed in UWV such as the business unit and the team that processed each application, the title of different letters that were sent to each client and the respective decision category, phase of an application, the office that contacted each client, and some more. The client related data attributes contain information about each application from the client perspective such as the age of the client, the business sector the client was working before unemployment, the number of hours per week the client used to work before,

the type of previous working contract, the previous salary, and some more. The client related attributes are defined on the process instance level, therefore they are global for a case and their values do not change. In contrast, most of the process related attributes are event specific and may change during the lifetime of a case. That is, they may be present in some events but not in others and their value may differ from one event to another within a case. We later used this information for root-cause analysis of violations. We would like to check if possible violations are related to client attributes or process attributes. This categorization is important. In case we would find process related attributes to be connected to a specific violation, the focus in compliance improvement will be on changing the process such that it supports the employees to be compliant in their operation. Whereas if we would find client related attributes to be connected to a specific violation, the focus in compliance improvement would be on changing the work culture in UWV.

Based on defined data requirements we chose the data sources that store our required data. We collected data from various systems including:

- *WVO* as the primary system for unemployment benefits. It includes event tables (activity executions in the unemployment process), event dimension tables, session table, and population table.
- *GCU* as the UWV standardized letters system. For the official announcements of decisions, there are templates available in this system that can be adjusted according to the message sent to clients.
- *K3CR* as the call center registration system. The date, and details of all the incoming calls from clients and all the outgoing calls from employees to clients are registered in this system.

Next to data from these systems, several other data sources were combined to derive whether a client receives other benefits than unemployment benefits or not. Figure 9.5 illustrates a simplified overview of our data collection process. In total we collected data from 6 different information systems and dozens of tables were used. The data from different source systems of UWV were extracted and stored in a Microsoft SQL Server database environment. The raw data, that is supplied monthly, goes through an ETL (Extract Transfer Load) process and ends up in data marts. After choosing the data sources we built the required data marts from the data warehouse. In total we connected 12 data marts and 5 tables containing master data. The data marts contain event data (e.g. decisions taken for clients in each phase, start of benefits, and closure of

benefits), the application phase of clients, detailed information about different unemployment benefits, the unemployment benefits each client are entitled to receive and many more process specific information.

Finally, we queried our analysis sets first for *Business Unit 1* during the data collection step of the pilot study and later for the three business units during the data collection step of the first study. We used the technique in [43] to prepare and transfer the data set to the format of an event log compatible with XES structure [57].

**Compliance checking.**

After data preparation, we checked the compliance of the data set against all the *eight* compliance rules in question. We checked compliance in the pilot analysis only for *Business Unit 1*. We used the technique presented in 4 for checking the control-flow compliance rule (*Rule (1)*), the technique presented in 5 for compliance checking of the temporal rules (*Rule (2)*, and (*7*)), and the technique presented in 6 for data-aware and resource-aware compliance rules (*Rules (3)*, (*4*), (*5*), (*6*), and (*8*)).

After we obtained the results we discussed them in form of some presentations and using the *Compliance Dashboard* (see Sect. 7.5) with the management team of *Business Unit 1* and iterated over the process several times until we refined the compliance rules to an acceptable extent and prepared data accordingly. We still observed some disagreement among the management team of *Business Unit 1* regarding the boundaries of what is considered to be compliant or violating for some compliance rules, yet the specification was good enough to continue the analysis with them.

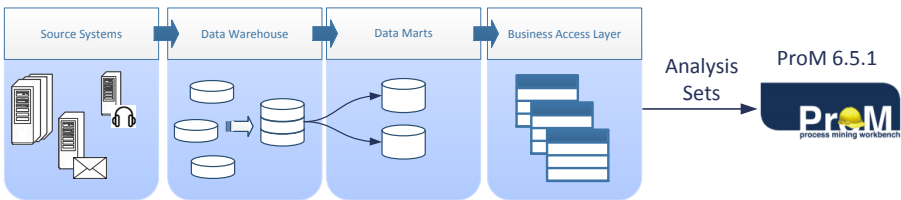


Figure 9.5: Collecting data from different data sources.

## 9.4 Results and Observations of the First Study

After establishing the correct compliance patterns for the eight rules and defining the correct data extraction, we conducted the first analysis. We extracted data from period of two months for each of the three business units (see Fig. 9.3).

Table 9.3 shows the number of cases together with the number of events that we analyzed in the first study for each business unit. Please note that we only included events in our data set that were relevant for the compliance rules. Hence, the number of events are not that large compared to the number of cases we analyzed.

Business Units	# Cases	# Events
BU 1	24077	91602
BU 2	32269	113565
BU 3	35182	136461

Table 9.3: The volume of cases and events in datasets for each business unit.

Next we will discuss our findings for each business Unit separately. We will also discuss the feedback we received from domain experts after sharing our results in *Business Unit 1*.

### 9.4.1 Results of the First Study: Business Unit 1

Table 9.4 contains the total number of rule instances and the violation frequency of each rule that are observed during the months of November and December for the three business units. In this section we focus on the results of *Business Unit 1*.

**Violations of compliance Rule (1).** In 58% of all 3527 situations where it was required that UWV calls a client in *Business Unit 1*, there was no call. Hence, there were  $(0.58 * 3527 = 2046)$  violations of *Rule (1)*. For this rule, we checked whether activity *send letter* for letters that contain a *high impact decision* were proceeded by activity *call* or not. The  $(100\% - 58\% = 42\%)$  compliant executions of the sequence  $\langle call, \dots, send\ letter \rangle$ , do not reveal much information because activity *call* could have been executed long before activity *send letter* for other reasons rather than informing the client about the *high impact decision*, but the violation percentage (i.e., 58%) reveals important information about the number of *missing calls*.

	Rule(1)	Rule(2)	Rule(3)	Rule(4)	Rule(5)	Rule(6)	Rule(7)	Rule(8)
BU 1	Violation	58%	66%	23%	10%	15%	8%	32%
	Total	3527	3527	3527	6861	6476	2930	1218
BU 2	Violation	32%	43%	22%	23%	15%	8%	2%
	Total	4281	4281	4281	8756	8120	3779	3485
BU 3	Violation	54%	66%	38%	2%	18%	10%	3%
	Total	8040	8040	8040	16365	11606	3379	93014

Table 9.4: Total number of rule instances and the violation frequency per compliance rule for each business unit.



**Violations of compliance Rule (2).** The violation percentage of the second rule shows that 66% of the calls were given to clients longer than one day before a *high impact decision* was officially announced to them. Since we checked the delay between the occurrence of activity *call* and activity *send letter* that contained a *high impact decision*, we were able to compute the distribution of violations for different category of letters that were sent. Table 9.5 shows for all the three business units the absolute number of letters sent to clients in each *letter category* and the violation frequency per letter category. The letters of categories (A), (B), (C), and (D) are sent to clients to announce different high impact decisions. The results suggest that by improving compliance especially in the second category (letter B), the overall compliance w.r.t. *Rule (2)* will improve dramatically.

Activity	BU 1		BU 2		BU 3	
	Total	Violations	Total	Violations	Total	Violations
Send letter category (A)	1441	40%	1658	22%	2845	29%
Send letter category (B)	2029	84%	2529	54%	5121	86%
Send letter category (C)	15	93%	30	80%	11	100%
Send letter category (D)	42	85%	64	78%	63	86%

Table 9.5: Total number of letters per category and distribution of violations w.r.t. compliance *Rule 2* over letter categories.

The first column shows the distribution of violations of *Rule (2)* in *Business Unit 1* over the four different letter categories. For instance, in total 1441 letters of category (A) were sent to clients in *Business Unit 1*, of which 40% were violating *Rule (2)*.

**Violations of compliance Rules (3), and (4).** Table 9.4 shows that the fraction of violations for *Rule (3)* in *Business Unit 1* is 23%. In these cases the calls given to clients prior to a *high impact decision* were not registered as *high impact*. Moreover, we observe that 10% of regular calls given to clients (i.e., the calls that were not followed by a *high impact decision*) were registered as *high impact* (violations of *Rule (4)*).

The first four compliance rules are derived from the same constraint and they are related. As we discussed earlier, compliance or violations of these rules (especially the first two rules) are perceived to influence the *customer satisfaction* in UWV. Therefore, improving the compliance of these rules has a high priority for UWV.

We discussed these results intensively with the management team in *Business*

*Unit 1* and they speculated upon various reasons that could explain the high number of violations for these rules. These reasons were as following:

- Although the guideline on calling a client prior to a *high impact decision* is not new in UWV, it is only recently that its influence on *customer satisfaction* has been revealed and hence got priority in the organization. Therefore, this guideline is not yet fully incorporated in the daily operations of UWV.
- Unfortunately the guideline is not clear enough and does not concretely specify what is considered to be a *high impact decision*. Hence, the employees themselves should judge whether they need to call clients or not.
- The underlying information systems used for instance for registering calls is not very intuitive. Consequently some employees do not exactly know how they should register calls properly (subject of *Rule (3)*, and *(4)*).

**Violations of compliance Rule (5).** In Table 9.4, we observe 15% violations for compliance *Rule (5)* in *Business Unit 1*. This value shows that 15% of the letters that were sent to clients to announce a decision, were not sent in the correct application phase. As we discussed earlier, different categories of letters are sent to clients. Table 9.6 shows the absolute number and the distribution of violation frequency per letter category.

Activity	BU 1		BU 2		BU 3	
	Total	Violations	Total	Violations	Total	Violations
Send letter category (A)	1441	20%	1658	18%	2845	41%
Send letter category (A-2)	9	33%	13	54%	19	63%
Send letter category (B)	2029	23%	2529	24%	5121	11%
Send letter category (B-2)	67	45%	141	52%	242	35%
Send letter category (C)	15	7%	30	17%	11	9%
Send letter category (D)	42	26%	64	19%	63	5%
Send letter category (E)	2794	5%	3577	5%	3165	7%
Send letter category (E-2)	79	35%	108	39%	140	36%

Table 9.6: The total number of letters per category and distribution of violations w.r.t. compliance *Rule 5* over letter categories.

We discussed these violations with domain experts in *Business Unit 1*. They believe a portion of these violations could be related to the fact that it is not always easy to draw a line between different phases that an application goes

through. Consequently, sometimes an application has moved to the next phase, yet some steps needed to be taken that corresponds to the previous phase. Therefore, although it is good to decrease the number of violations for this rule, domain experts did not find the 15% violations a high number or severe and they believe a large portion of these violations are false positives.

Activity	BU 1		BU 2		BU 3	
	Total	Violations	Total	Violations	Total	Violations
Send letter category (C)	15	20%	30	13%	11	27%
Send letter category (D)	42	12%	64	16%	63	11%
Send letter category (E)	2794	8%	3577	7%	3165	10%
Send letter category (E-2)	79	15%	108	15%	140	11%

Table 9.7: The total number of letters per category and distribution of violations frequency w.r.t. compliance *Rule 6* per letter category.

**Violations of compliance Rule (6).** The 8% of violations shown in Table 9.4 for the compliance *Rule (6)* in *Business Unit 1* are related to letters that contain a decision announcement but they were not sent according to an appropriate *decision category*. Similar to previous rules, Table 9.7 shows the absolute number and the distribution of violations for the compliance *Rule (6)* per letter category.

**Violations of compliance Rule (7).** The 2% violations for the compliance *Rule (7)* in *Business Unit 1* (Table 9.4), although is not high, were perceived to be interesting by the management team in this unit. For checking this compliance rule, we analyzed activity *benefit closure* for all “weekly benefits” and checked if they occurred on a *Monday* or not. Note that, this compliance rule is enforced by the system, i.e., the underlying information system should not allow to end “weekly benefit” on days other than Mondays. Hence, “no violations” were expected for this rule.

**Violations of compliance Rule (8).** The 32% violations observed for compliance *Rule (8)* shown in Table 9.4 in *Business Unit 1* are related to clients that have multiple applications running simultaneously in different departments of UWV. We discussed this result with domain experts in *Business Unit 1*. They believe to reduce violations of this rule, more system support is required. Several systems have been developed within UWV to handle different types of applications. These systems are not always well integrated. Hence, sometimes it is not easy for employees in one department to be aware of other applications of the same client being processed simultaneously in other systems.

### 9.4.2 Results of the First Study for Business Unit 2 and Business Unit 3

During the first analysis, we also checked compliance *Rules (1) to (7)* in *Business Unit 2* and *Business Unit 3*. Table 9.4 shows the violation frequencies and the absolute number of rule instances observed for these units. Note that we checked the compliance *Rule (8)* only for *Business Unit 1* because we did not have the required data for checking this rule in other business units.

Figure 9.6 compares the share of violations of business units per compliance rule. As can be seen, the different business units widely vary w.r.t. the compliance of *Rules (1), (2), (3), and (4)* but the difference gets lower for the last three compliance rules.

Our results show that less violations are observed for *Business Unit 2* especially for the first three compliance rules which are the focus of UWV in this case study. As explained previously in Sect.9.2, we did not distribute the results of the analysis within the *group B (Business Unit 2 and 3)*. However, we discussed the results with the management team and domain experts of *Business Unit 1*. They believe that the differences relate to the initiatives that *Business Unit 2* took earlier than all other units to execute the guideline about ‘*call clients before high impact decisions*’. Therefore, employees in this unit had more time to get acquainted with this compliance requirement and its execution within their operations.

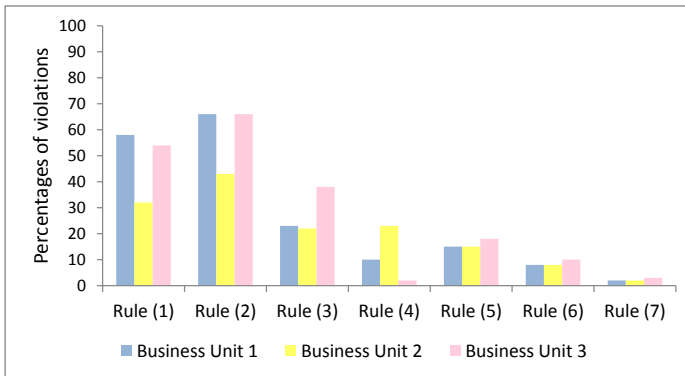


Figure 9.6: Comparing compliance results obtained during the first study for different business units.

Another interesting observation in Fig. 9.6 refers to the compliance *Rules (3)*, and *(4)* that are both about the registration of calls. One rule (*Rule (3)*) specifies that *important* calls must be registered as such and the other rule (*Rule (4)*) specifies that *regular* calls should not be registered as *high impact*. It seems that *Business Unit 2* put too much emphasis on compliance *Rule (3)*. Hence, although they show the lowest number of violations for other rules compared to other units (the yellow bar in Fig. 9.6 has the lowest percentage in all rules except *Rule 4*), they tend to register most of the calls given to clients as *high impact* which led to higher number of violations for *Rule (4)*.

## 9.5 Root-Cause Analysis and Dissemination of Results

In Sect. 9.4, we discussed the violations detected by the compliance checking approach described. We did a general root-cause analysis for all the rules and violations detected independent from specific observations explained in Sect. 9.4.

	BU 1	BU 2	BU 3
# Association rules	7254	7103	6801
# Filtered association rules-Process related	54	45	88
# Filtered association rules-Client related	30	31	37

Table 9.8: Association rules mined for different business units between violations and their contextual data.

As we explained earlier in Sect. 9.2, we enriched our dataset with two groups of data attributes including process related attributes and client related attributes. We used this contextual data to analyze the detected compliance violations further. For this we mined all the association rules between values for these two groups of data attributes and different types of violations using the technique presented in Chap. 7. Table 9.8 shows the large number of association rules we obtained for each business unit. In total we found 7254 association rules in *Business Unit 1* distributed over different violations. We filtered these rules based on their frequency and significance. We evaluated the significance of an association rule using a strength metric accepting a value between  $[0-1]$  (for details about this metric (CPIR) please read Sect. 7.5.2). We filtered the association rules and kept only those that have a *frequency higher than 40*, and *strength*

metric more than 0.3. After trying several filtering configurations together with a domain expert, we found this balance between the frequency, and strength metric most suitable. We found out that a frequency higher than 40, is high enough to filter out less important association rules and low enough to contain the interesting ones. Similarly we also found that 0.3 for strength metric filters out many less important association rules and keeps the interesting ones. We divided the filtered association rule into two groups of client related attributes and process related attributes. For instance Table 9.8, shows 54 association rules in *Business Unit 1* indicating a correlation between different violation types and process related attributes that remained after filtering.

Next we will discuss our observations related to two groups of associations. The detailed analysis of the remaining association rules can be found in Appendix D.

### 9.5.1 Analyzing Process Related Information

Figure 9.7 depicts the association rules (after filtering) that we mined between different violations and process related attributes in *Business Unit 1*. The brown ellipses in the middle, titled with the compliance rule numbers, show the violations of respective compliance rules. The other nodes refer to values of different attributes and they are grouped based on their attribute.

The association rules that we have found are directed. The direction of rules are specified by their color. The blue dashed lines show an association rule of

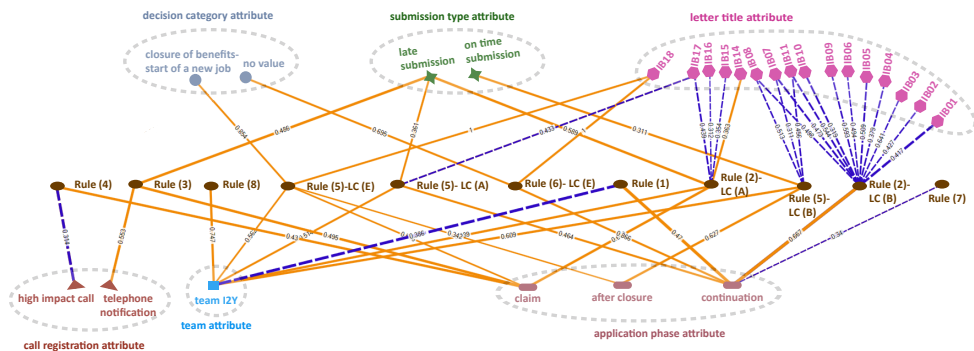


Figure 9.7: Filtered association rules mined between different violations and process related attributes in *Business Unit 1*.

type *attribute value*  $\rightarrow$  *violation* (i.e., attribute value implies the violation) and the solid lines colored in orange show an association rule of type *violation*  $\rightarrow$  *attribute value* (i.e., violation implies an attribute value). The association rules in principle do not show causality between the two nodes of the rule. However, we think the possibility that we can predict a violation is higher if the direction of the association is from the attribute value towards the violation. For instance, if a specific attribute-value assignment implies a violation, we can predict that future executions of activities with that attribute-value assignment are probably violating.

The width of the line between two nodes represents the frequency of the association rule observed, the thicker the more frequent. The numbers written over the lines indicate the value of the association rule strength metric, a value between [0,1] with 1 showing a strong association and 0 no connection. Note that the associations shown in Fig. 9.7 have at least frequency of 40, and value of 0.3 for the strength metric.

For instance, the second right ellipse titled *Rule (2)-LC (B)* shows a group of letters from category (B) that were sent to clients without a prior on time call. These violations are associated with a group of *letter titles* and attribute *continuation application phase*. The orange line between *application phase = continuation* and *Rule (2)-LC (B)* shows that: ‘violation type *Rule (2)-LC (B)* implies *application phase = continuation*’. That is, we can say usually when we see a violation of type *Rule (2)-LC (B)*, we expect it to have occurred in *continuation phase* of an application. Similarly the association rule between *Rule (2)-LC (B)* and *letter title = IB01* shows that ‘*letter title = IB01* implies violation of type *Rule (2)-LC (B)*’. We can say usually when we see a *letter title* to be *IB01*, we expect the violation of type *Rule (2)-LC (B)* to have occurred. Next, we will discuss some of the interesting association rules that we observed.

**Associations between violations of *Rule (2)*, and different *letter titles*.** As can be seen in Fig. 9.7, violations of *Rule (2)* related to either *letter category (A)*, or *(B)* are connected with a specific group of letter titles. We discussed in detail these letters in a workshop with domain experts. They believed they knew that their operation is violating especially *Rule (2)* but the detailed data they received on the level of letter titles helps them get insight and make the assumptions concrete and measurable. It was interesting for them to know which letter titles are connected to violations of *Rule (2)* and where they need to focus to improve the compliance w.r.t. *Rule (2)*.

Domain experts in *Business Unit 1* speculate that some of these letters may not be interpreted by employees as *high impact*. Hence, the employees did not

feel the necessity to inform the clients one day before the letter sent but the call may have happened a week before or even more.

We discussed the possibility to improve this situation. It seems first of all improving the guideline, to make it concrete and clear about the letters that are considered *high impact*, is necessary for improving the compliance of *Rule (2)*. In addition, raising awareness among employees and constant emphasis of management on the importance of complying with *Rule (2)* is required. Especially for the latter, after we discussed our results of the first study, UWV has started a regular report to be distributed monthly throughout the whole organization to show the process compliance of the organization within each period. This report is accessible by all employees active in the *unemployment* process in the three business units.

We checked other business units if we find common patterns between violations of *Rule (2)* and attribute *letter titles*. We checked whether in other business units, the violations are connected to the same letter titles that we found in *Business Unit (1)* or different ones. For this, we mined the association rules between different violations and different attribute values for other business units as well. Similar to *Business Unit 1*, we filtered the association rules for those with a frequency more than 40 and strength metric value more than 0.3. Figure 9.8 shows the result of our comparison of the association rules in three business units between violations of compliance *Rule (2)* and different letter titles.

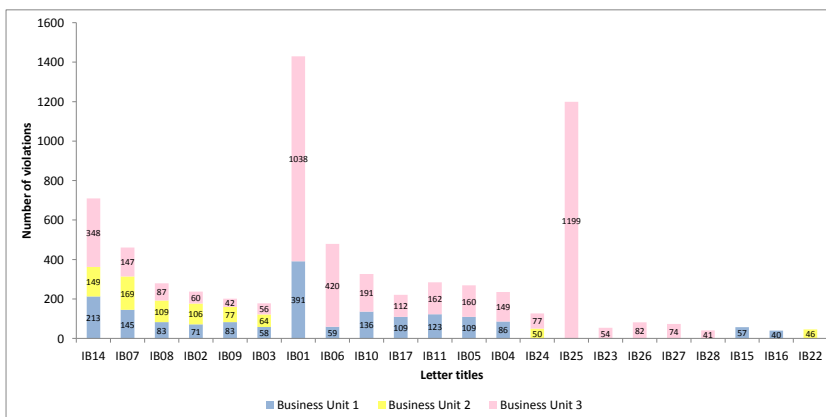


Figure 9.8: Letter titles connected to violations of *Rule (2)* in each business unit.



As can be seen in this histogram, there are several letter titles in relation with violations of compliance *Rule (2)* that are common between all three different business units. These include *IB14*, *IB07*, *IB08*, *IB02*, *IB09*, and *IB03*. We also found some letters that are common between *Business unit (1)*, and *(2)*. Similarly the letter title *IB24* is common between *Business unit (2)*, and *(3)*. Some of the letter titles are specific to one business unit. Domain experts in *Business unit (1)* believe that the interpretation of *letters containing high impact decision* seems to differ not only among employees of one unit but also there are differences among the way business units have categorized letters as high impact.

This observation once more urges the necessity to improve guidelines and make the concepts of *letters with high impact decisions* clear and concrete for the employees in all units. For instance in total 406, 346, and 1103 letters with title *IB01* were sent respectively in *Business Unit 1*, *2*, and *3* from which 391 in *Business Unit 1*, and 1038 in *Business Unit 3* were violating. As can be seen this letter title was not considered to have *high impact decision* according to interpretation of *Business Unit 1*, and *3* whereas it was considered to be a letter with *high impact decision* in *Business Unit 2*.

We also see that the letter title *IB25* is strongly connected with the violations of *Rule (2)* in *Business Unit 3*. From the total number of 1237 number of letters sent to clients with *IB25* title, 1199 violated *Rule (2)* with the strength metric value of 0.429. However, this letter title is not a very good example for comparing the three business units because it was sent seldom (with frequency less than 10) in the other two business units. However, it is interesting to see that this letter is not considered to contain a *high impact decision* in *Business Unit 3*.

**Associations between *team I2Y*, and different violation types.** Another interesting observation in Fig. 9.7 is related to the associations between *team I2Y* and violations of *Rule (1)*, *Rule (2)-LC (A)*, *Rule (5)-LC (B)*, *Rule (5)-LC (A)*, *Rule (5)-LC (E)*, and *Rule (8)*. Particular the association rule between *Rule (8)* and *team I2Y* reveals that in most of the times when we expected that a client case with multiple applications to be processed by *team T*, it is processed by *team I2Y*. We discussed this finding with domain experts. They speculate that this observation is related to the point in the process when cases are assigned to teams and in these cases instead of *team T*, they have been assigned to *team I2Y*. Note that each team processes cases with distinct characteristics. *Team I2Y* have members that are not trained for specific cases, but they are skilled to be able to handle various types of cases. Hence, *team I2Y* is responsible for cases that usually cannot be assigned to other teams. Consequently it seems that most of

the complex cases that were not clear to which team they should be assigned to, were assigned to team *team I2Y*. For instance, usually cases where clients move from one city to another are more complex due to some information loss. In general conditions of a case may change during the processing that changes the way it should be processed. In such cases other teams prefer these cases to be assigned to *team I2Y* that have general skills to handel cases of different types. Consequently different violations are observed in connection with this team. We discussed possible changes in the process to improve compliance w.r.t. this observation. It seems that better system support is required that ensures all the information of a case is available and assigns cases more precisely to teams based on their characteristics.

We checked the association rules in other business units to see if we can find the same pattern or not. For this, we checked whether teams with the same functionality as team *team I2Y* in other business units are connected to different violations. Table 9.9 shows the associations of different violations with *team I2Y* together with the direction of associations in different business units. Note that  $V \rightarrow A$  indicates that violation ( $V$ ) implies the attribute value ( $A$ ), and  $A \rightarrow V$  indicates that attribute value ( $A$ ) implies the violation ( $V$ ). *Business Unit 2*, and *3* are common in associations between this team and violations of *Rule (7)* and *Business Unit 1*, and *3* are common in associations between this team and violations of *Rules (2)*, and *(5)*. Note that *Rule (8)* is checked only in *Business Unit 1*.

Violations	BU 1	BU 2	BU 3
Rule (1)	$A \rightarrow V$		
Rule (2)	$V \rightarrow A$		$V \rightarrow A$
Rule (5)	$V \rightarrow A$		$V \rightarrow A$
Rule (7)		$V \rightarrow A$	$V \rightarrow A$
Rule (8)	$V \rightarrow A$		

Table 9.9: Filtered associations between different violations and *team I2Y* in the three business units.

### 9.5.2 Analyzing Combination of Process Related Attribute Values with Occurrences of Different Violations

Association rule mining helps us find out whether a particular attribute value assignment is connected with a violation. We use contextual data of violations (i.e., process and client related attributes) further to detect conditions (i.e., combination of attributes and their values) under which a certain violation may occur. For this, we choose a combination of attributes and check w.r.t. a specific rule, how compliant and violating activities differ from each other. We detect these patterns using the classification techniques presented in Chap. 7. We have 21 data attributes including process related and data related attributes which force us to consider  $2^{21} - 1$  combinations of attributes. Classifying violations and non-violations of a compliance rule with each of these combinations is tedious. In addition, we would like to combine attributes such that we would have the most information gain. An option would be to use a feature selection technique [45, 90, 154] to choose groups of attributes which are dependent. However, we used two sources of information for choosing appropriate combination of attributes. We used domain expert knowledge and also the association rules we mined previously. Using this information, we chose six different-sized groups of attributes to check whether violations and non-violations of a compliance rule differ from each other w.r.t. these combination of attributes. The grouping of attributes is shown in Table 9.10.

From these six groups of combined attributes, we only checked some for each rule. Because some of the data attributes are local for specific activities. For instance the attribute *call registration* which stores how employees registered a phone call with a client, is local to the activity call. Hence, it is useless to classify violations and non-violations of activity *call* (for instance w.r.t. *Rule (8)*) based on this attribute value.

After classifying violations w.r.t. different combination of attribute values, we found very few violating and compliant patterns between the combination of process related attribute values and different violations. We found also many situations where compliant and violating activities could not be classified based on a combination of attribute values. Next we will discuss some of our findings.

We classified violations and non-violations of *Rule (1)* with the combination of attributes '*submission type, application phase, and Team*'. We filtered out detected patterns that had frequency lower than 40, and confidence measure lower than 50%. Figures 9.9, 9.10, and 9.11 respectively show the violating and compliant patterns w.r.t. *Rule (1)* for *Business Unit 1, 2, and 3*.

As can be seen in Fig. 9.9, the combination of '*late submission and claim*

Attribute names	Grouping
daily wages working hours/week	1
number of contracts type of contracts reasons for unemployment	2
daily wages working hours/week number of contracts type of contracts reasons for unemployment entitled for additional allowance	3
letter title decision category	4
submission type application phase call registration team	5
letter title decision category submission type application phase team call registration	6

Table 9.10: Grouping of attributes for classifying violations versus non-violations.

phase' together with different teams reveals a compliant pattern in *Business Unit 1*. Similarly, we can consider the combination of 'team I2Y and continuation phase' as a violating pattern. Note that this result conforms with the association rules we found for violations of *Rule (1)* that is shown in Fig.9.7. However, we can not conclude the same for a specific type of submission as it seems violations are happening in all three types of it.

Figure 9.10 illustrates the classification of violations and non-violations of *Rule (1)* based on the same set of attributes namely: *submission type*, *team* and *application phase* in *Business Unit 2*. As can be seen: the information gain in this tree is very low. Apart from connecting non-violations to *late submissions*, we cannot conclude much. The information gain is even lower in *Business Unit 3*.

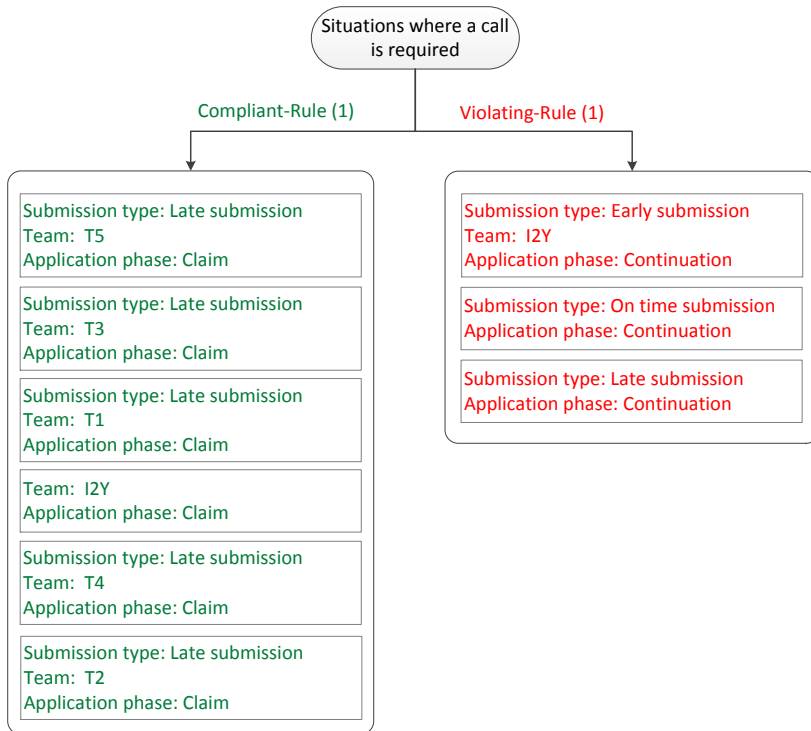


Figure 9.9: Violating patterns that distinguish compliant and violating activities w.r.t. compliance *Rule (1)* in *Business Unit 1*.

This result is shown in Fig. 9.11 for classifying violations and non-violations of *Rule (1)* with the same set of attributes. However, we can predict that most probably, *after closure* and *continuation* phases are more prone to have violations of *Rule (1)*. We expect applications in *after rejection* and *claim* phases to be compliant with *Rule (1)*.

We checked if we can differentiate compliant and violating activities w.r.t. compliance *Rule (2)* and combination of attributes *letter title* and *decision category*.

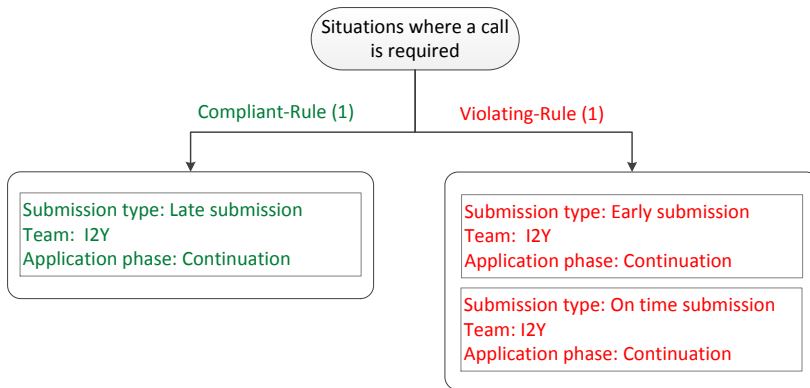


Figure 9.10: Violating patterns that distinguish compliant and violating activities w.r.t. compliance *Rule (1)* in *Business Unit 2*.

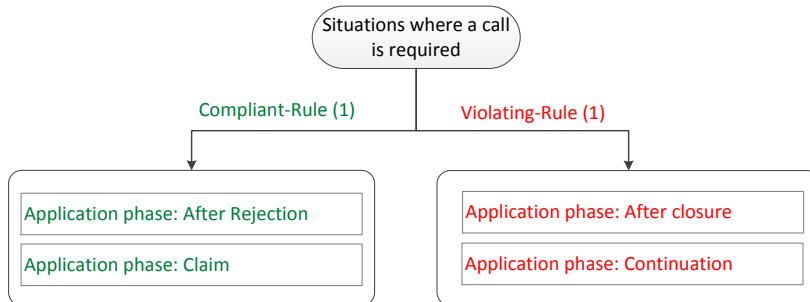


Figure 9.11: Violating patterns that distinguish compliant and violating activities w.r.t. compliance *Rule (1)* in *Business Unit 3*.

This information helps us understand conditions under which we expect a violation of *Rule (2)* to occur. We could detect some patterns that differentiated violations from non-violations of *Rule (2)* in *Business Unit 1*. However, after filtering for those patterns that have frequency higher than 40 and confidence measure more than 50%, only one interesting pattern for violations remained. The decision tree shown in Fig. 9.12 indicates in *Business Unit 1* the condition i.e., combination of ‘letter title: *IB17* and decision category: (GU) NIET WERK. IVM GEEN URENVERL. CQ LOONVERL, under which we expect violations of *Rule (2)* to occur. This information can be used for prediction purposes and improving compliance.

An interesting observation can be made w.r.t. letter title: *IB14*. This information at the first glance seems to be contradictory with the association rules we mined that connect violations of *Rule (2)* and letter title: *IB14* (Please see this association rule in Fig. 9.7). This “contradiction” can be explained by the direction of the association rule that connects violations of *Rule (2)* with letter title: *IB14*. This association is directed towards the value *IB14* for letter title attribute, meaning that violations of type *Rule (2)* implies that letter title will be *IB14*. Therefore, we expect that whenever we observe a violation of *Rule (2)*, letter title *IB14* to be present. The reverse direction may not hold (shown in the respective decision tree in Fig. 9.12).

We checked other business units as well to see if we can detect distinguishing patterns between violations and non-violations of *Rule 2*. We could not detect any interesting pattern in *Business Units 2* and *3*.

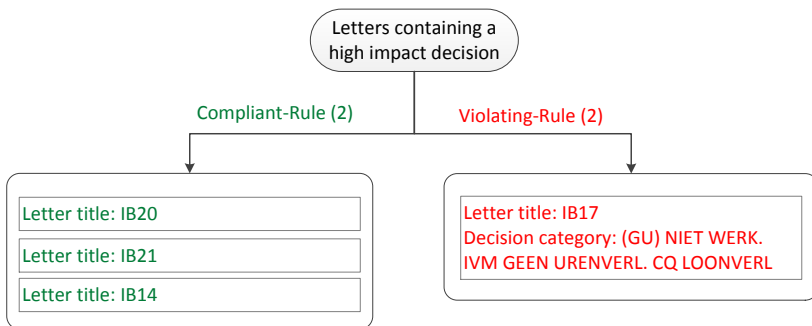


Figure 9.12: Violating patterns that distinguish compliant and violating activities w.r.t. compliance *Rule (2)* in *Business Unit 1*.

### 9.5.3 Analyzing Client Related Information

Similar to process related attributes, we used client related contextual data to analyze detected compliance violations further. Hence, we mined all the association rules between different values of client related data attributes and different type of violations. We used the same filtering procedure as before to keep important association rules (i.e., association rules with frequency higher than 40 and strength metric more than 0.3). As shown in Table 9.8, the resulting association rules mined after filtering are 30, 31, and 37 respectively for *Business Unit 1*, 2, and 3.

Figure 9.13 depicts the association rules that we mined between different violations and client related attributes in *Business Unit 1*. As explained earlier, the brown ellipses in the middle titled with the compliance rule numbers, show the violations of different compliance rules. The other nodes refer to values of different attribute values and they are grouped based on different attributes.

From nine client related attributes we found connections between violations and six of these attributes in *Business Unit 1*. In general we can say that we did not find strong relations between characteristics of clients and different violations. This suggests that violations are almost independent from client profiles. This message was positively received by *UWV* since it shows that employees are not biased towards specific clients. Therefore, improvement in compliance will be achievable by taking corrective measures in the underlying business processes and providing better information system support. The details of associa-

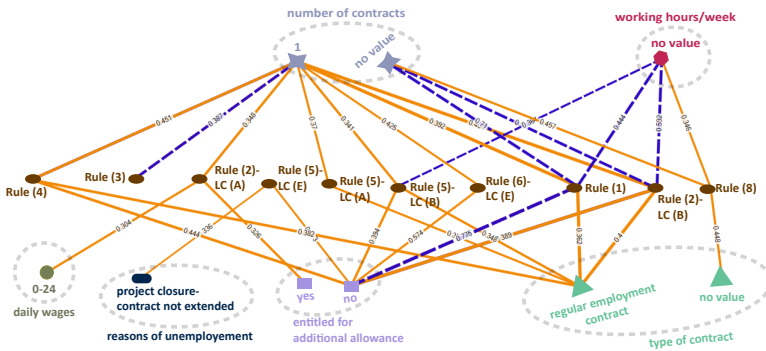


Figure 9.13: Association rules mined between different violations and client related attributes in *Business Unit 1*.



tions mined are discussed in Appendix D.

#### 9.5.4 Analyzing Combinations of Client Related Attribute Values with Occurrences of Different Violations

We checked different combination of attribute values to see if we can find violating patterns that distinguish compliant and violating activities w.r.t. a specific compliance rule. Next, we will discuss a few of these combinations that were more interesting than others.

For instance we checked if we can differentiate compliant and violating activities w.r.t. compliance *Rule (1)* and combination of attributes ‘*average working hours per week, number of contracts, type of contract, and business sector*’. This information helps us understand conditions under which we expect a violation of *Rule (1)* to occur.

The classification of violation versus non-violations shown in Fig. 9.14 indicates for *Business Unit 1*, the conditions (i.e., combinations of attribute values) where violating and compliant activities w.r.t. *Rule (1)* are distinguished.

As can be seen in Fig. 9.14, cases that were compliant with compliance *Rule (1)* have typically ‘*average working hours: [0 – 3]*’ and ‘*business sector: J.Uitzend*’. The information on their *type of contract* was missing. We also found some violating patterns among cases that were violating compliance *Rule (1)*. For instance, we can expect cases having ‘*[40 – 43]*’ *average working hours per week in financial services with regular employment contract*’ to violate *Rule (1)*.

The same analysis for *Business Unit 2* did not yield interesting results. Figure 9.15 illustrates for *Business Unit 3*, violating and compliant patterns w.r.t. compliance *Rule (1)* and discussed combination of client attributes.

As is shown in this classification, one of the interesting violating patterns is related to the combination of ‘*[75 – 99]*’ for daily wages, ‘*[36 – 39]*’ for average working hours per week, ‘*project closure-contract not extended*’ for reasons of unemployment and ‘*regular employment contract*’ for type of contracts.

#### 9.5.5 Summary of Observations from the First Study

Conducting the first study, we got many insights about the violations and their possible reasons. We conducted several workshops with domain experts in *Business Unit 1* to discuss the results. During these discussions the management of *Business Unit 1* was present. The presence of domain experts varied depending on their work load. The interesting result is that, the presence of domain

experts in workshops is reflected in the results of the follow-up study. Within these workshops discussions were mostly focused on the results we obtained in *Business Unit 1*. Nevertheless, we discussed the results of other business units with *Business Unit 1* as well.

Domain experts also found that violations are more dependent to the process of ‘handling unemployment benefits’ itself rather than client characteristics. We received positive feedbacks from domain experts and management team in UWV such as: “compliance checking make our assumptions concrete and measurable”, or “compliance checking provides us with a lot of detailed information that we can act upon”, or “we knew we are not always calling our clients on time, but we did not know how compliant or violating we are” or; “in addition to the interesting insights that compliance checking provided us with, the project started a big discussion inside the organization about the definition of letters containing *high impact decisions*”.

UWV started some measures to improve compliance based on the results of

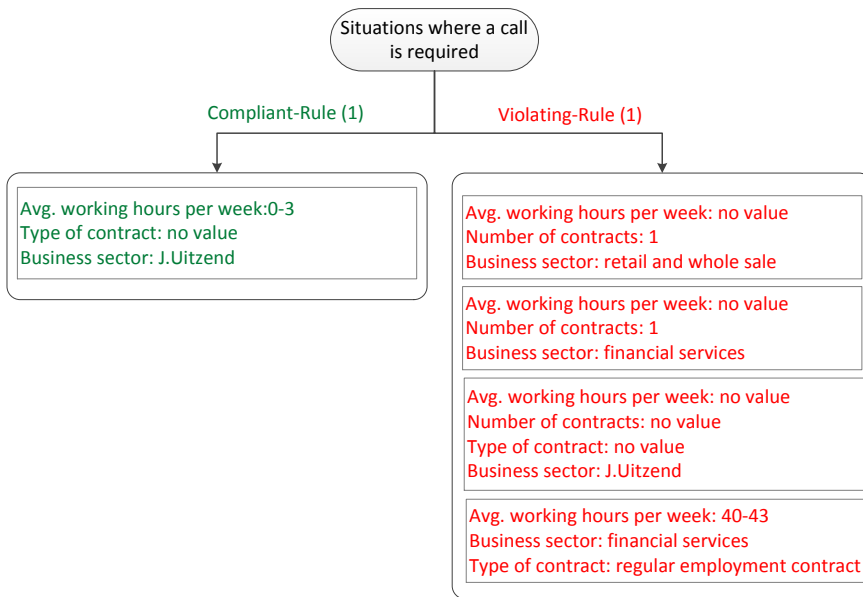


Figure 9.14: Violating patterns that distinguish compliant and violating activities w.r.t. compliance Rule (1) in *Business Unit 1*.

our first study. Some of these measures have been implemented improvement yet. Several ideas and proposals are still being discussed at different layers of the organization to get approved and executed. Nevertheless, the results have been acted upon by UWV. In the following we will discuss some of these measures.

UWV launched a regular report on compliance for the first four compliance rules. This report gets distributed monthly in all business units throughout the whole country to raise awareness about the importance of these rules and help different units to observe and monitor their compliance status. This report provides detailed statistics about the violations of the first four compliance rules. In addition UWV has started an internal discussion to improve the guideline on ‘call clients before high impact decisions’. The management of *Business Unit 1* set target KPIs within the organization to improve compliance w.r.t. first four rules and the performance of teams are subject of job evaluations. Several ideas and proposals also discussed to improve information systems to support em-

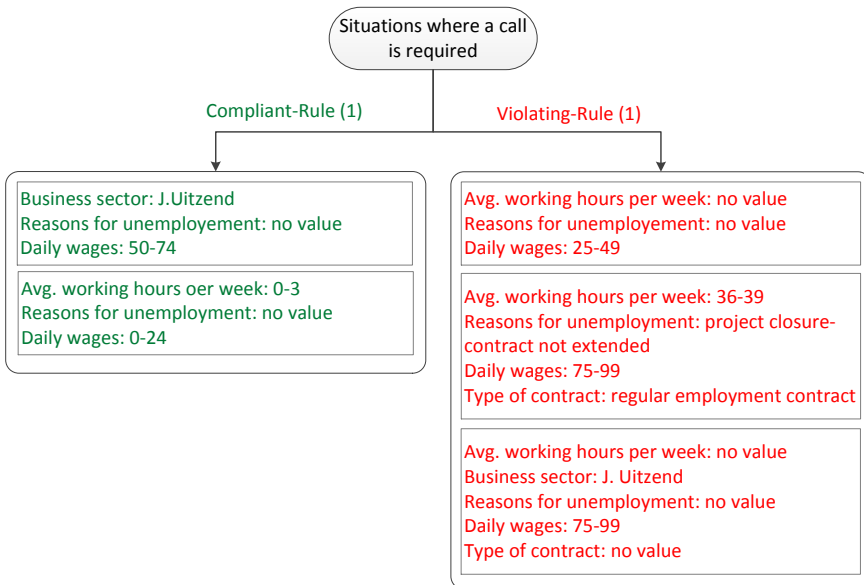


Figure 9.15: Violating patterns that distinguish compliant and violating activities w.r.t. compliance *Rule (1)* for *Business Unit 3*.

ployees better for assigning cases to appropriate teams and registration of the calls. In addition, initial ideas for a project to bring compliance checking in an online setting in UWV is discussed among higher management of UWV. UWV believes that such a system can warn the employees during the execution of cases whether some compliance rule is getting violated. Hence, employees would be able to take preventive or corrective measures earlier.

## 9.6 Follow-Up Analysis

After sharing our analysis results with domain experts in *Business Unit 1*, we repeated the analysis for the data of June and July 2015. We set up the case study as we did for the first study. We checked the same compliance rules for the three business units as in the first analysis. Clearly an impact of the analysis is shown in the results of the follow-up study. We saw a remarkable improvement in compliance related to *Rule (1), (2), (3), and (8)* in *Business Unit 1*. The compliance w.r.t. *Rule 5* showed a small improvement. The compliance related *Rule (4), (6), and (7)* decreased for interesting reasons. The improvements are not seen in other business units in the follow-up study. Either they did not improve or the compliance deteriorated. Next, we will discuss our observations from the follow-up study and will elaborate on the reasons caused the observations as such.

Business Units	# Cases	# Events
BU 1	27650	94979
BU 2	34210	116031
BU 3	33614	128289

Table 9.11: Number of cases and events in datasets of each business unit

We extracted the case study data sets of the follow-up study for all the three business units as we did for the first study. Table 9.11 shows the number of cases together with the number of events that we analyzed in the follow-up study for each business unit.

Table 9.12 summarizes the result of the follow-up analysis for the three business units. The *Total* column shows the total number of rule instances of each rule and *Violations* shows the frequency of violations per compliance rule. *Change* indicates the change in the frequency of violations, with (-) indicating

	Rule (1)	Rule (2)	Rule (3)	Rule (4)	Rule (5)	Rule (6)	Rule (7)	Rule (8)
BU 1	Violation	51%	59%	12%	27%	14%	13%	29%
	Total	3846	3846	3846	6850	6944	3064	3409
BU 2	Change	-7%	-7%	-11%	+17%	-1%	+5%	+27%
	Total	4640	4640	4640	7081	8888	4198	4416
BU 3	Change	+14%	+11%	0%	-6%	0%	+4%	+30%
	Violation	46%	54%	22%	17%	15%	12%	32%
BU 3	Total	7621	7621	7621	13667	11233	3509	3582
	Change	+3%	+2%	-4%	+2%	-2%	+5%	+28%

Table 9.12: Follow-up study, summary of violations per compliance rule in three business units.

decrease in violations and improvement in the compliance level and (+) showing an increase in violations.

### **Comparing results of the first analysis and the follow up analysis for compliance Rule (1).**

As can be seen in Table 9.12, the result of our analysis w.r.t. first compliance *Rule (1)* shows 7% improvement of compliance in *Business Unit 1* as the *A group* in our case study. We see an increase of 14% and 3% in violations respectively for *Business Unit 2* and *Business Unit 3*.

We expected an improvement for *Business Unit 1*, however, we did not expect an increase in violations of other units. That is, we expected ‘no change’ or ‘a slight change’ in other units. We discussed our observations with domain experts in *Business Unit 1*. They informed us about the enforcement of a new national law in the *unemployment process* that caused business units to focus on the introduction and enforcement of this new law in their operations. As a result, many employees in all the business units over the country are participating in several training and are busy with adjusting their daily operation with the implementation of the new law. Consequently, the rule “call before high impact decision requirement” got less priority during this period. The increase in violations of other business units may be related to being less attentive about this requirement during this period.

Note that, the new law is introduced in all the three business units including *Business Unit 1*. Despite the distraction caused by the introduction of a new law, we see a strong improvement in compliance level of the *Business Unit 1* due to the compliance analysis project.

### **Comparing results of the first analysis and follow up analysis for compliance Rule (2).**

Similar to compliance *Rule (1)*, we observe an improvement in compliance related to *Rule (2)* in *Business Unit 1* as the *A group* in our case study (See Table 9.12). In other business units, we see an increase in the percentage of violations; 11% and 2% respectively for *Business Unit 2* and *3*. Again we expected ‘no change’ or ‘a slight change’ in these business units. However, as we explained earlier, we speculate that the introduction of the new law in the organization has caused *Business Unit 2* and *3* to mobilize their resources and attention to implementation of the new law.

**Comparing results of the first analysis and follow-up analysis for compliance Rule (3).**

Table 9.12 also compares for each business unit the result of our analysis from the first study and the follow-up study w.r.t. compliance *Rule (3)*. As can be seen the result of our analysis shows an 11% improvement in *Business Unit 1* as the *A group* in our case study. We see no or a slight change in other units. This result is in line with the fact that we only shared the results of analysis with *A group*.

**Comparing results of the first study and follow up study for compliance Rule (4).**

As can be seen in Table 9.12, our analysis showed a 17% increase in violations related to *Rule (4)* in *Business Unit 1*. We did not expect this result for the *A group* in our case study, rather we expected an improvement. We found out that the increase in number of violations of *Rule (4)* is related to the definition of this rule and *Rule (3)*. *Rule (3)* states that “the calls before high-impact decisions must be registered as high impact.” This rule encourages the employees to register the calls as *high impact*. Whereas *Rule (4)* states: “regular calls must not be registered as high impact.” This rule encourages the employees not to register calls as *high impact* wherever it is not necessary. Note that, *Rule (4)* has less priority for *UWV* than *Rule (3)*. Consequently we observe that employees are overdoing *Rule (3)* and register calls as *high impact* even when it is not really necessary. This led to a dramatic increase in violations of *Rule (4)* in *Business Unit 1* compared to other units that were not aware of the case study.

**Comparing results of the first study and follow up study for compliance Rule (5).**

Table 9.12 also compares for each business unit the result of our analysis from the first study and the follow-up study w.r.t. compliance *Rule (5)*. As can be seen, we observe only a slight improvement (1%) in compliance related to *Business Unit 1* as the *A group* in our case study. We knew that compliance *Rules (4), (5), (6), (7), and (8)* had less priority in our case study compared to the first three rules. Therefore, except for the first three rules and *Rule (8)*, we did not discuss their results in detail during the dissemination of the results. Consequently, we did not detect a major improvement w.r.t. these rules.

Table 9.12 shows no change or slight improvement in the result of other business units.

**Comparing results of the first study and follow up study for compliance Rule (6).**

Similar to previous rules, the comparison of results from the first and follow-up study w.r.t. compliance *Rule (6)* is shown in Table 9.12 for each business unit. We observed almost 5% increase in the number of violations related to this rule in all business units. We discuss these results with domain experts in *Business Unit 1*. They speculated the increase in violations of *Rule (6)* also refers to the introduction of the new law in *UWV*. However no concrete conclusion was made on these results.

**Comparing results of the first study and follow up study for compliance Rule (7).**

Table 9.12 shows for all the three business units a drastic increase (between 27% to 30%) in the number of violations related to *Rule (7)*. We discussed these results with domain experts in *Business Unit 1*. They could not recall any specific change in the organization that may have caused this result. Note that, this compliance rule is enforced by the system. Hence, in general no violation is expected for this rule.

To find out what could have caused this drastic change, we analyzed the history of the compliance related to this rule. The result of this analysis from 1<sup>st</sup> January 2014 until 1<sup>st</sup> August 2015 is shown in Fig. 9.16. As can be seen, all

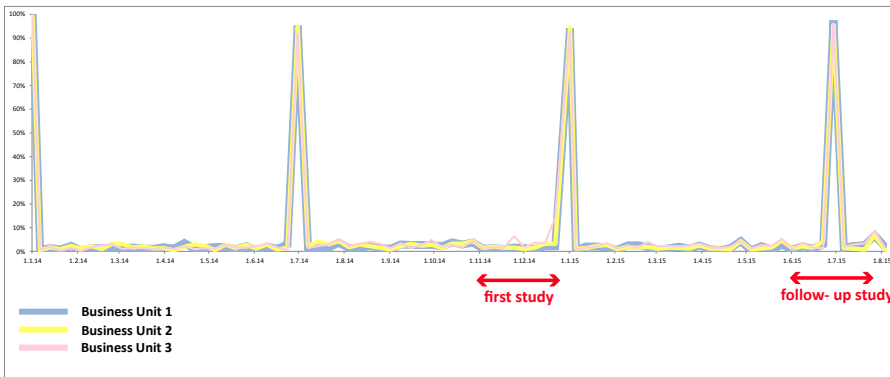


Figure 9.16: History of violations of *Rule (7)* in three business units.



the three business units follow the same pattern. The violations show a sudden increase from last week of December to first week of January and from last week of June to first week of July. This pattern is repeated for all the three units. We indicate in this figure, the periods where the data of the first study and the follow-up study is taken. As you can see, the data of the follow-up study falls in the period where there is a sudden increase of violations. Hence, the average of violations for *Rule (7)* is much higher in the follow-up study compared to the first study.

We discussed the detected pattern with domain experts in *Business Unit 1*. We could not find a concrete reason why these sudden changes are observed. However, we speculate that these changes are related to accounting reference periods, therefore some weekly benefits have ended in days other than a Monday.

#### **Comparing results of the first study and the follow up study for compliance Rule (8).**

As mentioned earlier, the results of the root-cause analysis of *Rule (8)* were discussed in detail with domain experts. As a result, we see 23% improvement in compliance related to *Rule (8)*.

## **9.7 Related Work**

For this chapter, we reviewed case studies that were conducted in three main domains: compliance checking and improvement, business process improvement, and analyzing and improving business processes using process mining techniques.

Most of the case studies we found in compliance, focus on introducing new regimens and incorporating controlling tasks and procedures within business processes to restrict the executions of non-compliant behavior.

The paper in [125] reports on a case study that aims at improving compliance of a warranty billing business process of a car dealing company using Six-Sigma DMAIC<sup>2</sup> approach. In this project available metrics to measure compliance of the process are reviewed and redefined to better evaluate relevant compliance constraints. Data were collected based on the new metrics and include stored data and interviews with respective people in charge. Based on the

---

<sup>2</sup>DMAIC stands for: Define, Measure, Analyze, Improve, and Control.

analyzed data, new controlling tasks were introduced in the process. Finally, the improvement of compliance based on new metrics is analyzed. The project included structural changes in the process together with several training to introduce changes and new ways of working to employees. Although a generic and structured approach (Six-Sigma) is followed in this project for compliance improvement, the root-cause analysis of the study is specific to the underlying warranty process and must be redesigned for a different process. In addition this analysis is not automated and can only be done by domain experts familiar with the process.

The paper in [72] reports a compliance improvement project in health care domain. The authors sought to improve compliance by constructing a centralized database of guidelines. The authors evaluate the project successful, however, keeping the database up to date remains a difficult task. The improvement of compliance using the centralized guideline is not quantified in this paper.

In [100], a case study is documented that evaluates compliance measures placed by authorities of Marine Protected Area management (MPA) in New South Wales. Measuring the “level of compliance” is considered as an important key performance indicators of success in MPA management. Authorities, placed several compliance improvement strategies. A study based on data analysis conducted to demonstrate the effectiveness of the compliance improvement strategies. The data analyzed include the number of enforcement actions (made up of written warning notices, penalty infringement notices and prosecutions) during a financial year for each marine park. For the purposes of this study, enforcement action types were pooled such that they included all offence types (e.g. fishing in a marine sanctuary, undersize fish, or illegal gear use) and actions taken (warning and infringement notices and prosecutions). Patrol effort was recorded in 0.5 hours slots per patrol type per day and was spatially recorded.

Five measures were defined for compliance, quantified based on the number of attacks and surveillance efforts required in an effective MPA management. To demonstrate the value of enforcement data in effective MPA management, more than 5000 enforcement actions from 2007 to 2013 from five New South Wales Marine parks were analyzed. The paper concludes that one of these compliance measures is not satisfactory. The analyzed data also revealed large differences in compliance rates among different age groups.

The authors provide the findings from compliance monitoring into compliance planning to improve MPA performance. Differences in the number of offences were compared among marine parks using a repeated-measures general linear model followed by Bonferroni-corrected LSD pair-wise post-hoc tests. An-

alyzing this data, the age demographic of offenders was understood. This information were used to adjust compliance strategies such as training and other forms of education with age-cohort behavioural patterns. The analysis resulted in changing of some of the compliance measures as well. The authors conclude that there was no consensus on the best way of measuring the success of a compliance program. However, their results show that evaluation of enforcement data can provide useful insights that can enhance compliance programs necessary for conservation of biodiversity in MPAs.

The paper in [61] discusses two strategies (namely control-based and commitment-based) to enforce traveling compliance in corporate companies. The study was done by comparing compliance results and costs and benefits of each strategy in several companies. The data for this study were obtained from several interviews and records of financial transactions. The study concludes that a combination of strategies scores better in all these criteria. This study does not follow a generic compliance checking approach and is not automated.

The business process improvement projects differ from each other either on their principles and techniques, or on the target area on which the improvements are focused.

A business process improvement framework is introduced in [35]. This framework extends business process management lifestyle on its evaluation phase. The integrated approach enables contagious improvement of business processes by analyzing executions of processes w.r.t. a set of KPI-like metrics. The metrics cover the devil's quadrangle including time, cost, quality and flexibility. The feasibility of this approach is showcased in a case study. The authors report on improvement points their approach can detect by analyzing the executions of business processes. The report does not contain the results after improvement.

The paper in [6] reports a case study to improve maintenance management procedures in an airport. This case study leveraged the ITIL-based <sup>3</sup> five stages improvement process. The project included identification of maintenance management sub processes, definition of KPIs to measure the process performance, collecting data, analyzing data and suggesting improvement actions. The paper unfortunately does not report on the improvement achieved after the case study.

A business process improvement technique (Tabular Application Development (TAD)) is introduced in [28]. The approach focuses on identification of business processes and modelling them using a UML-like activity diagram. The diagram leverages swim lanes to assign activities to specific organizational units.

---

<sup>3</sup>Information Technology Infrastructure Library

Some meta information about the activities such as estimated duration and cost are documented for activities in a separate table. Finally, an object model is created based on the modelled activities and is used for implementation. TAD framework aims at improving business processes by systematic identification, design and implementation of the model. However, the authors do not showcase whether using this approach led to an actual improvement in a real business scenario.

The authors of [71] propose a decision model that enables the selection and evaluation of a BPM road map. BPM road map is a portfolio of scheduled projects with different effects on business processes. This decision model helps companies to decide about execution and priority of business process improvement projects and projects that target developing of BPM capability (i.e., the skills to employ BPM methods and tools for business process change). In essence this model can be used to evaluate the impact of a compliance improvement project in a portfolio of business process improvement projects. However, the assumptions on the setting of the framework has simplified real world projects to great extent. For instance, it is assumed only one project to be implemented per period in a company which is far from reality especially in large organizations. Second, the authors assume limited set of parameters to define a project in their framework. These parameters are project quality, time, and sale's price of the project output, all estimated as a single value.

Several case studies are conducted to showcase the applicability of process mining techniques in analysing and understanding processes. One of the first case studies is reported in [139]. Here, the authors employ process mining techniques to understand control-flow, organization, and case perspective of provincial offices of the Dutch national public work department. In [124], authors focus on understanding the organization perspective of a process in a Dutch municipality.

To analyze process time, two cases studies are reported in [140]. The authors introduce a novel approach for predicting the completion time of running cases. In this approach the data is divided into a learning set and a test set and the focus is on the quality of predictions.

A case study paper in [51] studies the applicability of process mining techniques in real-life event log taken from telecom industry. The event log represents a highly flexible process. The authors compare the applicability of various techniques w.r.t. several criteria such as accuracy (i.e., the extent to which the induced model fits the behavior in the event log and can be generalized towards unseen behavior), comprehensibility (i.e., the extent to which an induced model is comprehensible to end-users), and justifiability (i.e., the extent to which an

induced model is aligned with the existing domain knowledge).

The authors of [128] developed a technique to discover interaction in an email-driven business process. Analysis of the discovered interactions provides insights about the interactions structure and resources involved in each interaction. They can detect active and passive participants, duration of interactions and message size of each interaction. These insights can be used to improve the underlying business process. The authors validate the feasibility of their technique in a case study. However, the paper does not report whether the insights obtained using this technique led to an actual improvement in the analyzed business process. In addition the technique is only beneficial for organizations where email is (one of) the most important communication media for human collaborations.

Unlike many case studies mentioned before, the authors of [110] analyze highly unstructured event data. This paper reports on a case study that was conducted on the testing process of manufacturing wafer scanners<sup>4</sup>. The authors employ various process mining techniques to analyze this process and suggest improvement measures.

The paper in [129] reports on a case study conducted in health care. The authors analyze and compare a process for which event logs are obtained from four hospitals. There exists some common grounds with our work in the sense that they analyze the differences and commonalities between different hospitals. The authors also discuss the challenges they faced during the analysis and suggest strategies to overcome some of the challenges. In [63], a case study in health care domain is showcased. The authors first list a set of questions usually posed by medical professionals. By means of a spectrum, different types of event data found in health care information systems are discussed to elaborate what type of event data allows for answering the questions posed earlier.

In [20] a six-step methodology for analyzing and understanding processes was introduced. This methodology includes log preparation, log inspection, control-flow analysis, performance analysis, role analysis, and transferring results. The methodology is employed in a case study in public sector. Various process mining techniques are used for each step in this study.

A *process mining success model* was introduced in [82]. The model consists of success factors in three main categories; project specific factors, information systems related factors, and process mining related factors. The model also includes success measures to evaluate a process mining project. The success

---

<sup>4</sup>ASML is the world-leading manufacturer and world largest supplier in the semiconductor industry.

measures include: model quality, process impact, and project efficiency. Note that model quality refers to the output of a process discovery project, whereas if we consider various process mining projects such as conformance checking or process performance analysis and enhancement, this measure will refer to evaluating the quality of outputs of such projects as well.

The authors of [127] discovered an invoice verification process of SAP in a company using Causal Net Miner. The authors are able to analyze the mined model w.r.t. the finding frequent patterns and paths in the process, the process paths that took the longest time versus those that took the least time, and process paths where essential controlling activities were missing for them. Their findings are provided to process owners for improving the analyzed process. However, the case study does not report any follow up study to check whether process improvements achieved based on these findings or not.

Among the case studies that analyzed process execution data of business processes, only few focus on improvement of process compliance. Most aim at improving business processes in general. The compliance checking technique using constraint-behavioral profile conformance metrics introduced in [146] is an exception. However, in this approach contextual data available in the process are not considered for root-cause analysis of detected violations. Understanding the deviations in this technique heavily depends on the knowledge about a concrete process. The authors tested the feasibility of the technique in a case study. The paper reports on compliance analysis of a process using proposed metrics. The analysis was not followed with a second study to measure the impact of the diagnostics provided by this approach in compliance improvement of the underlying business process.

Many of the compliance and business process improvement projects discussed in this section were designed and tailored towards a specific project. Our compliance analysis approach is generic and was not developed specific for UWV. Therefore, we can apply our compliance approach also to other organizations and processes. In addition, in some of the case studies discussed above (especially those used process mining techniques for improving processes), the analysis was focused on detecting bottlenecks and compliance problems. However, this analysis was never followed by any second analysis to report on actual improvements after recommending changes (this was done in our case study).

## 9.8 Conclusions and Lessons Learned

This chapter aimed to provide a deeper understanding of the benefits and challenges of a compliance improvement project in a highly rule-driven business process. We collected and specified eight compliance rules for *unemployment handling* business process. We designed a split test (A/B) to measure the impact of our approach in compliance improvement in three business units. We collected a data set for each business unit and analyzed these against the specified compliance rules. Within the bounds of this case study, our analysis results show using our compliance elicitation technique:

- we are able to specify different compliance rules precisely,
- we are able to check compliance rules of different types,
- we can provide detailed diagnostics about violations of each rule and an overview about overall compliance, and
- our root-cause analysis approach guides end users to hypothesize about the causes of different violations and helps them direct their compliance improvement initiatives.

We discussed the results of our first study with domain experts and management of *business Unit 1*. We repeated the same study with a new dataset for each business unit after some time. Our results show remarkable improvements in compliance results of the rules that were discussed in detail with domain experts and were of a high importance for the management.

The most challenging part of this study was the specification of the relevant compliance rules. Although we had good expertise for specifying them, compliance rules in many situations are subject to several exceptions and interpretations that make it difficult even for business owners to agree on a definition for a rule. Yet, the precise specification of these rules is very important for differentiating what is compliant or violating. Even more important, it helps domain experts to adhere to the rules.

The impact of the precise definition of compliance rules does not end here, but data requirements should be defined accordingly as well. This process can go through several iterations and it may lead to redefinition of rules as well. The choice of data attributes that need to be included in the datasets are very important. There are several data attributes that are mandatory to include in the dataset as they are used for checking the rules. When definition of rules

changes, new attributes may become relevant for the analysis. In addition, we need to select carefully other contextual data to include for root-cause analysis. Note that sometimes causes for violations lie outside the process being analyzed. Therefore, it is necessary to collect relevant data as much as possible and enrich the event log accordingly. The more data the richer the analysis will be. However, there is a trade-off. If the additional data attributes are not chosen carefully, the technique may find relations between data attributes and violations that do not provide much business insight.

In our analysis, apart from technical challenges of handling a large event log, too much data may lead to many results. Consequently we needed to spend enormous time and energy to analyze the results from business perspective to be able to extract important information from them. So far, we used specific data analysis techniques (association rule mining and classification) for our root-cause analysis approach. An option to tackle too many results could be using a group of techniques for the same type of analysis and concluding based on the combination of outcomes. For instance, for evaluating the strength of association rules, currently we are using *Conditional probability Increment Ratio* (CPIR) [152] metric. However, computing a combination of metrics such as *support*, *confidence* [12], and *lift* (interest) [22] together with CPIR would provide us the possibility to filter out association rules better and focus only on those that are scored high by all the metrics.

Another challenging part of a data analysis project is to provide a more tailored visualization of the results for domain experts. Interpretation of analysis results by domain experts adds valuable insight to any data analysis project and is very important. Therefore, it is crucial to visualize results of the analysis such that it is easier for domain experts to comprehend and discuss them. We used several ways to show our results and discuss them with domain experts. These initiatives include reporting to domain experts using tables, bar charts, different types of graphs, and textual format of results. However, we found out that organizing workshops and discussing the results together with domain experts has been more efficient than expected. Domain experts have problems reading graphs, tables, and charts on their own. Discussing the results in workshops opened up new perspectives and gave us and domain experts more insights about them. For instance, for discussing the important associations between violations and their contextual data, we found simpler and more compact visualization of analysis results. For example, the graphical representation we used in Fig. 9.7 together with complementary bar charts (see an example in Fig. 9.8) provided a good basis to discuss the results in a workshop.

We observed that organizations are dynamic ecosystems that are under con-



stant change. Hence, it is of utmost importance to choose right timing for a business process improvement project and target right people to minimize the impact of these changes on the result. Moreover we understood that changing a complex process (e.g. *handling unemployment benefits*), is time consuming and requires support of different levels in an organization.

# Chapter 10

## Conclusions and Future Work

This final chapter concludes the thesis. In Sect. 10.1, we summarize the main contribution of our compliance analysis approach. Section 10.2 discusses the limitations and open problems. Finally, we sketch our ideas for future research in Sect. 10.3.

### **10.1 Contributions of the Thesis**

In this thesis we introduced an approach for analyzing compliance of a business process. Our approach covers different phases of the compliance management life cycle introduced in Sect. 1.3. First of all, we advocated the idea of managing compliance separately from business process management. Hence, the changes in each of them (BPM or CM), will not impact the other and thereby increases the maintainability.

We have developed various techniques in our proposed compliance analysis solution. Figure 10.1 highlights the main contributions.

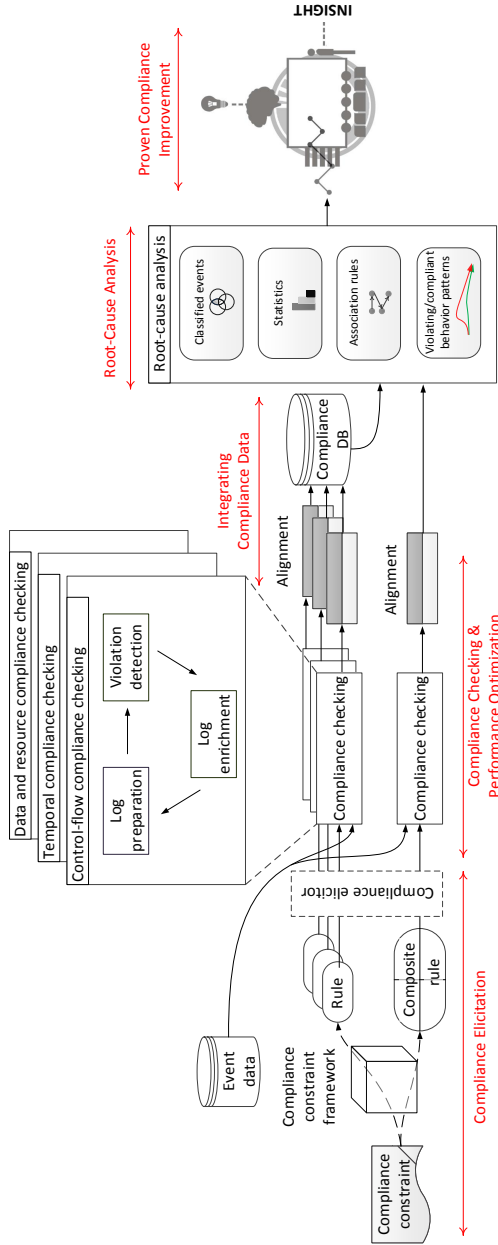


Figure 10.1: The main contribution of this work are highlighted in the thesis road map.

### **Compliance Elicitation and Specification**

The compliance constraint framework presented in Sect. 3.1 helps us to decompose complex compliance constraints to several individual rules each confining a perspective of a business process. We offered three collections of compliance rules for modelling control-flow, temporal, resource and data-aware compliance rules. We did an extensive literature study to include various rules from literature and practice in the rule repositories. We introduced a semi-automated technique for formalizing each control-flow and temporal compliance rule as a compliance pattern that can be directly used for compliance checking. This technique is supported by a tool and enables business users to formalize rules without being exposed to the technicalities of the formalization language used. Furthermore, we discussed the situations where a set of compliance rules should be modelled as a set of atomic compliance patterns or as a composite compliance model and explained in details the advantages and disadvantages. Either way, the impact on other phases of the compliance analysis solution is explained.

### **Compliance Checking**

Our approach includes dedicated techniques for checking compliance of control-flow, temporal, resource and data-aware compliance rules. Yet, all the techniques follow the same procedure including: log preparation, violation detection and log enrichment. However, the sequence of these steps and their repetition are customized in each checking technique to cover the requirements of checking rules from different process perspectives. Using these checking techniques, we can detect all violations from different process perspectives. The diagnostics we get include the exact location of violations, and the compensation value for a violation while the impact of the proposed value is considered on the compliance of the process globally. That is, the compensation value is proposed such that it minimizes the total number of violations and the amount of deviations globally. All the proposed checking techniques have been implemented and were evaluated on real life event logs.

### **Compliance Checking in Large Event Logs**

During the development of the checking techniques, their applicability on big data have been considered. The alignment technique which our compliance checking procedure approach is based on, requires solving optimization problems (e.g. an ILP problem). As the number of distinct activities and the average

trace length increases in large event logs, the complexity of these problems increases and thereby performance issues can arise. Our approach makes it possible to focus on a subset of activities and attributes that are confined by a compliance rule and abstract from all other information in the log. Typically, compliance patterns have only a few number of distinct activities and usually have a very simple structure. Furthermore, we shorten the traces during log preparation wherever the presence of events are not necessary for checking a rule. These optimization measures make our compliance analysis approach suitable to be applied in large datasets. For instance checking compliance of the dataset of one of the business units we used for case study (presented in Chap. 9) with over 100000 cases and over 35 attributes did not take longer than a few seconds.

### **Integrating Compliance Data in Compliance Database**

Checking compliance rules one by one will lead to several checking results that needs to be combined to get a complete picture of compliance of a business process. We introduce a framework to integrate compliance data from different sources in the ‘compliance database’. Compliance database gives us the flexibility to explore compliance data from different dimensions and on different abstraction levels. We can query compliance data based on even complex criteria. The result of these queries are subsets of events/move that can be used for further analysis including: (1) producing various statistics and reports, (2) root-cause analysis, and (3) generating enriched sublogs with diagnostics.

### **Providing Insights about Non-Compliance**

The last phase of our compliance analysis approach leverages various data analytic techniques to do root-cause analysis on detected violations. Association rule mining is applied to detect meaningful relations between violations and their context. These relations are shown to business users as list of problems in textual format. This list is ranked based on the importance of violations and the significance of the relation detected between a violation and its context. In addition, we apply classification techniques to detect patterns that differentiate violating and compliant patterns. This helps us understand the commonalities between violations (i.e., the conditions that are present when a certain violation occurs) and similarly the commonalities between non-violations (i.e., the conditions that are present when no violation occurs). This information has

predictive value and can help us to predict when a certain violation occurs. Furthermore, we provide descriptive statistics describing the severity of violations on different abstraction levels. The family of root-cause analysis techniques are implemented and evaluated on real life event logs.

### **Proved Impact on Compliance Improvement**

We have evaluated each component of our compliance analysis approach on real life event logs. We did this for each component separately, but also used the complete solution in a large case study. This experiment was designed to check the feasibility of our approach and applicability of its findings for improving compliance in a real business case. We did the experiment in collaboration with Dutch Employee Insurance Agency (UWV). The experiment was conducted in three business units of UWV using A/B testing. A set of compliance rules (of various types) was chosen and each rule was checked in all the three business units. One of the business units (A group) was informed about the results of our analysis and we did not distribute the results to the other two units (B group). The case study continued with a follow-up study using new datasets. We monitored and compared the changes in the compliance state of different business units. The results of the analysis showed that using our approach, we can provide detailed diagnostics and enable root-cause analysis to improve compliance. The compliance state w.r.t. some of the compliance rules showed up to 28% improvement in A group, the same effect was not observed in other business units. We received positive feedback from domain experts about the applicability of the approach and UWV is planning to incorporate our approach of compliance analysis in its daily operations.

## **10.2 Limitations**

In this thesis, not all the problems and open questions w.r.t. compliance management have been fully addressed. In this section, we discuss problems that when solved, would make our compliance analysis approach more complete.

**Compliance elicitation and formalization.** Although the compliance elicitation component of our approach enable business users to specify and formalize informal compliance constraints to great extent, formalizing informal text of compliance constraints still remains a challenge. The compliance rule collections (chapters 4, 5, and 6) include comprehensive lists of compliance rules,

however, in practice new rules or combination of rules may be used that cannot be foreseen. Yet, the repositories are extendible. The elicitation technique discussed in Chap. 4, is based on the collection of compliance rules in the control-flow compliance rules repository and may need to be extended accordingly. However, the maintainability of this technique is not straightforward. The configurable patterns that are used for specifying a concrete compliance rule can be extended easily, however, the questionnaire for selecting a specific configurable pattern and its configuration options should be updated for each new pattern and configuration option added to the repository. Furthermore, we did not validate this approach with end-users.

**Compliance checking.** The compliance checking techniques we employed in our approach aim at explaining the most probable scenario of non-compliance. For this, the techniques solve an optimization problem based on various costs that are assigned to violations. Tuning this cost function can lead to different solutions. Hence, sometimes domain knowledge is required to understand the best explanation for a violation. This problem has been discussed extensively in sections 4.1.1, 5.6, and 6.5.

**Root-cause analysis.** In our root-cause analysis approach, we use contextual data of violations, i.e., other data attributes and their values present at a violating event or its neighbouring events. We leverage association rule mining and classification techniques to detect possible meaningful relations between violations and their context. Such a root-cause analysis technique works best when we have a high number of rule instances (i.e., activations of the rules analyzed). In contrast, when applied on small dataset or a dataset with few violations, the insights may be of less value.

On the other hand, when the the number of attributes at events increases, we may end up with relations and patterns that although score high by the metrics employed, they do not indicate an important finding from a business point of view. Therefore, to make a precise conclusion, domain knowledge is required to select the relations that have higher business value. Checking all the detected relations and patterns (when many of them are detected) with domain experts can be a tedious task. For this challenge, we propose two solutions: (1) in case of association rule mining, we propose to use a combination of metrics for evaluating detected relations. Relations that score high on all metrics are more probable to indicate an important relation, (2) in case of classification techniques, we propose to filter the attributes that are less likely to be relevant for the analysis using domain knowledge. If no domain knowledge is available, we can leverage from feature selection techniques to choose groups of attributes

which are dependent. Details of these problems and possible solutions for them have been discussed in sections 9.5.1, 9.5.3, and 9.8.

Note that in many situations violations occur for reasons that are outside the context of the violations. For example a violation may occur due to an activity that was executed at the beginning of a process or it can be related to some external factors that are not even available in the recorded dataset. In such cases it is very difficult to detect causalities merely by analyzing the dataset and more domain knowledge is required to enrich the dataset with relevant contextual data.

## 10.3 Open Problems

This final section sketches ideas for further research in compliance analysis that extend beyond the scope of this thesis. All ideas relate to the ultimate goal of systematic understanding of non-compliance including detecting non-compliance and its root-cause analysis.

In this thesis, we covered analysis of compliance rules confining control-flow, temporal, resource and data perspectives of a business process. However, we detected two more dimensions in the compliance constraint framework (Sect. 3.1) that we did not discuss in this thesis. A compliance constraint may prescribe properties of a single case or of multiple cases (e.g. “20% of all the applications require a detailed check”). Such constraints are typical in Service Level Agreements (SLAs). We did not dedicate a specific part of the thesis to this dimension, however, compliance of such rules can be computed over the result of checking each case individually. The other dimension of compliance rules includes rules that prescribe properties of process design. An extensive body of literature is available about the modelling and analysis of such constraints. We briefly discussed these works in Sect. 1.3. However, compliance verification of process models during design time of business processes is outside the scope of this thesis.

The focus of this thesis has been on compliance auditing, i.e., post-mortem analysis of event logs. Enabling our approach of compliance analysis in a real time setting is an interesting extension of this work. For this, we propose a specialized engine for compliance management that communicates with an existing information system. A more concrete proposal about the compliance engine has been discussed in [98]. Figure 10.2 sketches the architecture of the proposed engine that is coupled with an existing information system. The relevant *compliance rules* for a process and their formalization are stored in a repository. The



*process definition* repository in Fig. 10.2 includes the information about the activities in a process and their order. *Business data definition* represents process data as a data model consisting of a set of entity types and their relationships between these entity types. The actual data of a process is given by a number of entities for each type. Each entity assigns a value to its attributes and is associated to other entities according to the relationships. Activities are associated with entity types defining which attributes the activities are allowed to read, write or update. The *organization definition* repository defines the association of activities to roles and assignment of agents to roles.

The repository of *runtime data* comprises all information on process executions. For instance, the events that have occurred, their order and duration, the values that were written by a particular event, the authorizations to access data granted to specific roles, or the role assignments given to specific agents. The *compliance checker* takes an instantiated compliance rule, in context of a specific process, and *runtime data* as input and detect violations or signal possible violations. The engine is allowed to control the information systems by a *risk interrupter*. The *risk interrupter* takes as input the discovered deviations and assesses based on its history of violations recorded in *compliance database* how severe the violation of a compliance rule would turn out in the future. In case of a severe risk, it can interrupt the process execution in the information system.

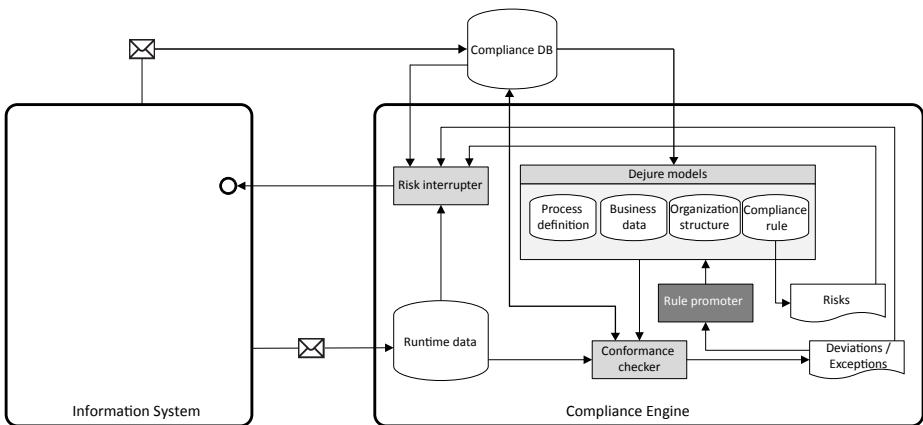


Figure 10.2: Proposed compliance engine for real time compliance management.

# Appendix A

## Repository of Control-Flow Compliance Rules

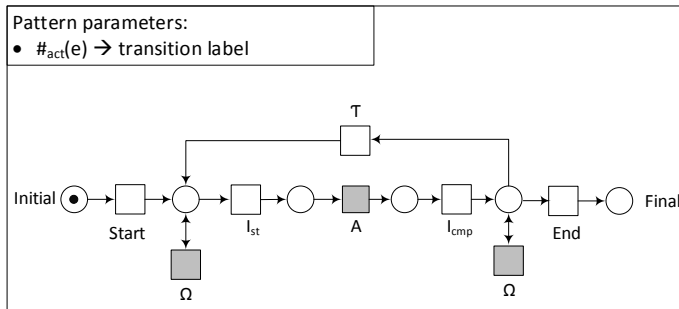
This appendix presents a repository of Petri net patterns modeling a collection of control-flow compliance rules. Table A.1 (also discussed in Chap. 2 and Chap. 4) gives the collection of control-flow compliance rules. These rules are distributed over ten categories. Each category includes several compliance rules. Categories are formed based on the type of constraints they include.

All the Petri net patterns discussed in this repository are created in a systematic manner that will be explained throughout this appendix. The basic principles for designing the control-flow compliance rules and modelling elements and constructs used in the patterns are discussed in Sect. 4.1.5. In all the patterns discussed in this appendix, the attribute *act* of run events is mapped to the *transition label* of the respective pattern. (See Sect. 2.3.3.

Compliance patterns are generic and can be instantiated in context of any compliance requirement and business process. For each pattern, we specify a set of instantiation parameters. A user can instantiate a pattern in context of a specific event log by setting these parameters during the alignment.

Category (Rules)	Description
Existence (2)	Limits occurrence or absence of an activity. [142], [14, 38], [55], [117], [118]
Bounded Existence (6)	Limits the number of times an activity must or must not occur. [38], [42]
Dependent Existence (6)	Limits the presence or absence of an activity with respect to existence or absence of another activity. [42]
Bounded Sequence (3)	Limits the number of times a sequence of activities must or must not occur. [38], [42]
Parallel (2)	Limits the occurrence of a specific set of activities in parallel. [117]
Precedence (10)	Limits the occurrence of an activity in precedence over another activity. [42], [38], [117], [118], [14], [49], [55]
Chain Precedence (4)	Limits the occurrence of a sequence of activities in precedence over another sequence of activities. [42], [38], [55]
Response (10)	Limits the occurrence of an activity in response to another activity. [117], [42], [55], [38], [119], [14], [49]
Chain Response (4)	Limits the occurrence of a sequence of activities in response to another sequence of activities. [42]
Between (7)	Limits the occurrence of an activity within (between) a sequence of activities. [38]

Table A.1: Categorization of the 54 control-flow compliance rules.

Figure A.1: *Existence. Activity Universality* compliance rule

## A.1 Existence Category

This category contains compliance rules which limit the occurrence or absence of an activity within a chosen scope<sup>1</sup>.

### Existence. Activity Universality

*Description:* Activity  $A$  must occur within a chosen scope. The compliance rule is violated if event  $A$  does not occur within the specified scope (e.g. a process instance). An instance of this compliance rule includes one time occurrence of  $A$ , i.e., if it occurs once, this compliance rule is satisfied. Figure A.1 shows the Petri net pattern that formalizes this rule.

After the pattern started, any event may occur. The *rule instance* is triggered as soon as  $A$  is executed. With the execution of  $A$ , the corresponding compliance rule of this pattern is satisfied and the *rule instance* is completed. After completion of the *rule instance*, any event may occur. In this situation the pattern may terminate. The transition *End* models that the end of the trace has been reached, i.e., it occurs *after* all events of the trace occurred.

*Pattern instantiation parameters:*

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$

<sup>1</sup>The scope can refer to a process instance (one specific case), a group of process instances, or a time line.

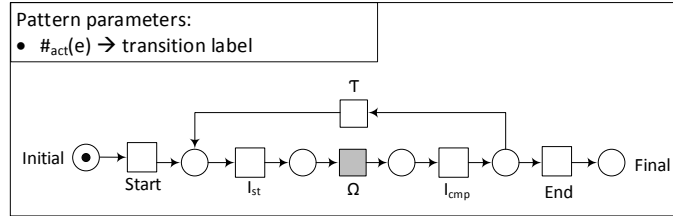


Figure A.2: Existence. Activity Absence compliance rule

### Existence. Activity Absence

*Description:* Activity  $A$  must not occur within a chosen scope. The rule is violated if activity  $A$  occurs within the specified scope. An instance of this compliance rule includes occurrence of any activity except  $A$ . That is, as long as  $A$  does not occur, this compliance rule is satisfied but if  $A$  occurs this rule is violated. Fig. A.2 shows the Petri net pattern that formalizes this rule.

The pattern specifies that any event but  $A$  may occur. Therefore, by construction in every instance of this pattern only the  $\Omega$ -labeled transition is enabled. If an  $A$  occurs a deviation is captured. The transition  $End$  models that the end of the trace has been reached, i.e., it occurs *after* all events of the trace occurred.

*Pattern instantiation parameters:*

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$

## A.2 Dependent Existence Category

This category of compliance rules limits the presence or absence of an activity with respect to existence or absence of another activity.

### Dependent Existence. Exclusive

*Description:* Presence of activity  $A$  mandates the absence of activity  $B$  within a chosen scope. This rule is violated if within the specified scope, both activities  $A$  and  $B$  would be present. An instance of this compliance rule includes all occurrences of activity  $A$  or all occurrences of activity  $B$ . The Petri net pattern illustrated in Fig. A.3 formalizes this rule.

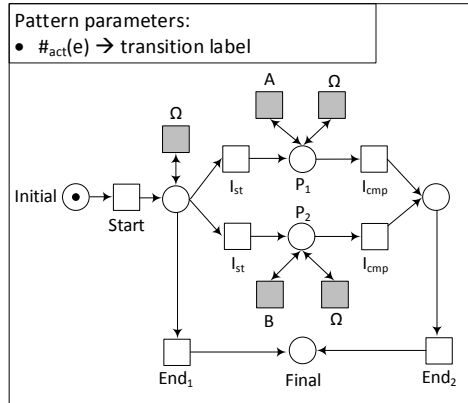


Figure A.3: *Dependent Existence. Exclusive compliance rule*

After the pattern started, any activity may occur. The *rule instance* is activated as soon as the first *A* or *B* occurs. If *A* occurs, place  $P_1$  is marked. At this marking, *B* is not enabled anymore, thereby ensuring that *A* and *B* cannot be present together. Symmetrically when the first *B* occurs, place  $P_2$  is marked. At this marking, *A* is not enabled anymore, thereby ensuring that *A* and *B* cannot be present together.

The pattern may terminate at any point in time by firing one of the transitions *End*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Dependent Existence. Mutual Exclusive**

*Description:* Within a chosen scope, either activity *A* or activity *B* must exist but not none of them or both. This rule is violated if both events *A* and *B* occur together or be absent together within the specified scope. An instance of this compliance rule includes all occurrences of activity *A* or all occurrences of activity *B*. The Petri net pattern illustrated in Fig. A.4 formalizes this rule.

The behavior of this pattern is similar to the pattern described in Fig. A.3; with the difference that in the pattern illustrated in Fig. A.4 absence of both

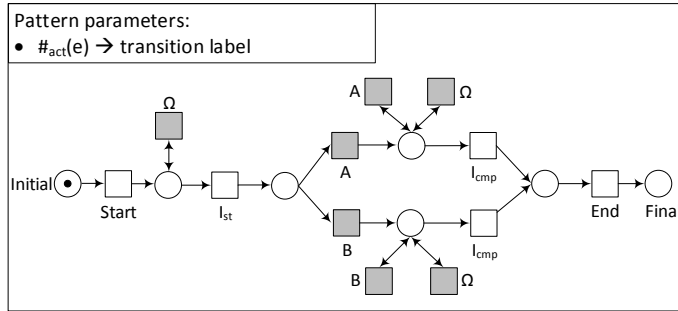


Figure A.4: *Dependent Existence. Mutual Exclusive* compliance rule

events  $A$  and  $B$  is a violation. Therefore, the pattern enforces that one of the events  $A$  or  $B$  must occur. The pattern cannot terminate unless the *rule instance* is completed, i.e., only after the occurrence of one of the events  $A$  or  $B$ . After the condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Dependent Existence. Prerequisite

*Description:* Absence of activity  $A$  mandates that activity  $B$  is also absent within a chosen scope. This rule is violated if within the specified scope, activity  $B$  occurs without any occurrence of activity  $A$ . This compliance category consists of one compliance rule. An instance of this compliance rule includes occurrences of both activities  $A$  and  $B$ . The Petri net pattern illustrated in Fig. A.5 formalizes this rule.

The occurrence of any activity triggers the *rule instance*. If  $A$  occurs, it may be followed by  $B$  or not. In both cases, the behavior is compliant, i.e., the presence of  $A$  does not create an obligation. The *rule instance* can complete and the pattern may terminate in this situation if no event is to be executed. However, as soon as  $B$  occurs the structure of the pattern must ensure that  $A$  also occurs at least once. Therefore, the occurrence of  $B$  requires occurrence of  $A$ . Note that, the sequence of occurrences of  $A$  and  $B$  does not matter but if  $B$  occurs,  $A$  must

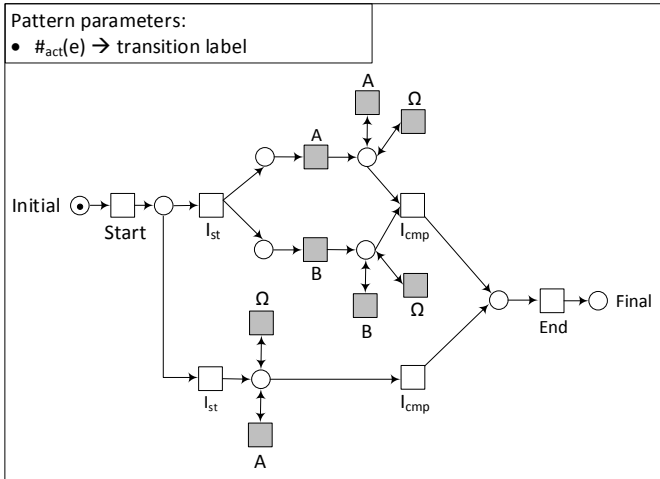


Figure A.5: *Dependent Existence. Prerequisite compliance rule*

have occurred at least once before it or it must eventually occur after it. The next occurrences of *B* (even if they are not followed by *A*) are still compliant as *A* has already occurred once and the rule is satisfied. Please note that absence of both activities is allowed based on the rule. When the condition of the rule is satisfied, the *rule instance* is completed and the pattern may terminate by firing the transition *End*.

*Pattern instantiation parameters:*

- *A, B*
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Dependent Existence. Inclusive**

*Description:* Presence of activity *A* mandates that activity *B* is also present within a chosen scope. This rule is violated if within the specified scope, activity *A* occurs without any occurrence of activity *B*. This compliance category consists of one compliance rule. An instance of this compliance rule includes all occurrences of activities *A* and *B*. The Petri net pattern illustrated in Fig. A.6 formalizes this rule.





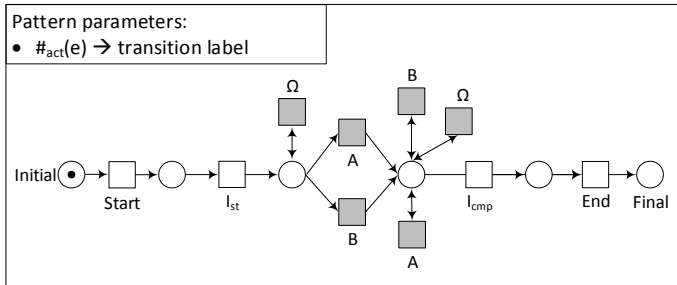


Figure A.7: *Dependent Existence. Substitute* compliance rule

**Dependent Existence. Substitute**

*Description:* Activity *B* substitutes the absence of activity *A* within a chosen scope. This rule is basically the logical *OR* between occurrences of two activities *A* and *B*. This rule is violated if within the specified scope, none of the activities *A* or *B* occurs (i.e., both be absent). This compliance category consists of one compliance rule. An instance of this compliance rule includes all occurrences of activities *A* and *B*. The Petri net pattern illustrated in Fig. A.7 formalizes this rule.

An occurrence of any activity triggers the *rule instance*. The *rule instance* cannot complete unless at least one of the activities *A* or *B* occur. The occurrence of *A* does not create an obligation for the occurrence or non-occurrence of *B*, however its absence obliges the occurrence of *B*. When the condition of the rule is satisfied, the pattern may terminate by firing the transition *End*. *Pattern instantiation parameters:*

- *A, B*
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Dependent Existence. Co-requisite**

*Description:* Within a chosen scope, either activities *A* and *B* should exist together or be absent together. This rule is violated if within the specified scope, only one of the activities *A* or *B* occurs. This compliance category consists of one compliance rule. An instance of this compliance rule includes all occurrences of

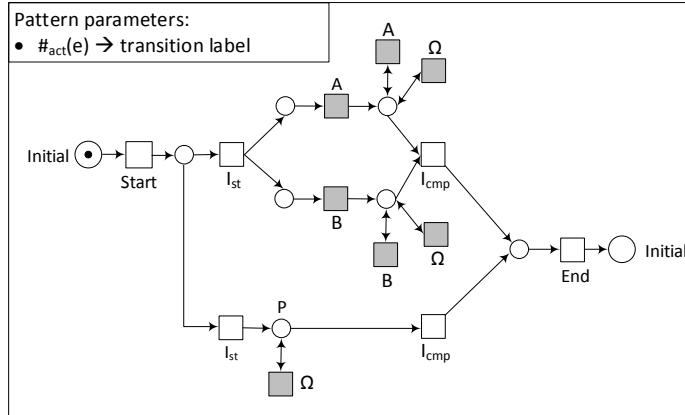


Figure A.8: *Dependent Existence*. Co-requisite compliance rule

$A$  and  $B$  if they occur. The Petri-net pattern illustrated in Fig. A.8 formalizes this rule.

The pattern described in Fig. A.8 is similar to the pattern described in Fig. A.5, with the difference in adjacent transitions to the place  $P$ .

An occurrence of any activity triggers the *rule instance*. As soon as activity  $A$  occurs, the structure of the pattern must ensure that  $B$  also occurs. The next occurrences of  $A$  (even if they are not followed by  $B$ ) are still compliant as  $B$  has occurred once and the rule is satisfied. Symmetrically if activity  $B$  occurs, the structure of the pattern must ensure that  $A$  also occurs. The next occurrences of  $B$  (even if they are not followed by  $A$ ) are still compliant as  $A$  has occurred once and the rule is satisfied. Please note that absence of both activities  $A$  and  $B$  is also compliant. When the condition of the rule is satisfied, the *rule instance* completes and the pattern may terminate by firing the transition  $End$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

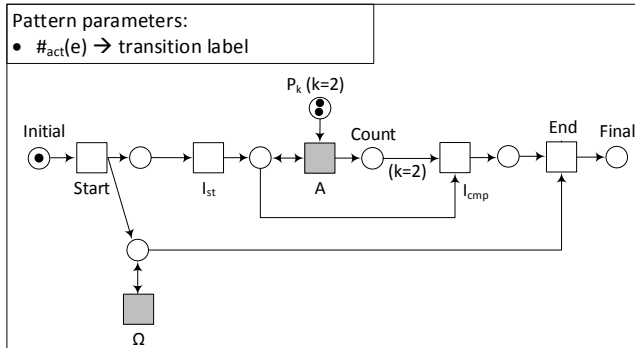


Figure A.9: *Bounded Existence of an Activity. Exactly k Times* compliance rule

### A.3 Bounded Existence Category

This category includes compliance rules that limit the number of times an activity must or must not occur.

#### Bounded Existence of an Activity. Exactly $k$ Times

*Description:* Activity  $A$  must occur exactly  $k$  times within a chosen scope. The rule is violated if  $A$  occurs less than or more than  $k$  times within the specified scope. An instance of this compliance rule includes  $k$  occurrences of activity  $A$ . That is, as soon as  $A$  occurs  $k$  times, this rule is satisfied. Figure A.9 shows the Petri net pattern that formalizes this rule for the case  $k=2$ .

After the pattern started any event may occur. The first occurrence of  $A$  triggers the *rule instance*. The pre-place  $P_k$  of  $A$  is initially marked with  $k$  tokens. The  $k$  tokens in place  $P_k$  assure that activity  $A$  can occur at most  $k$  times, as each occurrence of  $A$  decrements the number of tokens in  $P_k$  and increments the number of tokens in place *Count*. Place *Count* counts the occurrences of  $A$ . After  $k$  occurrences of  $A$ ,  $P_k$  is empty and *Count* contains  $k$  tokens. In this situation transition  $A$  is not enabled anymore. The compliance rule is satisfied and the *rule instance* is completed. The pattern can terminate only if the *rule instance* is completed implying that the condition of the rule is satisfied. The transition *End* models that the end of the trace has been reached. Please note that  $\Omega$ -labeled transition is enabled throughout the whole pattern and may occur at any point in time.

Pattern instantiation parameters:

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

### Bounded Existence of an Activity. At Least $k$ Times

*Description:* Activity  $A$  must occur at least  $k$  times within a chosen scope. If  $A$  occurs less than  $k$  times within the specified scope, the rule is violated. An instance of this compliance rule includes  $k$  occurrences of activity  $A$ . That is, as soon as  $A$  occurs  $k$  times, this rule is satisfied. In addition, the rule also allows for more occurrences of  $A$ . Fig. A.10 shows the Petri net pattern that formalizes this rule for the case  $k = 2$ .

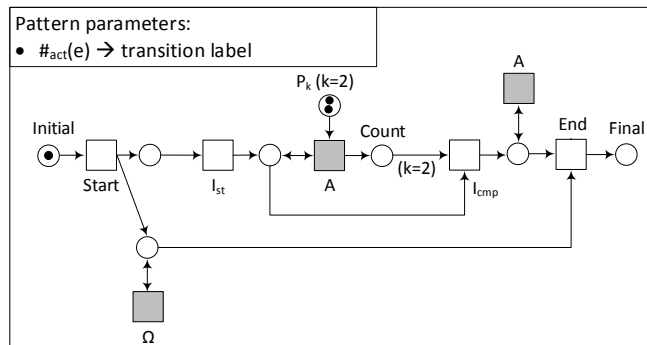


Figure A.10: Bounded Existence of an Activity. At Least  $k$  Times compliance rule

The basic structure of the pattern in Fig. A.10 is similar to the pattern described in Fig. A.9. The first occurrence of  $A$  activates the *rule instance*. The  $k$  tokens in place  $P_k$  limits the occurrence of the very left  $A$ -labeled transition to  $k$  times. After  $k$  occurrences of  $A$  the condition of the rule is satisfied, hence the *rule instance* is completed. The rule specifies that  $A$  must occur at least  $k$  times; therefore after  $k$  occurrences of  $A$ , further occurrences of  $A$  are possible. The  $\Omega$ -labeled transition is always enabled. The transition *End* models that the end of the trace has been reached.

Pattern instantiation parameters:



- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

**Bounded Existence of an Activity. Exactly  $k$  Times in a Row**

*Description:* Activity  $A$  must occur exactly  $k$  times in a row (directly one after the other) within a chosen scope. The rule is violated if the sequence  $\underbrace{\langle A, \dots, A \rangle}_k$

does not occur within the specified scope. An instance of this compliance rule includes  $k$  occurrences of activity  $A$  in a row. That is, if  $A$  occurs exactly  $k$  times without any other activities occurring between occurrences of  $A$ , this rule is satisfied. Fig. A.12 shows the Petri net pattern that formalizes this rule for the case  $k=2$ .

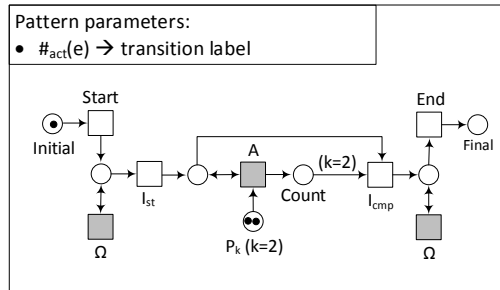


Figure A.12: *Bounded Existence of an Activity. Exactly  $k$  Times in a Row* compliance rule

After the pattern started, any event may occur. The first occurrence of  $A$  activates the *rule instance*. After the start of the *rule instance*, no  $\Omega$ -labeled transition is enabled anymore until  $A$  occurs  $k$  times in a row. The  $k$  tokens in pre-place  $P_k$  of  $A$  limit the number of occurrences of  $A$  to  $k$  times (similar to the structure described in Fig. A.9). The place *Count* counts the number of occurrences of  $A$ . The *rule instance* completes only if there are  $k$  tokens in place *Count*, implying that the condition of the rule is satisfied. In this situation the *rule instance* completes and any non- $A$  event may occur (captured by the  $\Omega$ -labeled transition) or the pattern may terminate by firing the transition *End*.  
*Pattern instantiation parameters:*

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

**Bounded Existence of an Activity. At Least  $k$  Times in a Row**

*Description:* Activity  $A$  must occur at least  $k$  times in a row (directly one after the other) within a chosen scope. This rule is violated if  $\underbrace{\langle A, \dots, A \rangle}_k$  does not

occur within the chosen scope. An instance of this compliance rule includes  $k$  occurrences of activity  $A$  in a row. That is, as soon as  $A$  occurs exactly  $k$  times without any other activities occurring between occurrences of  $A$ , this rule is satisfied. Fig. A.13 shows the Petri net pattern that formalizes this rule for the case  $k=2$ .

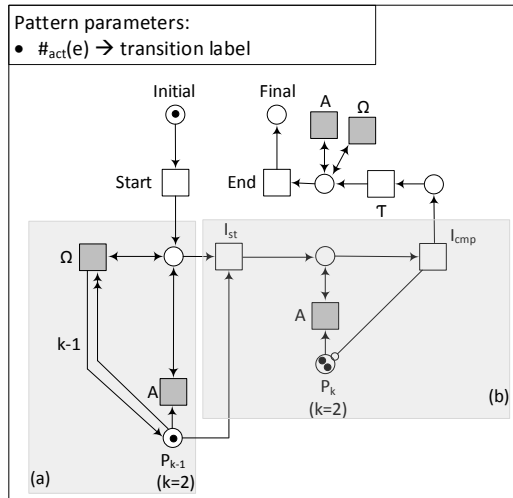


Figure A.13: *Bounded Existence of an Activity. At Least  $k$  Times in a Row* compliance rule

After the pattern started, any activity may occur including  $A$ . As soon as the first activity  $A$  occurs, we distinguish two scenarios: *Scenario 1* is that  $(k)$   $A$ 's occur in a row; *Scenario 2* is that an  $\Omega$  occurs after less than  $(k)$   $A$ 's in a row.



In case of *Scenario 1*, as soon as the first  $A$  occurs the *rule instance* is activated (please see the shadowed subnet labeled (b)) and no  $\Omega$ -labeled transition is enabled any more until the *rule instance* completes. The  $k$  tokens in the place  $P_k$  limits the number of occurrences of  $A$  to  $k$  in the *rule instance*. The *rule instance* completes only if the subsequence  $\underbrace{\langle A, \dots, A \rangle}_k$  occurs; implying that the

condition of the rule is satisfied. The *inhibitor* arcs connecting the place  $P_k$  to the transition  $I_{cmp}$  assures that the  $I_{cmp}$  can fire only if the place  $P_k$  is empty. The compliance rule specifies that more than  $k$  occurrences of  $A$  in a row is allowed; hence after the compliance rule is satisfied any arbitrary occurrences of  $A$  or  $\Omega$  is possible. In this situation the pattern may terminate by firing the transition *End*.

The left part of the pattern (please see the shadowed subnet labeled (a)) models the *Scenario 2*. That is,  $A$  may occur  $k-1$  times in a row. The  $k-1$  tokens in the place  $P_{k-1}$  limits the occurrences of  $A$  directly one after the other to  $k-1$  in the subnet (a). Please note that  $A$  may occur arbitrary number of times in the subnet (a) as long as, the sequence of  $A$ 's is interrupted by an  $\Omega$ -labeled activity before the sequence  $\underbrace{\langle A, \dots, A \rangle}_k$  occurs. The pattern may only terminate

if the condition of the rule is satisfied implying that the *rule instance* must be executed.

*Pattern instantiation parameters:*

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

### **Bounded Existence of an Activity. At most $k$ Times in a Row**

*Description:* Activity  $A$  must not occur more than  $k$  times in a row (directly one after the other) within a chosen scope. This rule is violated if  $\underbrace{\langle A, \dots, A \rangle}_{>k}$  occurs

within the specified scope. An instance of this compliance rule includes all occurrences of activity  $A$ . Fig. A.14 shows the Petri net pattern that formalizes this rule for the case  $k=2$ .

After the pattern started, the place  $P$  is marked. At this marking, any activity may occur. Based on this rule, less than  $k$  occurrences of  $A$  or exactly  $k$  occurrences of  $A$  in a row are considered as compliant behavior. Therefore, at the

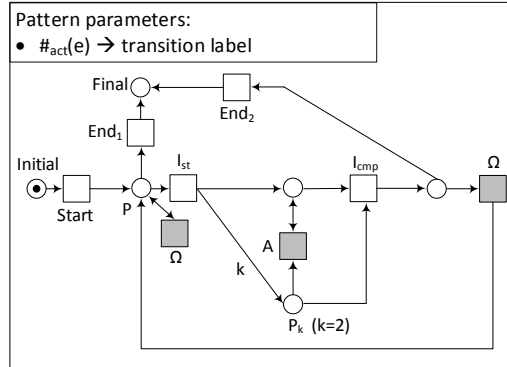


Figure A.14: Bounded Existence of an Activity. At Most k Times in a Row compliance rule

marking where the place  $P$  is marked (no  $A$  has occurred yet), the pattern may terminate by firing the transition  $End_1$ .

The rule instance is triggered as soon as the first  $A$  occurs. Firing the transition  $I_{st}$  produces  $k$  tokens in the place  $P_k$ ; implying that  $A$  may occur at most  $k$  times in a row in every instance of the pattern. Please note that the compliance rule allows for less than  $k$  occurrences of  $A$  in a row, hence the rule instance can complete even if there are tokens left in the place  $P_k$  (less than  $k$  or zero tokens). The reset arc connecting the place  $P_k$  to the transition  $I_{cmp}$  removes all the remaining tokens in the place  $P_k$ . After the rule instance completes, the pattern may terminate (by firing the transition  $End_2$ ) because the condition of the rule is satisfied so far. However if an  $\Omega$ -labeled activity occurs, the pattern returns to the marking where the place  $P$  is marked. At this marking an  $\Omega$ -labeled activity may occur, another instance of the pattern may be triggered or the pattern may terminate.

Pattern instantiation parameters:

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

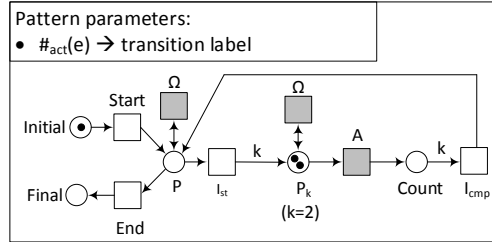


Figure A.15: *Bounded Existence of an Activity. Bursts of k Occurrences* compliance rule

### Bounded Existence of an Activity. Bursts of $k$ Occurrences

*Description:* Activity  $A$  must occur in bursts of  $k$  occurrences within a chosen scope. This rule is violated if  $A$  does not occur in bursts of  $k$  occurrences within the specified scope. An instance of this compliance rule includes  $k$  occurrences of  $A$ . Fig. A.15 shows the Petri net pattern that formalizes this rule.

After the pattern started, any activity may occur. The *rule instance* is triggered as soon as the first  $A$  occurs. Start of the *rule instance* produces  $k$  tokens in place  $P_k$ , which restricts the number of occurrences of  $A$  in every *rule instance* to at most  $k$  times. Occurrence of each  $A$  decrements the number of tokens in  $P_k$  and increases the number of tokens in *Count*. The *rule instance* may complete only if there are  $k$  tokens in place *Count*. When the *rule instance* is completed, the pattern returns to the marking where there is a token in place  $P$ .

The pattern structure in Fig. A.15 is modeled such that it models a cyclic behavior of occurrences of  $A$ . That is,  $A$  may occur any arbitrary number as long as it occurs in bursts of  $k$  times. Therefore, after completion of every instance of the pattern, the next *rule instance* may start, an  $\Omega$ -labeled activity may occur or the pattern may terminate (by firing the transition *End*) because the condition of the rule is satisfied. Please note that every occurrence of  $A$  is captured in a *rule instance*, i.e.,  $A$  may not occur outside of the *rule instance*.

*Pattern instantiation parameters:*

- $A$
- $\Omega = \Sigma_L \setminus \{A\}$
- $k$

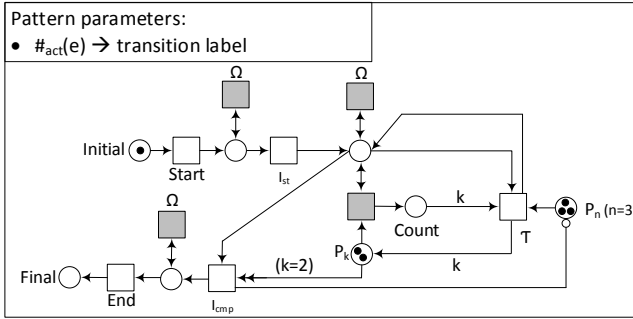


Figure A.16: Bounded Existence of an Activity. n Bursts of k Occurrences compliance rule

**Bounded Existence of an Activity. n Bursts of k Occurrences**

*Description:* Activity A must occur in n bursts of k occurrences within a chosen scope. This rule is violated if A does not occur in n bursts of k occurrences within the specified scope. An instance of this compliance rule includes all occurrences of A. Figure A.16 shows the pattern of this rule for k=2 and n=3.

After the pattern started, any activity may occur. The rule instance is triggered as soon as first A occurs. The k tokens in the place P\_k assures that in each burst, A occurs exactly k times. Every occurrence of A decrements a token from P\_k and increments the number of tokens in place Count. Firing the transition τ represents the completion of one burst. The completion of each burst empties the place Count; implying that A occurred exactly k times and returns the pattern to the marking where the next burst can be executed by producing k tokens in place P\_k. In addition, the completion of each burst decrements a token from place P\_n. The n number of tokens in place P\_n limits the execution of bursts to n number. The rule instance may complete only if the place P\_n is empty; implying that n bursts were executed. Please note that Ω-labeled transition is always enabled and it may occur between occurrences of A's. After the completion of the rule instance, the pattern may terminate by firing the transition End.

Pattern instantiation parameters:

- A
- $\Omega = \Sigma_L \setminus \{A\}$
- k, n

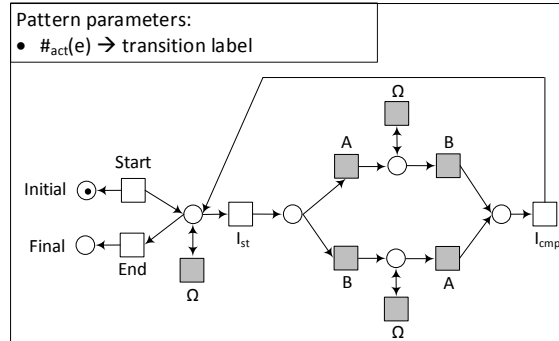


Figure A.17: *Bounded Sequence of Activities. One to One Coexistence* compliance rule

## A.4 Bounded Sequence Category

This category includes compliance rules that limit the number of times a sequence of activities must or must not occur within a chosen scope.

### Bounded Sequence of Activities. One to One Coexistence

*Description:* For every activity  $A$ , there should exist one activity  $B$  and for every activity  $B$  there should exist one activity  $A$ . If  $A$  and  $B$  do not occur in form of a pair, the rule is violated. An instance of this compliance rule includes one occurrence of the pair  $(A,B)$  in any order. Figure A.17 shows the Petri net pattern that formalizes this rule.

After the pattern started, any activity may occur. The occurrence of first  $A$  or  $B$  triggers the *rule instance*. If the pattern starts with an  $A$ ,  $B$  must follow it eventually. Symmetrically occurrence of the first  $B$  requires that  $A$  follows it eventually, otherwise the *rule instance* cannot complete. At this situation the pattern may terminate by firing the transition *End*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

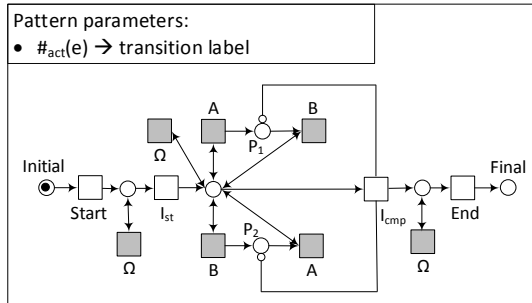


Figure A.18: *Bounded Sequence of Activities. Coexistence compliance rule*

### Bounded Sequence of Activities. Coexistence

*Description:* For any given number of activities  $A$ , there should exist the same number of activities  $B$  within a chosen scope. This rule is violated if the number of occurrences of  $A$  is not equal to the number of occurrences of  $B$ . An instance of this compliance rule includes all occurrences of activities  $A$  and  $B$ . Fig. A.18 shows the Petri net pattern that formalizes this rule.

After the pattern started, any activity may occur. The first occurrence of  $A$  or  $B$  triggers the *rule instance*. In the upper subnet illustrated in the *rule instance*, place  $P_1$  counts occurrences of  $A$ . Each occurrence of activity  $A$  increments the number of tokens in  $P_1$ , and each occurrence of activity  $B$  decrements the number of tokens in  $P_1$ . This construction ensures that for each occurrence of activity  $B$ ,  $A$  must have occurred earlier. Therefore, place  $P_1$  becomes empty only if  $B$  occurs as many times as  $A$  has occurred.

Symmetrically in the lower subnet of the *rule instance*, place  $P_2$  counts occurrences of  $B$ . Each occurrence of  $B$  increments the number of tokens in  $P_2$ , and each occurrence of  $A$  decrements the number of tokens in  $P_2$ . This construction ensures that for each occurrence of activity  $A$ ,  $B$  must have occurred earlier. Therefore, place  $P_2$  becomes empty only if  $A$  occurs as many times as  $B$  has occurred.

The *rule instance* can complete only if the places  $P_1$  and  $P_2$  are empty, implying that  $A$  and  $B$  occurred the same number. The *inhibitor arcs* connecting  $P_1$  and  $P_2$  to the transition  $I_{cmp}$  ensure that the *rule instance* can complete only if  $P_1$  and  $P_2$  are both empty. After the completion of the *rule instance*, the pattern may terminate by firing the transition  $End$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Bounded Sequence of Activities. Exactly $k$ Times

*Description:* The sequence of activities  $\langle A_1, \dots, A_n \rangle$  must occur exactly  $k$  times within a chosen scope. The compliance rule is violated if  $\langle A_1, \dots, A_n \rangle$  does not occur in the specified sequence or not  $k$  times. An instance of this compliance rule includes  $k$  occurrences of the sequence  $\langle A_1, \dots, A_n \rangle$ . Figure A.17 shows the Petri net pattern that formalizes this rule for the case  $k=2$ .

After the pattern started, any event may occur. The *rule instance* starts as soon as the first activity in the sequence  $\langle A_1, \dots, A_n \rangle$  occurs. An occurrence of  $A_1$  decrements a token from the pre-place  $P_k$  of  $A_1$ . After the first occurrence of  $A_1$ , any event may occur but eventually  $A_2$  must occur.  $A_1$  may be executed only after the first sequence of  $\langle A_1, \dots, A_n \rangle$  completes. The rule instance can complete only if the places named  $P_k$  are empty, implying that every element of the sequence  $\langle A_1, \dots, A_n \rangle$  occurred exactly  $k$  times. The *inhibitor arcs* connecting the places named  $P_k$  to the transition  $I_{cmp}$  assure that  $I_{cmp}$  may only fire if the places named  $P_k$  are empty. After the *rule instance* completed any activity may occur or the pattern may terminate by firing the transition *End*.

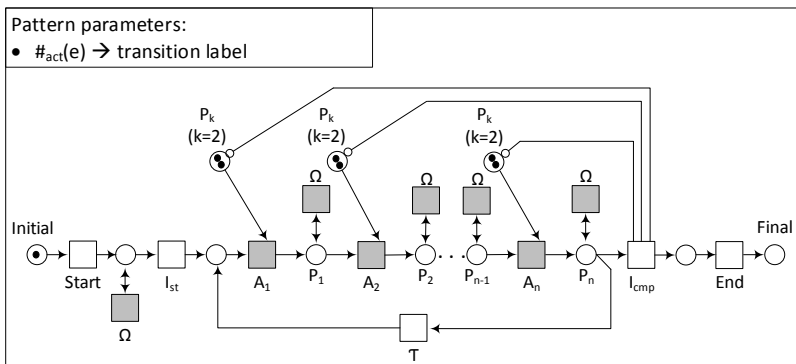


Figure A.19: *Bounded Sequence of Activities. Exactly  $k$  Times* compliance rule

*Pattern instantiation parameters:*

- $A_1, \dots, A_n$

- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n\}$
- $n$

## A.5 Parallel Category

This category includes compliance rules that limit the occurrence of an activity in parallel with or during another activity.

### Parallel. Simultaneous

*Description:* Activity  $A$  must always occur in parallel with activity  $B$  within a chosen scope. The compliance rule is violated if  $A$  and  $B$  does not occur simultaneously. The instance of this compliance rule includes start and completion of both activities  $A$  and  $B$ . Figure A.20 shows the Petri net pattern that formalizes this rule.

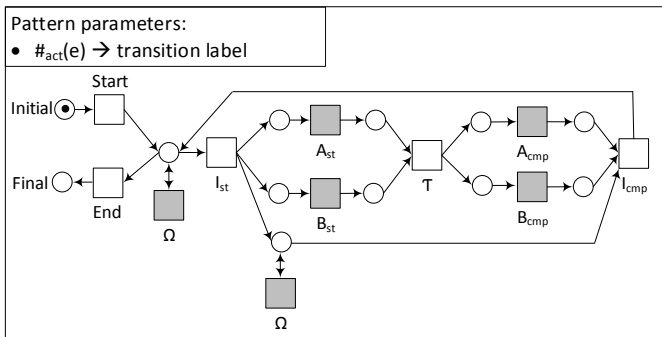


Figure A.20: *Parallel. Simultaneous* compliance rule

As it was mentioned earlier, some compliance rules require to model the start and completion of activities in the Petri net pattern. This compliance rule requires the activity  $A$  to be represented by two events  $A$ -start ( $A_{st}$ ) and  $A$ -complete ( $A_{cmp}$ ) indicating the start and completion of  $A$ . Likewise activity  $B$  is represented by two events  $B$ -start ( $B_{st}$ ) and  $B$ -complete ( $B_{cmp}$ ). After the pattern started, any event may occur. The *rule instance* is triggered as soon as activity  $A$  or  $B$  starts.



If *A* starts, *B* should also start, i.e., activity *A* cannot complete unless *B* has already started. Similarly, if start of the activity *B* activates the *rule instance*, it is required that activity *A* also starts. The pattern enforces by construction that the activities *A* and *B* must complete together otherwise the *rule instance* cannot complete, hence the pattern may not terminate. Note that,  $\Omega$ -labeled activity may occur independently from occurrences of *A* and *B* throughout the pattern. The transition *End* models that the end of the trace has been reached.

Pattern instantiation parameters:

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}\}$

**Parallel. During**

*Description:* Activity *B* must be executed during activity *A* within a chosen scope. The compliance rule is violated if *B* does not occur within execution of *A*. The instance of this compliance rule includes start and completion of both activities *A* and *B*. Fig. A.21 shows the Petri net pattern that formalizes this rule.

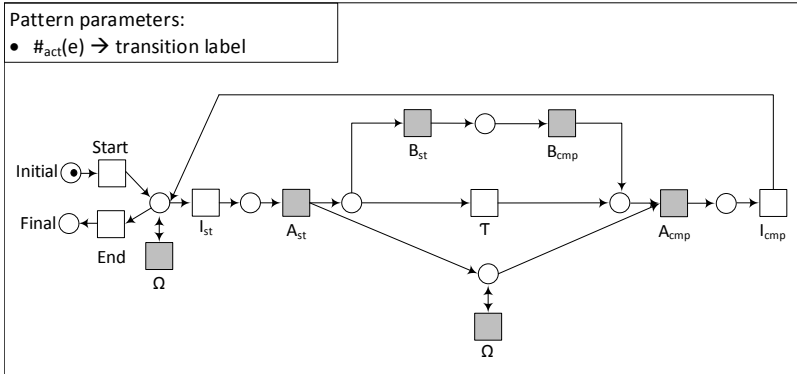


Figure A.21: ‘Parallel. During’ compliance rule

After the pattern started, any activity may occur. The *rule instance* is triggered as soon as activity *A* starts. Activity *B* may start only after *A* has started and *B* must complete before *A* completes. The pattern enforces by construction that if *B* occur, it must be executed during the execution of *A*. Please note that  $\Omega$ -labeled activity may occur independently from occurrences of *A* and *B*

throughout the pattern. The transition *End* models that the end of the trace has been reached.

Pattern instantiation parameters:

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}\}$

## A.6 Precedence Category

This category includes compliance rules that limit the occurrence of a one activity in precedence over other activities.

### Precedence. Simultaneous or Before

*Description:* Activity *A* must always occur before or simultaneously with activity *B* within a chosen scope. This rule is violated if activity *A* occurs after activity *B* within the specified scope. An instance of this compliance rule includes start and completion of both activities *A* and *B*. The Petri net pattern illustrated in Fig. A.22 formalizes this rule.

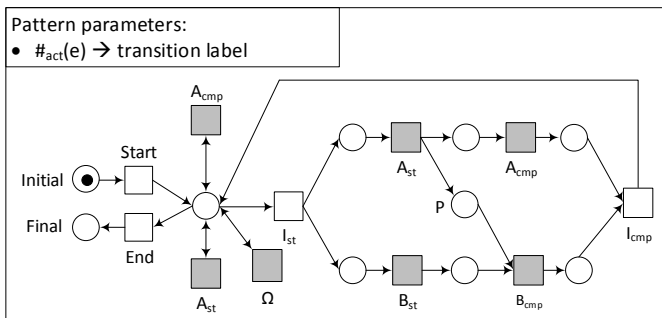


Figure A.22: *Precedence. Simultaneous or Before* compliance rule

The start of activity *B* or the start of activity *A* which will be followed by *B* triggers the *rule instance*. This pattern models two options (specified in the rule) for occurrences of *A* and *B*.

The case where activities *A* and *B* are executed simultaneously, requires that as soon as *A* starts activity *B* must be executed as well, i.e., *B* cannot complete

unless  $A$  has already started. This is specified by the pre-place  $P$  of  $B_{cmp}$ . The *rule instance* may only terminate if both activities  $A$  and  $B$  complete.

The case where  $A$  completes directly before  $B$  starts, is also described in the main cycle of the *rule instance*. After the *rule instance* started,  $A$  starts and completes and directly after that  $B$  starts and completes. In this case also completion of both activities  $A$  and  $B$  is required such that *rule instance* can complete, hence the pattern may terminate by firing the transition  $End$ .

There is no part of the pattern that permits the execution of activity  $B$  without a preceding or simultaneous  $A$ . If there is no  $B$  or if  $A$  just occurred, the *rule instance* is not activated and any event but  $B_{st}$  and  $B_{cmp}$  may occur. This is also the situation when the pattern may terminate by firing the transition  $End$ .

*Pattern instantiation parameters:*

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}\}$

**Precedence. Direct**

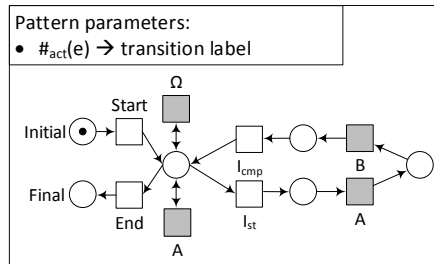


Figure A.23: *Precedence. Direct* compliance rule

*Description:* Every activity  $B$  must be preceded by activity  $A$  within a chosen scope. If  $B$  occurs without a directly preceding  $A$  within the specified scope, the rule is violated. An instance of this compliance rule includes execution of activity  $B$  and its preceding  $A$ . The Petri net pattern illustrated in Fig. A.23 formalizes this rule.

The *rule instance* is triggered as soon as an  $A$  occurs which is followed by  $B$ . The *rule instance* describes a cycle of  $A$  and  $B$ , such that  $B$  can only occur if  $A$  has directly preceded it. In this situation the *rule instance* can complete and

the pattern may terminate. If there is no  $B$  or  $A$  just occurred, any activity may occur. This is also the situation when the pattern may terminate by firing the transition  $End$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Precedence. Direct or Indirect**

*Description:* Every activity  $B$  must be preceded (directly or indirectly) by activity  $A$  within a chosen scope. If  $A$  does not occur before  $B$  within the specified scope, the rule is violated. An instance of this compliance rule includes execution of activity  $B$  and its preceding  $A$ . The Petri net pattern illustrated in Fig. A.24 formalizes this rule.

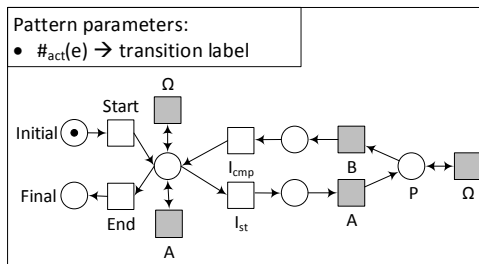


Figure A.24: *Precedence. Direct or Indirect* compliance rule

The pattern described in Fig. A.24 is similar to the pattern described in Fig. A.23; with the difference that the adjacent  $\Omega$ -labeled transition to place  $P$  in Fig. A.24, allows the indirect precedence of  $B$  with  $A$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Precedence. At Least Once**

*Description:* Every activity  $B$  must be preceded by activity  $A$  at least once within a chosen scope. If  $A$  does not precede  $B$  at least for one time within the spec-

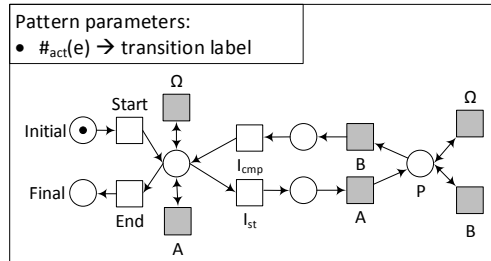


Figure A.25: *Precedence. At Least Once* compliance rule

ified scope, the rule is violated. An instance of this compliance rule includes execution of all  $B$  activities and a preceding  $A$ . The Petri net pattern illustrated in Fig. A.25 formalizes this rule.

After the pattern started any activity including  $A$  may occur. As soon as an activity  $A$  occurs which is followed by first  $B$ , the *rule instance* starts. The *rule instance* structure describes that  $B$  can only occur if before it at least one time  $A$  has occurred. As soon as  $A$  occurs, place  $P$  is marked. At this marking,  $B$  may occur an arbitrary number of times; because the condition of the rule is satisfied. After the *rule instance* is completed, the pattern may terminate by firing the transition *End*.

The pattern described in Fig. A.25 is similar to the pattern described in Fig. A.24; with the difference that the adjacent  $B$  transition to place  $P$  in Fig. A.25, allows arbitrary number of occurrences of  $B$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Precedence. Direct Multiple Activities

*Description:* Every activity  $B$  must be preceded directly by another execution of activity  $B$  or execution of activity  $A$  within a chosen scope. If directly before  $B$  one of the activities  $B$  or  $A$  does not occur within the specified scope, the rule is violated. An instance of this compliance rule includes all occurrences of  $B$  and a preceding  $A$ . The Petri net pattern illustrated in Fig. A.26 formalizes this rule.

After the pattern started any activity including  $A$  but  $B$  may occur; because before the first  $B$  at least once  $A$  must have occurred. As soon as an activity  $A$

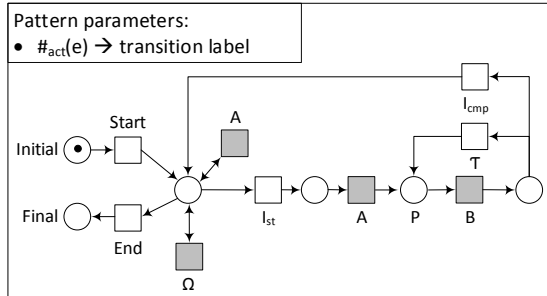


Figure A.26: Precedence. Direct Multiple Activities compliance rule

occurs which is followed by the first *B*, the *rule instance* starts. After the first occurrence of *A*, place *P* is marked. An this marking, *B* can occur an arbitrary number of times and still the condition of the rule is satisfied. That is, the pattern structure describes that *B* can only occur if directly before it any of the activities *A* or *B* was executed. After the *rule instance* is completed, the pattern may terminate by firing the transition *End*.

Pattern instantiation parameters:

- *A, B*
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Precedence. Direct or Indirect Multiple Activities**

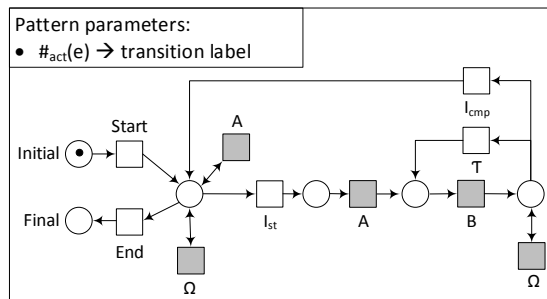


Figure A.27: Precedence. Direct or Indirect Multiple Activities compliance rule

*Description:* Every activity  $B$  must be preceded by another execution of activity  $B$  or the execution of activity  $A$  within a chosen scope. If before  $B$  an activity different from  $B$  or  $A$  occurs, within the specified scope, the rule is violated. An instance of this compliance rule includes all occurrences of  $B$  and a preceding  $A$ . The Petri net pattern illustrated in Fig. A.27 formalizes this rule.

The pattern in Fig. A.27 is similar to the pattern described in Fig. A.26; with the difference that as long as  $B$  is preceded (even indirectly) by any of the activities  $A$  or  $B$ , the condition of the rule is satisfied. The adjacent  $\Omega$ -labeled transition to place  $P$ , allows for indirect precedence of  $B$  by  $A$  or  $B$ . The transition *End* models that the end of the trace has been reached.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Precedence. Direct Multiple Different Activities**

*Description:* Every activity  $B$  must be directly preceded by activity  $C$  or  $A$  within a chosen scope. If within the specified scope, directly before  $B$  one of the activities  $A$  or  $C$  does not occur, the rule is violated. An instance of this compliance rule includes activity  $B$  and its preceding  $A$  or  $C$ . The Petri net pattern illustrated in Fig. A.28 formalizes this rule.

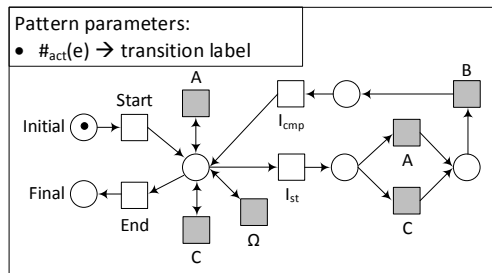


Figure A.28: *Precedence. Direct Multiple Different Activities* compliance rule

Being in the initial marking, any activity but  $B$  may occur. The rule instance starts as soon as an activity  $A$  or  $C$  occurs which is followed by  $B$ . The structure of the pattern is such that  $B$  may occur only if  $A$  or  $C$  has already occurred before

it. After the condition of the rule is satisfied, the *rule instance* may terminate. The transition *End* models that the end of the trace has been reached.

*Pattern instantiation parameters:*

- $A, B, C$
- $\Omega = \Sigma_L \setminus \{A, B, C\}$

**Precedence. Direct or Indirect Multiple Different Activities**

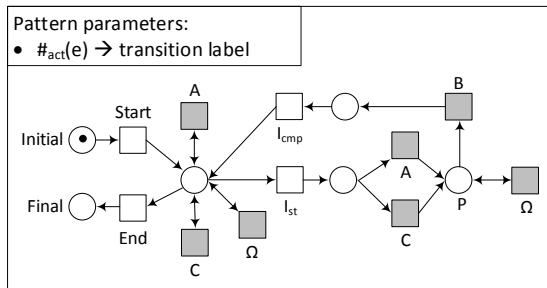


Figure A.29: *Precedence. Direct or Indirect Multiple Different Activities* compliance rule

*Description:* Every activity *B* must be preceded at least once by activity *C* or *A* within a chosen scope. If within the specified scope, before *B* one of the activities *A* or *C* does not occur, the rule is violated. An instance of this compliance rule includes activity *B* and its preceding *A* or *C*. The Petri net pattern illustrated in Fig. A.29 formalizes this rule.

The pattern in Fig. A.29 is similar to the pattern described in Fig. A.28; with the difference that as long as *B* is preceded (even indirectly) by any of the activities *A* or *C*, the condition of the rule is satisfied. The adjacent  $\Omega$ -labeled transition to place *P* allows for indirect precedence of *B* by *A* or *C*. The transition *End* models that the end of the trace has been reached. *Pattern instantiation parameters:*

- $A, B, C$
- $\Omega = \Sigma_L \setminus \{A, B, C\}$



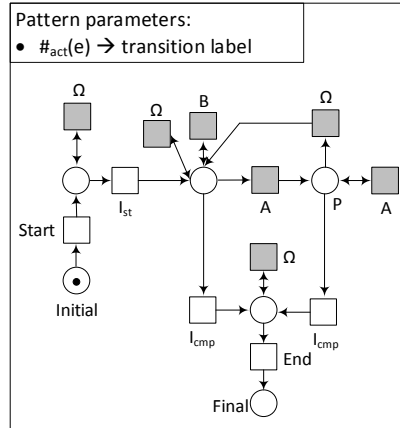


Figure A.30: *Precedence. Never Direct* compliance rule

### Precedence. Never Direct

*Description:* No activity  $B$  must be preceded directly by  $A$  within a chosen scope. If  $A$  occurs directly before  $B$  within the specified scope, the rule is violated. An instance of this compliance rule includes occurrence of all activities  $A$  and  $B$  and those  $\Omega$ -labeled activities which occur between occurrences of pair  $(A,B)$ . The Petri net pattern illustrated in Fig. A.30 formalizes this rule.

After the pattern started, any activity may occur. The *rule instance* triggers as soon as activity  $A$  or  $B$  occurs. The structure of the pattern should ensure that  $B$  cannot occur directly after  $A$ . Therefore, as soon as first  $A$  occurs, place  $P$  is marked and  $B$  is not enabled anymore.  $B$  may occur only after occurrence of an  $\Omega$ . When the condition of the rule is satisfied, the *rule instance* completes and the pattern may terminate by firing any of the transitions *End*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Precedence. Never

*Description:* Every activity  $B$  must never be preceded by  $A$  within a chosen scope. If  $A$  occurs before  $B$  within the specified scope, the rule is violated. An instance

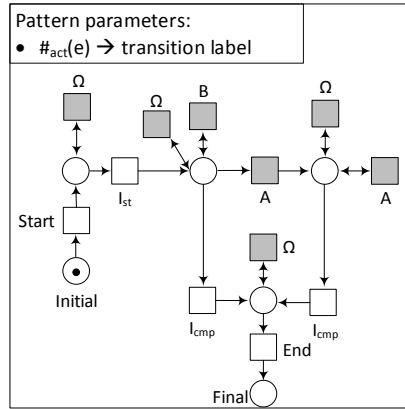


Figure A.31: Precedence. Never compliance rule

of this compliance rule includes occurrence of all activities *A* and *B*. The Petri net pattern illustrated in Fig. A.31 formalizes this rule. After the pattern started, any activity may be executed. The *rule instance* triggers as soon as is triggered as soon as first *A* or *B* occurs. After occurrence of the first *A*, the structure of the pattern should ensure that *B* cannot occur anymore. Therefore after *A* occurred, place *P* is marked and *B* is not enabled anymore. When the condition of the rule is satisfied, the *rule instance* completes and the pattern may terminate firing the transition *End*, if no event is to be executed.

Pattern instantiation parameters:

- *A, B*
- $\Omega = \Sigma_L \setminus \{A, B\}$

## A.7 Chain Precedence Category

This category of compliance rules limits occurrence of a sequence of activities in precedence over another sequence of activities.

### Chain Precedence. Direct

*Description:* Every sequence of activities  $\langle B_1, B_2, \dots, B_m \rangle$  must be preceded directly by the sequence of activities  $\langle A_1, A_2, \dots, A_n \rangle$  within a chosen scope. This

rule is violated if within the specified scope, directly before the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ , the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  does not occur. The Petri net pattern illustrated in Fig. A.32 formalizes this rule.

This pattern describes the allowed behaviors specified in the rule in *two* cycles. The *rule instance* is triggered as soon as first  $A_1$  occurs which later leads to the occurrence of the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  and is followed directly by the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ .

The main cycle of the pattern (please see the shadowed subnet labeled (a)), includes the *rule instance*. The rule instance is highlighted in the shadowed subnet labeled (b). When the *rule instance* is triggered, it is specified that the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  can only occur if the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  has occurred directly before it. If the condition of the rule is satisfied, the *rule instance* completes and the pattern return to the marking where place  $P$  is marked. At this marking, other events may occur, another instance of the pattern may start, or the pattern may terminate by firing transition *End*.

However, in the main cycle (subnet labeled (a)) if the *rule instance* is not triggered, from any place where any of the sequences  $\langle A_1, A_2, \dots, A_n \rangle$  or  $\langle B_1, B_2, \dots, B_m \rangle$  does not complete, it is possible to return to the marking where place  $P$  is marked or terminate the pattern if no other event occurs. The return paths are indicated by the smaller cycles inside the main cycle of the pattern.

Being at the marking where there is a token in place  $P$ , an occurrence of any sequence over the set of events  $\Sigma_L \setminus \{A_1, B_1\}$  is possible and the pattern can also terminate in this situation if it reaches its end. However as soon as  $A_1$  occurs, its occurrence is captured in the main cycle of the pattern in order to provide the possibility to detect the behavior if  $\langle A_1, A_2, \dots, A_n \rangle$  completes.

Likewise, as soon as  $B_1$  occurs, the left cycle (please see the shadowed subnet labeled (c)) in the pattern is followed, in order to avoid the completion of the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ . In this cycle, at most the occurrence of the sequence  $\langle B_1, B_2, \dots, B_{m-1} \rangle$  is possible. From any place in this cycle where the sequence  $\langle B_1, B_2, \dots, B_{m-1} \rangle$  does not complete, it is possible to return to the marking where there is a token in place  $P$  or terminate the pattern.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

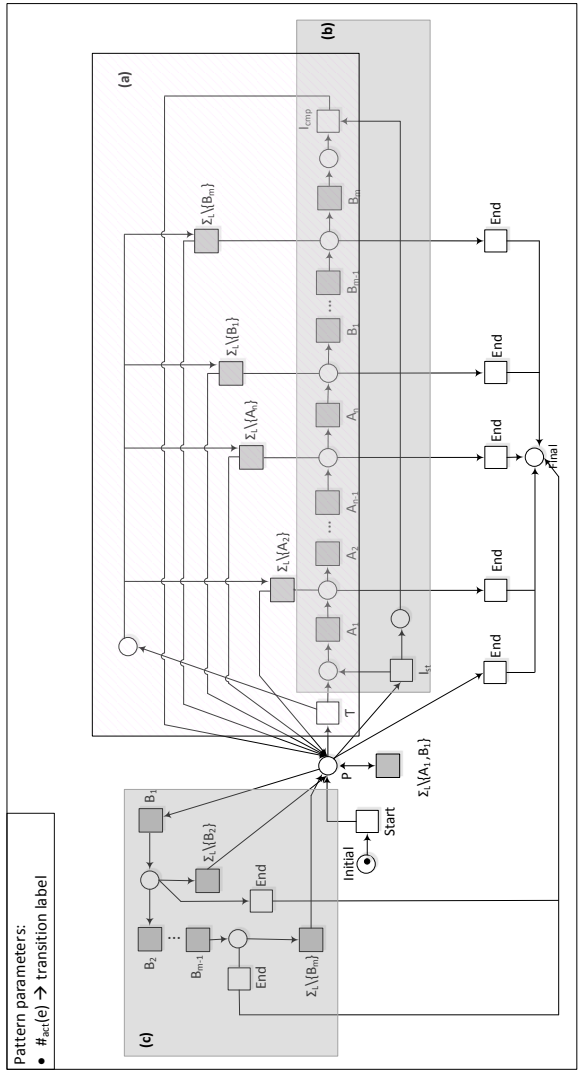


Figure A.32: Chain Precedence. Direct compliance rule

**Chain Precedence. Direct or Indirect**

*Description:* Every sequence of activities  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  must be preceded by the sequence of activities  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  within a chosen scope. The rule is violated if within the specified scope, before the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$ , the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  does not occur. The Petri net pattern illustrated in Fig. A.33 formalizes this rule.

The behavior of this pattern is similar to the pattern described in Fig. A.32, with the difference that both direct or indirect precedence of the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  by the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  considered to be compliant (based on the compliance rule).

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$



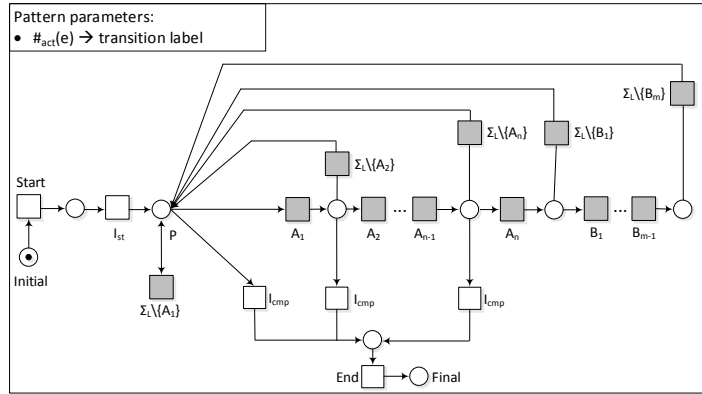


Figure A.34: Chain Precedence. Never Direct compliance rule

### Chain Precedence. Never Direct

*Description:* A sequence of activities  $\langle B_1, B_2, \dots, B_m \rangle$  must not be preceded directly by the sequence of activities  $\langle A_1, A_2, \dots, A_n \rangle$ . The rule is violated if within the specified scope and directly before the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ , the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  occurs. An instance of this compliance rule includes occurrence of all activities. The Petri net pattern illustrated in Fig. A.34 formalizes this rule.

The occurrence of any activity triggers the *rule instance*. In the main cycle of the pattern (*rule instance*), it is specified that the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  cannot occur if the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  has occurred directly before it. This is ensured by the last transition in the main cycle, being any transition but  $B_m$ ; implying that the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  can never complete directly after the sequence  $\langle A_1, A_2, \dots, A_n \rangle$ . From any place in the main cycle, where any of the sequences  $\langle A_1, A_2, \dots, A_n \rangle$  or  $\langle B_1, B_2, \dots, B_{m-1} \rangle$  does not complete, it is possible to return to the marking with a token in the place  $P$  or terminate the pattern, if no event is to be executed. The return paths are indicated with smaller cycles inside the main cycle of the pattern.

Being at the marking where there is a token in place  $P$ , the occurrence of any sequence over the set of events  $\Sigma_L \setminus \{A_1\}$  is possible. However, as soon as  $A_1$  occurs, its occurrence is captured in the main cycle of the pattern in order to provide the possibility to detect if  $\langle A_1, A_2, \dots, A_n \rangle$  completes. The *rule instance* can complete at any point in time if no activity is to be executed and consequently

the pattern may terminate.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

### Chain Precedence. Never

*Description:* Sequence of activities  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  must never be preceded by a sequence of activities  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  within a chosen scope. The rule is violated if within the specified scope and before the occurrence of the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$ , the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  occurs. An instance of this compliance rule includes the occurrence of all activities. The Petri net pattern illustrated in Fig. A.35 formalizes this rule.

The behavior of this pattern is similar to the pattern described in Fig. A.34, with the difference that the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  can never (neither directly nor indirectly) be preceded by the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$ . This is ensured in the pattern by the last transition in the main cycle, being any transition but  $B_m$  or  $\Omega$ ; implying that the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  can never complete after the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  occurred. In the main cycle, as soon as  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  completes, the place  $P_n$  is marked. At this marking, any event may occur as long as the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  does not complete. Moreover, after this marking, the cycle cannot return to the marking where  $P$  is marked to ensure that the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  can never occur (even indirectly) after the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$ . The *rule instance* can terminate at any point in time if no activity is to be executed and consequently the pattern may terminate.

Please note that  $\Omega$ -labeled event may occur any time throughout the entire pattern, even within the specified sequences in the compliance rule:  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  and  $\langle B_1, \dots, B_2, \dots, B_m \rangle$ .

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$



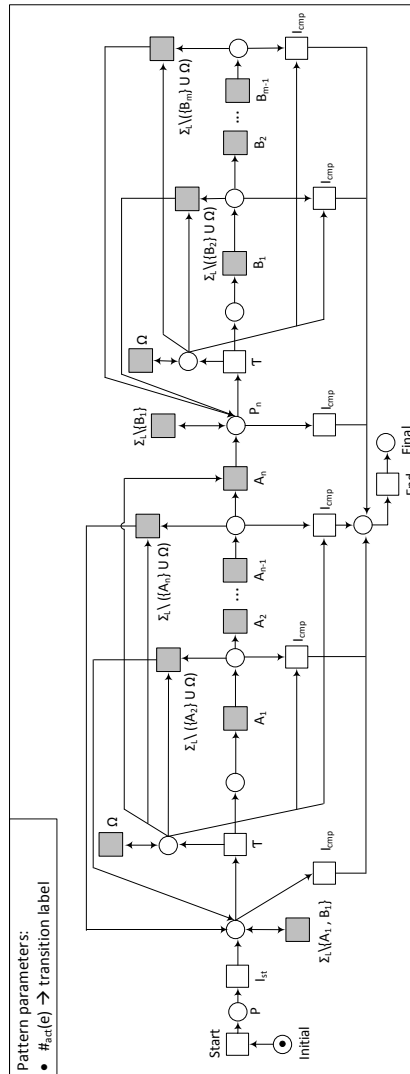


Figure A.35: Chain Precedence. Never compliance rule



In the current compliance rule execution of  $A$ , restricts the behavior of the pattern. Therefore,  $B$  and  $(\Omega)$ -labeled transitions are adjacent to place  $P$  implying; being at the marking where there is a token in place  $P$ , if there is no  $A$ ,  $B$  or any  $(\Omega)$ -labeled transitions may occur. This is also the situation when the pattern may terminate by firing transition  $End$ .

Pattern instantiation parameters:

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}\}$

**Response. Direct**

Description: Every activity  $A$  must be followed directly by activity  $B$  within a chosen scope. If within the specified scope,  $B$  does not occur directly after  $A$ , the rule is violated. An instance of this compliance rule includes activity  $A$  and an activity  $B$  which is preceded by  $A$ . The Petri net pattern illustrated in Fig. A.37 formalizes this rule.

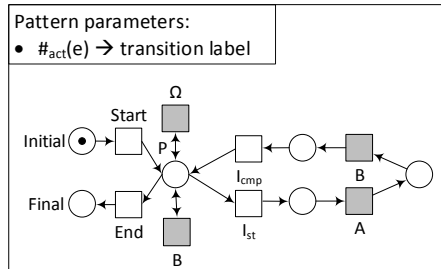


Figure A.37: Response. Direct compliance rule

The pattern described in Fig. A.37 is similar to the pattern described in Fig. A.23; with the difference in adjacent transitions to place  $P$ . In current pattern, occurrence of event  $A$  restricts the behavior of the pattern. Therefore,  $B$  and  $(\Omega)$ -labeled transitions are adjacent to place  $P$  implying; being in the marking with a token in place  $P$ , if there is no  $A$ ,  $B$  or any  $(\Omega)$ -labeled transitions may occur.

Pattern instantiation parameters:

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Response. Direct or Indirect

*Description:* Every activity  $A$  must be followed eventually by activity  $B$  within a chosen scope. If  $B$  does not occur after  $A$  within the specified scope, the rule is violated. An instance of this compliance rule includes activity  $A$  and an activity  $B$  which is preceded by  $A$ . The Petri net pattern illustrated in Fig. A.38 formalizes this rule.

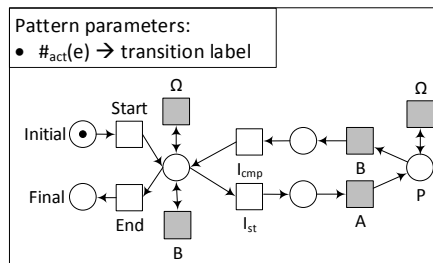


Figure A.38: *Response. Direct or Indirect* compliance rule

The pattern illustrated in Fig. A.38 is similar to the pattern described in Fig. A.37; with the difference that the adjacent  $\Omega$ -labeled transition to place  $P$  in Fig. A.38, allows that  $B$  indirectly follows any  $A$ .

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Response. At Least Once

*Description:* Every activity  $A$  must be followed eventually by activity  $B$  within a chosen scope. If within the specified scope,  $B$  does not occur at least once after  $A$ , the rule is violated. An instance of this compliance rule includes all activities  $A$  and their following activity  $B$ . The Petri net pattern illustrated in Fig. A.39 formalizes this rule.

The pattern structure describes that  $A$  can only occur if after it, at least one time  $B$  occurs. The pattern illustrated in Fig. A.39 is similar to the pattern described in Fig. A.38; with the difference that the adjacent  $A$ -labeled transition to place  $P$  in Fig. A.39, allows for an arbitrary number of occurrences of  $A$ . However, eventually  $B$  must occur to satisfy the condition of the rule.

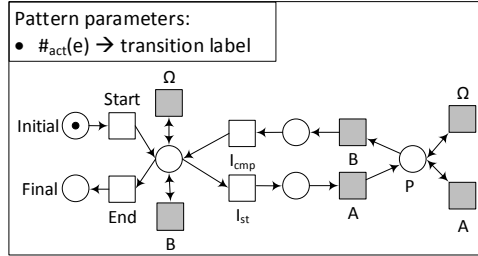


Figure A.39: Response. At Least Once compliance rule

Pattern instantiation parameters:

- A, B
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Response. Direct Multiple Activities**

*Description:* Every activity A must be followed directly by activity B or activity A within a chosen scope. If within the specified scope, directly after A one of the activities B or A does not occur, the rule is violated. An instance of this compliance rule includes an activity B and all activities A preceding it. The Petri net pattern illustrated in Fig. A.40 formalizes this rule.

The pattern structure describes that A can only occur if directly after it any of the activities A or B occurs. After the pattern started any event may occur.

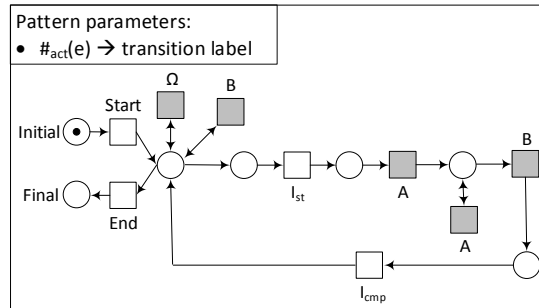


Figure A.40: Response. Direct Multiple Activities compliance rule

The *rule instance* starts as soon as *A* occurs. *A* may occur an arbitrary number of times, but the only possibility to complete the *rule instance* is the occurrence of *B*. When the condition of the rule is satisfied, the pattern can terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

**Response. Indirect Multiple Activities**

*Description:* Every activity *A* must be followed eventually by one of the activities *A* or *B* within a chosen scope. If within the specified scope, after *A* one of the activities *B* or *A* does not occur, the rule is violated. An instance of this compliance rule includes an activity *B* and all occurrences of activity *A* preceding it. The Petri net pattern illustrated in Fig. A.41 formalizes this rule.

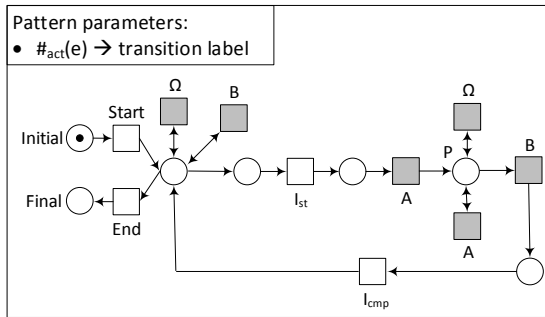


Figure A.41: *Response. Indirect Multiple Activities* compliance rule

The pattern illustrated in Fig. A.41 is similar to the pattern described in Fig. A.40; with the difference that as long as *A* is followed (even indirectly) by any of the activities *A* or *B*, the condition of the rule is satisfied. The adjacent  $\Omega$ -labeled transition to place *P* in Fig. A.41, allows that *A* is followed indirectly by *A* or *B*.

*Pattern instantiation parameters:*

- $A, B$
- $\Omega = \Sigma_L \setminus \{A, B\}$

### Response. Direct Multiple Different Activities

*Description:* Every activity  $A$  must be followed directly by activity  $B$  or activity  $C$  within a chosen scope. If directly after  $A$  one of the activities  $B$  or  $C$  does not occur, the rule is violated. An instance of this compliance rule includes activity  $A$  and its following activity  $B$  or  $C$ . The Petri net pattern illustrated in Fig. A.42 formalizes this rule.

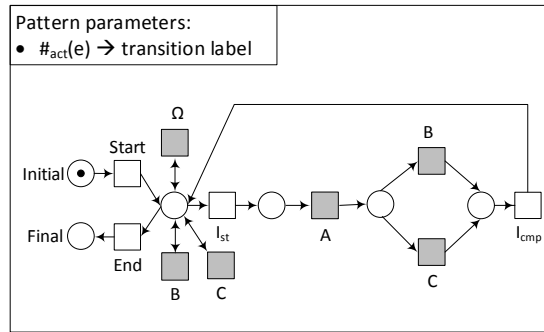


Figure A.42: *Response. Direct Multiple Different Activities* compliance rule

After the pattern started, any activity may occur. The *rule instance* is triggered as soon as  $A$  occurs. This pattern describes two options (specified in the rule) for occurrence of  $A$ . The case where  $B$  directly follows  $A$  is formalized by the upper cycle of the net and the case where  $C$  directly follows  $A$  is formalized by the lower cycle of the net. There is no cycle that permits an occurrence of activity  $A$  without a following  $B$  or  $C$ . When the condition of the rule is satisfied the *rule instance* completes and the pattern may terminate. The transition *End* models that the end of the trace has been reached, if no event is to be executed.

*Pattern instantiation parameters:*

- $A, B, C$
- $\Omega = \Sigma_L \setminus \{A, B, C\}$

## A.9 Chain Response Category

This category of compliance rules limits occurrences of a sequence of activities in precedence over another sequence of activities within a chosen scope.

The compliance patterns in this category are similar to the compliance patterns described in the category *Chain Precedence Category* in Sect. A.7, with slight differences in termination of the patterns and the transitions enabled after the pattern starts. In the patterns described in Sect. A.7, the occurrence of the sequence of activities  $\langle B_1, B_2, \dots, B_m \rangle$  puts a limitation on the behavior of the patterns; while in the patterns described in the current category *Chain Response Category*, the occurrence of sequence of activities  $\langle A_1, A_2, \dots, A_n \rangle$  limits the behavior of the patterns.

### Chain Response. Direct

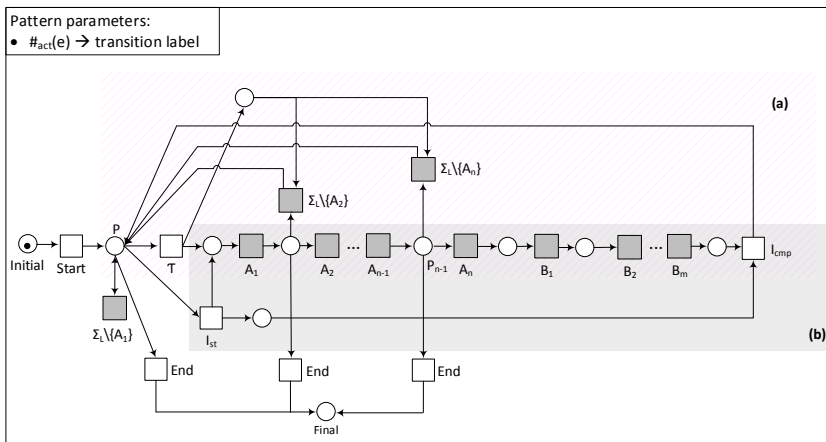


Figure A.43: ‘Chain Response. Direct’ compliance rule

*Description:* Every sequence of activities  $\langle A_1, A_2, \dots, A_n \rangle$  must be followed directly by a sequence of activities  $\langle B_1, B_2, \dots, B_m \rangle$  within a chosen scope. The rule is violated if within the specified scope, directly after the sequence  $\langle A_1, A_2, \dots, A_n \rangle$ , the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  does not occur. An instance of this compliance rule includes sequence of  $\langle A_1, A_2, \dots, A_n \rangle$  and its following activities which



is specified to be directly the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ . The Petri net pattern illustrated in Fig. A.43 formalizes this rule.

This pattern describes the allowed behavior specified in the rule in one main cycle. The main cycle of the pattern (subnet labeled (a)) includes the activities of the *rule instance* (shadowed subnet labeled (b)) as well as occurrences of other activities. The *rule instance* is triggered as soon as first  $A_1$  occurs which later leads to sequence  $\langle A_1, A_2, \dots, A_n \rangle$ . The *rule instance* structure is such that the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  can only occur if it is followed directly by the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ .

From any place in the main cycle (subnet labeled (a)), between place  $P$  and place  $P_{n-1}$  where the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  does not complete, it is possible to return to the marking where there is a token in  $P$  or terminate the pattern by firing one of the transitions *End* if no event is to be executed. The return paths are indicated with smaller cycles inside the main cycle of the pattern. As soon as the *rule instance* starts, i.e., as soon as sequence  $\langle A_1, A_2, \dots, A_n \rangle$  is complete, the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  must occur directly after that. After the completion of the *rule instance*, any event may occur, another *rule instance* may start or the pattern may terminate.

At the marking where there is a token in place  $P$ , the occurrence of any sequence over the set of events  $\Sigma_L \setminus \{A_1\}$  is possible and the pattern can also terminate without executing  $A_1$ . As soon as  $A_1$  occurs its occurrence is captured in the main cycle of the pattern in order to provide the possibility to detect the behavior if  $\langle A_1, A_2, \dots, A_n \rangle$  completes.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

### Chain Response. Direct or Indirect

*Description:* Every sequence of activities  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  must be followed eventually by sequence of activities  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  within a chosen scope. The rule is violated if within the specified scope and after the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$ , the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  does not occur. An instance of this compliance rule includes sequence of  $\langle A_1, A_2, \dots, A_n \rangle$  and its following activities which is specified to be the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ . The Petri net pattern illustrated in Fig. A.44 formalizes this rule.

The behavior of this pattern is similar to the pattern described in Fig. A.43, with the difference that both indirect or direct occurrence of sequence  $\langle B_1, \dots,$

$B_2, \dots, B_m$ ) after sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  considered to be compliant based on the compliance rule.

This pattern describes the allowed behavior specified in the rule in one main cycle. The main cycle of the pattern (subnet labeled (a)) includes the activities of the *rule instance* (shadowed subnet labeled (b)) as well as occurrences of other activities. The *rule instance* is triggered as soon as first  $A_1$  occurs which later leads to sequence  $\langle A_1, A_2, \dots, A_n \rangle$ . The *rule instance* structure is such that the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  can only occur if it is followed eventually by the sequence  $\langle B_1, B_2, \dots, B_m \rangle$ .

From any place in the main cycle between place  $P$  to  $P_{n-1}$  where the  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  does not complete, it is possible to return to the marking where there is a token in  $P$  or terminate the pattern by firing transition *End* if no event is to be executed. The return paths are indicated with smaller cycles inside the main cycle of the pattern. After the *rule instance* starts, i.e., sequence  $\langle A_1, A_2, \dots, A_n \rangle$  is complete, any activity may occur; implying the possibility that the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  can be followed indirectly by the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$ . The pattern cannot terminate anymore after  $P_n$  is marked unless the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  eventually completes, however it is possible to return to the marking where  $P_n$  is marked.

Please note that  $\Omega$ -labeled activity may occur any time throughout the entire pattern, even within the specified sequences of the rule:  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  and  $\langle B_1, \dots, B_2, \dots, B_m \rangle$ .

After the pattern starts, occurrence of any sequence over the set of events  $\Sigma_L \setminus \{A_1\}$  is possible and the pattern can also terminate without executing an  $A_1$ . As soon as  $A_1$  occurs its occurrence is captured in the main cycle of the pattern in order to handel the situation where  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  completes.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

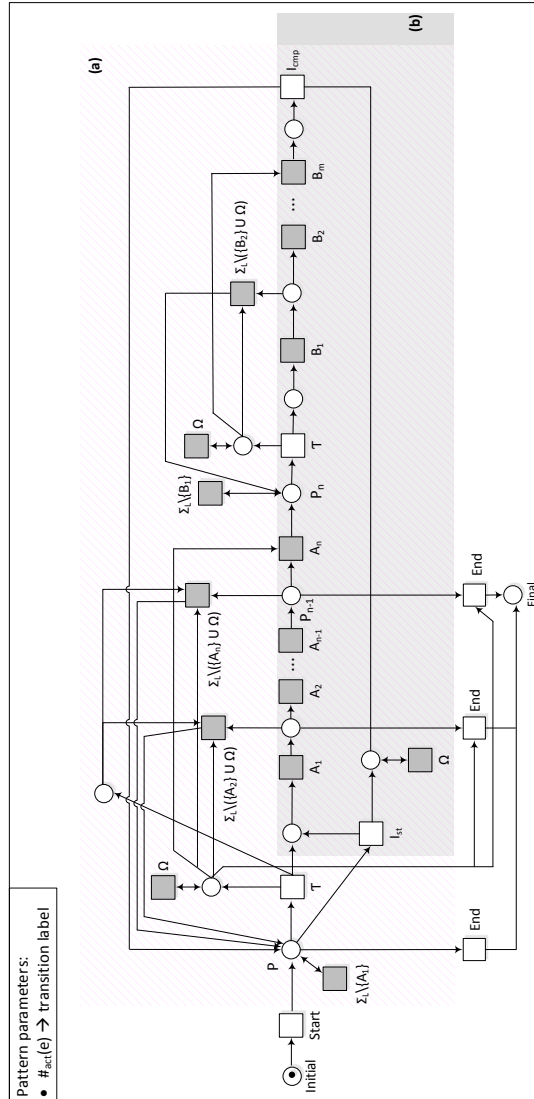


Figure A.44: Chain Response. Direct or Indirect compliance rule

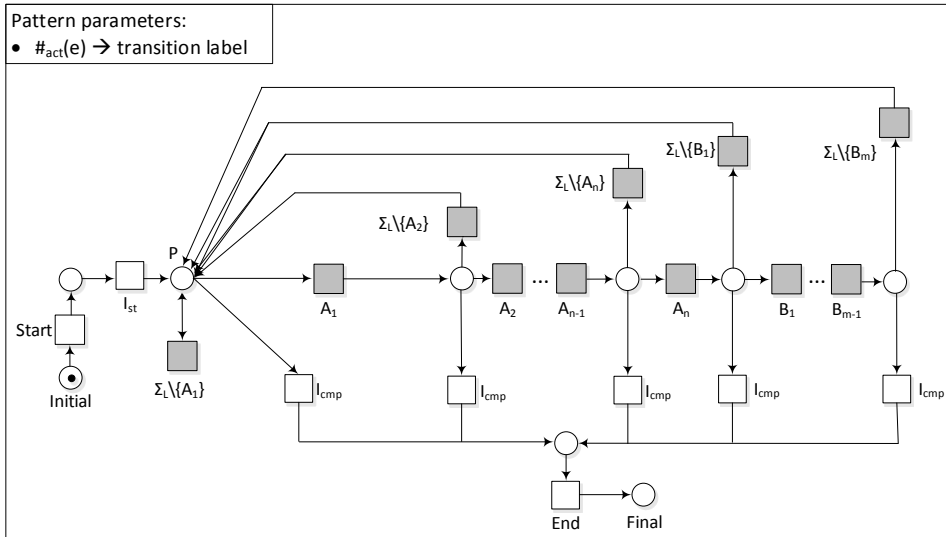


Figure A.45: Chain Response. Never Direct compliance rule

**Chain Response. Never Direct**

*Description:* Sequence of activities  $\langle A_1, A_2, \dots, A_n \rangle$  must never be followed directly by sequence of activities  $\langle B_1, B_2, \dots, B_m \rangle$  within a chosen scope. The rule is violated if within the specified scope and directly after the sequence  $\langle A_1, A_2, \dots, A_n \rangle$ , the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  occurs. An instance of this compliance rule includes occurrences of all activities. The Petri net pattern illustrated in Fig. A.45 formalizes this rule.

The occurrence of any activity in the main cycle of the pattern triggers the *rule instance* start. The *rule instance* structure is such that sequence  $\langle B_1, B_2, \dots, B_m \rangle$  cannot occur directly after the sequence  $\langle A_1, A_2, \dots, A_n \rangle$  has occurred. This is ensured by the last transition in the main cycle, being any transition but  $B_m$ ; implying that the sequence  $\langle B_1, B_2, \dots, B_m \rangle$  can never complete directly after the sequence  $\langle A_1, A_2, \dots, A_n \rangle$ . From any place in the *rule instance*, where sequences  $\langle A_1, A_2, \dots, A_n \rangle$  or  $\langle B_1, B_2, \dots, B_{m-1} \rangle$  does not complete, it is possible to return to the marking where the place  $P$  is marked. The pattern may terminate at any point in time if it reaches its end and no event is to be executed.

The remaining structure of this pattern was already explained in the pattern

described in Fig. A.43.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

### Chain Response. Never

*Description:* The sequence of activities  $\langle A_1, \dots, A_2, \dots, A_n \rangle$  must never be followed by sequence of activities  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  within a chosen scope. The rule is violated if within the specified scope and after the occurrence of the sequence  $\langle A_1, \dots, A_2, \dots, A_n \rangle$ , the sequence  $\langle B_1, \dots, B_2, \dots, B_m \rangle$  occurs. An instance of this rule includes occurrences of all activities. The Petri net pattern illustrated in Fig. A.35 formalizes the current compliance rule and the *Chain Precedence. Never* compliance rule. The behavior of the pattern was already described in Sect. A.7.

*Pattern instantiation parameters:*

- $A_1, \dots, A_n, B_1, \dots, B_m$
- $\Omega = \Sigma_L \setminus \{A_1, \dots, A_n, B_1, \dots, B_m\}$

## A.10 Between Category

This category of compliance rules, limits the occurrence of an activity within (between) a sequence of activities within a chosen scope.

### Between. After-Before

*Description:* Every activity  $B$  must always occur after an occurrence of activity  $A$  and before an occurrence of activity  $C$  within a chosen scope. The rule is violated if within the specified scope,  $B$  does not occur between  $A$  and  $C$  (after  $A$  and before  $C$ ). An instance of this compliance rule includes activity  $B$  and its preceding activity  $A$  and following activity  $C$ . The Petri net pattern illustrated in Fig. A.46 formalizes this rule.

After the pattern started any activity but  $B$  may occur. As soon as an activity  $A$  occurs which is followed by  $B$ , the *rule instance* starts.  $B$  can only occur if  $A$  has already occurred before it. Moreover,  $B$  must be followed eventually by  $C$ , otherwise there is no possibility that *rule instance* completes. When the

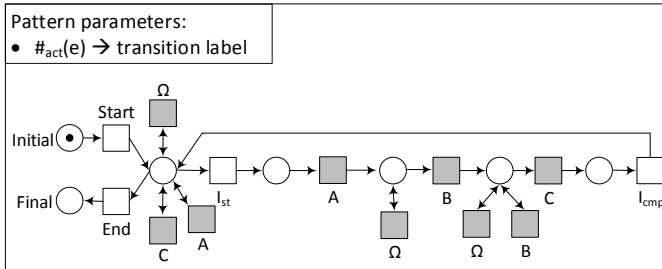


Figure A.46: *Between. After-Before* compliance rule

condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A, B, C$
- $\Omega = \Sigma_L \setminus \{A, B, C\}$

**Between. Simultaneously or After-Before**

*Description:* Every activity  $B$  must always occur directly after or simultaneously with an occurrence of activity  $A$  and directly before an occurrence of activity  $C$  within a chosen scope. The rule is violated if within the specified scope,  $B$  does not occur after or simultaneous with  $A$  and directly before  $C$ . An instance of this compliance rule includes activity  $B$  and its preceding or simultaneous activity  $A$  and following activity  $C$ . The Petri net pattern illustrated in Fig. A.47 formalizes this rule.

After the pattern started, any activity may occur. As soon as  $B$  starts or an activity  $A$  occurs which is followed by  $B$ , the *rule instance* starts. The compliance rule specifies two possibilities for occurrence of  $B$ : (1) simultaneously with  $A$  and directly before  $C$ , (2) directly after  $A$  and directly before  $C$ .

The occurrence of  $B$  with respect to  $A$  (simultaneous with  $A$  or directly after  $A$ ) is described in the shadowed subnet in Fig. A.47, which is similar to the structure already described in the pattern in Fig. A.22.

When activities  $A$  and  $B$  both complete, place  $P$  is marked. At this marking  $C$  must occur, otherwise there is no possibility to complete the *rule instance*. When the condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

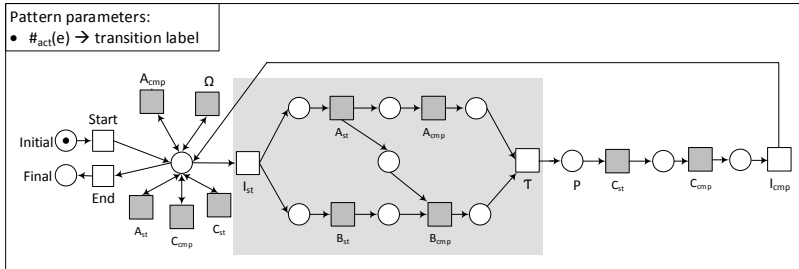


Figure A.47: *Between. Simultaneously or After-Before* compliance rule

Pattern instantiation parameters:

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}\}$

**Between. After-Simultaneously or Before**

*Description:* Every activity  $B$  must always occur directly after an occurrence of activity  $A$  and directly before or simultaneously with an occurrence of activity  $C$  within a chosen scope. The rule is violated if within the specified scope,  $B$  does not occur directly after  $A$  and directly before or simultaneously with  $C$ . An instance of this compliance rule includes activity  $B$  and its preceding activity  $A$  and following or simultaneous activity  $C$ . The Petri net pattern illustrated in Fig. A.48 formalizes this rule.

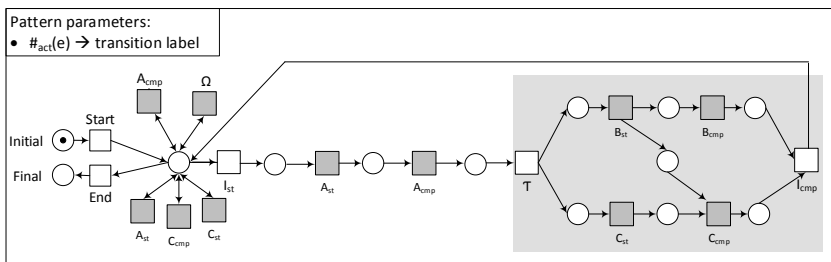


Figure A.48: *Between. After-Simultaneously or Before* compliance rule

After the pattern started any activity but  $B$  may occur. As soon as an activity  $A$  occurs which is followed by  $B$ , the *rule instance* starts. After  $A$  has occurred, the compliance rule specifies two possibilities for occurrence of  $B$ : (1) directly after  $A$  and simultaneous with  $C$ , (2) directly after  $A$  and directly before  $C$ .

The occurrence of  $B$  with respect to  $C$ , (simultaneous with  $C$  or directly before  $C$ ) is described in the shadowed subnet in Fig. A.48, which is similar to the structure already described in the pattern in Fig. A.22.

The *rule instance* may complete only if both  $B$  and  $C$  are completed. When the condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}\}$

**Between. Directly After-Directly Before**

*Description:* Every activity  $B$  must always occur directly after activity  $A$  and directly before activity  $C$  within a chosen scope. The rule is violated if within the specified scope,  $B$  does not occur between the sequence of activities  $\langle A, C \rangle$ . An instance of this compliance rule includes activity  $B$  and its directly preceding activity  $A$  and directly following activity  $C$ . The Petri net pattern illustrated in Fig. A.49 formalizes this rule.

This rule specifies the occurrence of the exact sequence of activities  $\langle A, B, C \rangle$ . After the *rule instance* starts, occurrence of any activity but  $B$  is possible. As soon as an activity  $A$  occurs which is followed directly by  $B$ , the *rule instance* starts. The *rule instance* may complete only if the exact sequence  $\langle A, B, C \rangle$  completes.

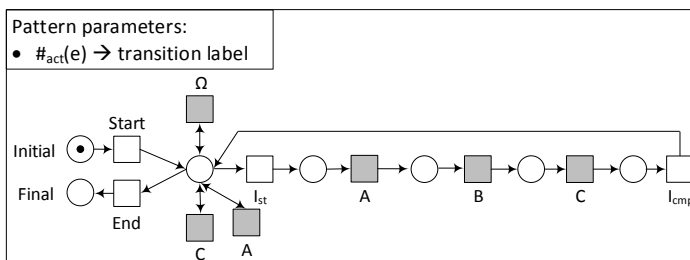


Figure A.49: *Between. Directly After-Directly Before* compliance rule



When the condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A, B, C$
- $\Omega = \Sigma_L \setminus \{A, B, C\}$

### Between. Simultaneously-Simultaneously

*Description:* Every activity  $B$  must always occur simultaneously with activities  $A$  and  $C$  within a chosen scope. If within the specified scope, activity  $B$  does not occur at the same time with occurrence of activities  $B$  and  $C$ , this compliance rule is violated. An instance of this compliance rule includes activity  $B$  and an activity  $A$  and an activity  $C$  which are executed simultaneously with  $B$ . The Petri net pattern illustrated in Fig. A.50 formalizes this rule.

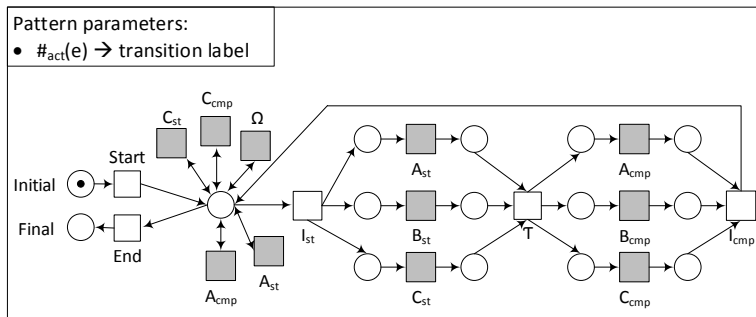


Figure A.50: *Between. Simultaneously-Simultaneously* compliance rule

After the pattern started, any activity may occur. The *rule instance* starts as soon as  $B$  starts or any of the activities  $A$  or  $C$  start which are simultaneous with  $B$ . After  $B$  started,  $B$  can only proceed for completion if the activities  $A$  and  $C$  have already started. When all the activities  $A, B$  and  $C$  are completed, the *rule instance* completes. When the condition of the rule is satisfied, the pattern may terminate by firing transition *End*.

*Pattern instantiation parameters:*

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}\}$

**Between. Simultaneously or After-Simultaneously or Before**

*Description:* Every activity  $B$  must always occur directly after or simultaneously with activity  $A$  and directly before or simultaneously with activity  $C$  within a chosen scope. This rule is violated if within the specified scope,  $B$  occurs before  $A$  or after  $C$  or not in the exact sequence of  $\langle A, B, C \rangle$ . An instance of this compliance rule includes activity  $B$  and its directly preceding or simultaneous  $A$  and its directly following or simultaneous activity  $C$ . The Petri net pattern illustrated in Fig. A.51 formalizes this rule.

The pattern described in the Fig. A.51 is the combination of two simpler patterns described already in the Fig. A.22 and the Fig. A.36; with  $B$  being the common element in them.

The *rule instance* starts as soon as  $B$  starts or an activity  $A$  starts which is followed directly by occurrence of  $B$ . The compliance rule specifies two possibilities for occurrence of  $B$  with respect to  $A$  : (1) directly after  $A$ , (2) simultaneous with  $A$ .

The occurrence of activity  $B$  with respect to  $A$  (simultaneous with  $A$  or directly after  $A$ ) is described in the shadowed subnet labeled (a) in Fig. A.51, which is similar to the structure described in the pattern in Fig. A.22.

Symmetrically the compliance rule specifies two possibilities for occurrence of  $B$  with respect to  $C$  : (1) directly before  $C$ , (2) simultaneous with  $C$ .

The occurrence of  $B$  with respect to  $C$  (simultaneous with  $C$  or directly before

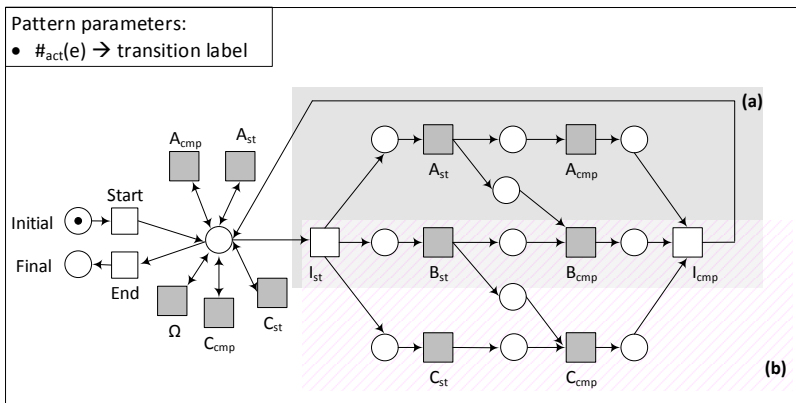


Figure A.51: *Between. Simultaneously or After-Simultaneously or Before* compliance rule

C) is described in the shadowed subnet labeled (b) in Fig. A.51, which is similar to the structure described in the pattern in Fig. A.36.

After the patterns started, any activity may occur. This is also the situation where the condition of the rule is satisfied and the pattern may terminate by firing transition *End*.

Pattern instantiation parameters:

- $A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}$
- $\Omega = \Sigma_L \setminus \{A_{st}, A_{cmp}, B_{st}, B_{cmp}, C_{st}, C_{cmp}\}$

**Between. At Least One Other Activity**

*Description:* This compliance rule specifies that *A* should never happen directly after *B* and *B* should never happen directly after *A*. This rule is violated if within the specified scope, *B* occurs directly after *A* or if *A* occurs directly after *B*. An instance of this compliance rule includes activity *A*, activity *B* and in case they appear in a sequence, the  $\Omega$ -labeled activity occur between the sequence  $\langle A \dots B \rangle$ . The Petri net pattern illustrated in Fig. A.52 formalizes this rule.

After the pattern started any activity may occur. The *rule instance* is triggered as soon as one of the activities *A* or *B* occurs. The right cycle in the pattern ensures that as soon as activity *A* occurs, it cannot be followed directly by *B*, i.e., *B* can only occur if after *A* at least one other activity ( $\Omega$ ) is executed. Symmetrically, the left cycle in the pattern ensures that as soon as *B* occurs, it cannot be followed directly by *A* and *A* can only occur if after *B* at least one other activity ( $\Omega$ ) is executed. The pattern can terminate at any point of time if no event is to be executed by firing transition *End*.

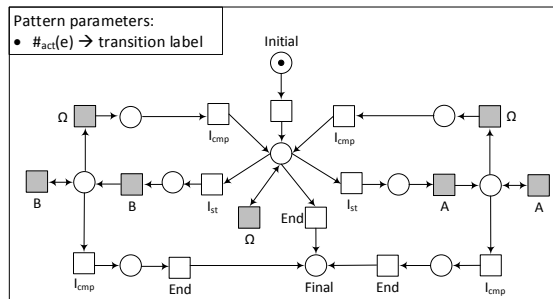


Figure A.52: *Between. At Least One Other Activity* compliance rule

# Appendix B

## Repository of Temporal Compliance Rules

This appendix presents a repository temporal compliance rules. Table 5.1 (also discussed in Chap. 2) lists our collection of temporal compliance rules. These rules are distributed over seven categories. Each category includes several compliance rules. Categories are formed based on the type of constraints they include.

The *generic temporal pattern*, discussed in Chap. 5 and visualized in Fig. B.1, together with respective guards formalizes our collection of temporal compliance rules. This pattern, together with the guards used to specify the temporal rules, follow same principles explained for the control-flow compliance patterns in Sect. 4.1.5. In addition, there are some specific principles about the temporal Petri net pattern and temporal rules. In the following, we discuss these principles.

- Occurrences of activity(s) specified in the temporal compliance rule are mimicked respectively by the  $X_1, \dots, X_n$ -labeled transitions.  $n$  equals the number of activities specified in the respective temporal compliance rule.
- Occurrences of any other activities than the activity(s) specified in the temporal compliance rules are mimicked by the  $\Omega$ -labeled transition. This way, the temporal Petri net pattern abstracts from all other activities that are not described in the temporal compliance rule.

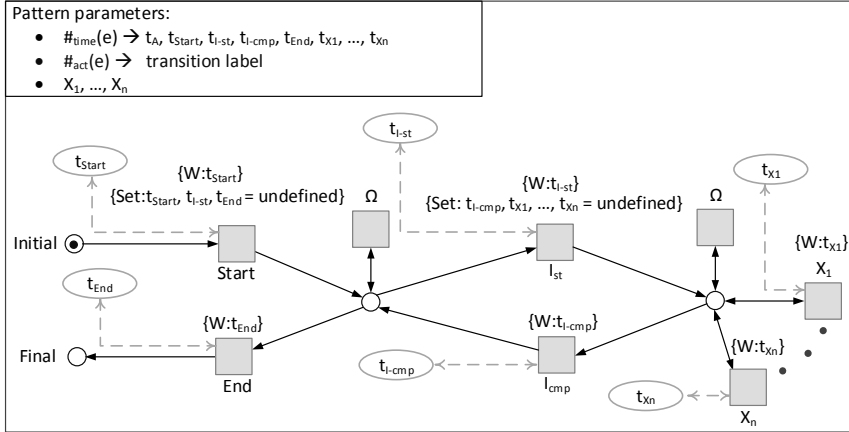


Figure B.1: The generic temporal pattern.

- Every transition in the temporal Petri net pattern (Fig. B.1) including  $\{I_{st}, I_{cmp}, X_1, \dots, X_n\}$  has a data attribute assigned to it which stores the timestamp of the events mapped into respective transitions. The value of the data attribute (the time stamp of the mapped event) is updated with the occurrence of the next mapped event.
- Occurrence of  $I_{st}$  writes the value *Undefined* over the time stamp of events  $\{X_1, \dots, X_n\}$ .
- Each temporal compliance rule comes with one or several guards. The guard or collection of guards describe all the possible compliant behaviors based on its corresponding temporal compliance rule.
- Guards are specified in terms of a set  $M$ . The set  $M$  in the temporal Petri net pattern denotes the set of timestamps which is compliant with the respective temporal compliance rule.
- A trace  $\sigma$  *complies* to a (rule of the) temporal Petri net pattern if after executing  $\sigma$ , the pattern reaches its *final marking* and all the time stamps of the events in the trace are elements of  $M$ .
- Every time interval in temporal compliance rules has a lower bound and an upper bound, indicated by  $[\alpha, \beta]$  or  $[\gamma, \zeta]$ .

- Every time interval in temporal compliance rules starts since time  $t$ , where  $t$  can be:
  - start of the rule instance ( $t_{I_{st}}$ )
  - completion of the rule instance ( $t_{I_{cmp}}$ )
  - start of the process instance ( $t_{case}$ )
  - start of the calendar ( $t_{Calendar}$ )
  - execution time of a specific event ( $t_{X_i}$ ) within the *rule instance*.

## B.1 Pattern Duration

This category of temporal compliance rules limits the time length in which a control-flow pattern must be executed.

### Pattern Duration. Pattern Duration

*Description:*

Every instance of a control flow pattern must be completed within  $[\alpha, \beta]$  time units since time  $t$ .

*Guard:*

$pattern\ duration(t, \alpha, \beta) \equiv t'_{I_{cmp}} \in M \wedge M = [t + \alpha, t + \beta]$  where  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{case}, t_{Calendar}, t_{X_{ist}}, t_{X_{icmp}}\}$

The guard *pattern duration* is assigned to the  $I_{cmp}$  transition in Fig. B.1.

### Pattern Duration. Negation Pattern Duration

*Description:*

No instance of a control flow pattern must be completed within  $[\alpha, \beta]$  time units since time  $t$ .

*Guard:*

$negation\ pattern\ duration(t, \alpha, \beta) \equiv t'_{I_{cmp}} \in M \wedge S = [0, \infty) \setminus [t + \alpha, t + \beta]$  where  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{Case}, t_{Calendar}, t_{X_{ist}}, t_{X_{icmp}}\}$

The guard *negation pattern duration* is assigned to the  $I_{cmp}$  transition in Fig. B.1.

## B.2 Delay Between Instances

This category of temporal compliance rules limits the delay between two instances of a control-flow rule and it has one temporal compliance rule.

*Description:*

The delay between execution of two instances of a control flow pattern must be within  $[\alpha, \beta]$  time units since time  $t$ .

*Guard:*

*delay between instances* $(t, \alpha, \beta) \equiv t'_{I_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{I_{cmp}}\} \wedge t_{I_{cmp}} \neq$   
*Undefined*

The guard *delay between instances* is assigned to the  $I_{st}$  transition in Fig. B.1.

## B.3 Validity

This category of temporal compliance rules limits the time length in which an activity must be executed or may be executed.

### Validity. Activity Duration

*Description:*

Every activity  $A$  within all instances of a control flow pattern must be completed within  $[\alpha, \beta]$  time units since it starts (time  $t_{A_{st}}$ ).

For this temporal rule, the temporal Petri net pattern in Fig. B.1), has  $n = 2$  transitions. Therefore the events  $A_{st}$  and  $A_{cmp}$  should be respectively mapped to the  $X_1$  and  $X_2$ -labeled transitions in the temporal Petri net pattern.

*Guard:*

*activity duration* $(A_{st}, A_{cmp}, t, \alpha, \beta) \equiv t'_{A_{cmp}} \in M \wedge S = [t + \alpha, t + \beta] \wedge X \neq A_{st} \wedge t \in$   
 $\{t_{A_{st}}\}$

The guard *activity duration* is assigned to the  $X_2$ -labeled transition in Fig. B.1, to which  $A_{cmp}$  is mapped.

### Validity. Activity Execution

*Description:*

Every activity  $A$  within all instances of a control flow pattern must/may be executed within  $[\alpha, \beta]$  time units since time  $t$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1, has  $n = 1$  transition. Therefore the event  $A$  is mapped to the  $X_1$ -labeled transitions in the temporal Petri net pattern.

Please note that this constraint might be associated only to start and completion events of an activity too; in order to restrict only the start or completion of the respective activity (e.g., latest or earliest start date, latest or earliest completion date). If the constraint restricts the start or completion of  $A$ ,  $A_{st}$  or  $A_{cmp}$  should be mapped to  $X_1$ -labeled transition in the temporal Petri net pattern.

*Guard:*

*activity execution*  $(A, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [t + \alpha, t + \beta]$  where  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{case}, t_{Calendar}, t_{X_i}\}$ .

The guard *activity execution* is assigned to the  $X_1$ -labeled transition in Fig. B.1 to which the exact event specified in the guard is mapped. This event can be  $A$  or  $A_{st}$  or  $A_{cmp}$  based on the guard. Please note that if  $t \in \{t_{X_i}\}$ , then the temporal Petri net pattern in Fig. B.1 will have  $n = 2$  transitions; Where  $X_i$  transition is mapped to the additional transition.

### Validity. Negation Activity Execution

*Description:*

No activity  $A$  within all instances of a control flow pattern may be executed within  $[\alpha, \beta]$  time units since time  $t$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1, has  $n = 1$  transition. Therefore the event  $A$  is mapped to the  $X_1$ -labeled transitions in the temporal Petri net pattern.

*Guard:*

*negation activity execution*  $(A, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [0, \infty) \setminus [t + \alpha, t + \beta]$  where  $t$  can be chosen by the user from  $t \in \{t_{I_{st}}, t_{I_{cmp}}, t_{case}, t_{Calendar}, t_{X_i}\}$ .

The guard *negation activity execution* is assigned to the  $X_1$ -labeled transition in Fig. B.1 to which  $A$  is mapped. Please note that if  $t \in \{t_{X_i}\}$ , then the temporal Petri net pattern in Fig. B.1 will have  $n = 2$  transitions; Where  $X_i$  transition is mapped to the additional transition.

## B.4 Time Restricted Existence

This category of temporal compliance rules limits the execution time of a given activity.



### Time Restricted Existence. Calendar Support Existence

*Description:*

Every activity  $A$  within all instances of a control flow pattern may only be executed at times  $t_1, \dots, t_n$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1) has  $n = 1$  transition. Therefore the event  $A$  is mapped to the  $X_1$ -labeled transition in the temporal Petri net pattern.

Please note that this constraint might be associated only to start and completion events of an activity too; in order to restrict only the start or completion of the respective activity (e.g., an activity must start only at times  $t_1, \dots, t_n$ ).

If the constraint restricts the start or completion of  $A$ ,  $A_{st}$  or  $A_{cmp}$  should be mapped to  $X_1$ -labeled transition in the temporal Petri net pattern.

*Guard:*

$$\text{calendar support existence}(A, t_1, \dots, t_n, \alpha, \beta) \equiv t'_A \in M \wedge M = \{t_1, \dots, t_n\}$$

The guard *calendar support existence* is assigned to the  $X_1$ -labeled transition in Fig. B.1 to which the exact event specified in the guard is mapped. This event can be  $A$  or  $A_{st}$  or  $A_{cmp}$  based on the guard.

### Time Restricted Existence. Calendar Support Absence

*Description:*

Every  $A$  within all instances of a control flow pattern must not be executed at times  $t_1, \dots, t_n$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1, has  $n = 1$  transition. Therefore the event  $A$  is mapped to the  $X_1$ -labeled transitions in the temporal Petri net pattern.

Please note that this constraint might be associated only to start and completion events of an activity too; in order to restrict only the start or completion of the respective activity (e.g., an activity must not start at times  $t_1, \dots, t_n$ ).

If the constraint restricts the start or completion of  $A$ ,  $A_{st}$  or  $A_{cmp}$  should be mapped to  $X_1$ -labeled transition in the temporal Petri net pattern.

*Guard:*

$$\text{calendar support absence}(A, t_1, \dots, t_n, \alpha, \beta) \equiv t'_A \in M \wedge M = [0, \infty) \setminus \{t_1, \dots, t_n\}$$

The guard *calendar support absence* is assigned to the  $X_1$ -labeled transition in Fig. B.1, to which the exact event specified in the guard is mapped. This event can be  $A$  or  $A_{st}$  or  $A_{cmp}$  based on the guard.

## B.5 Repetition Time Constraint

This category of temporal compliance rules limits the repetition of instances of a control flow pattern or an activity over time.

### Repetition Time Constraint. Delay Between Activities of One Kind

*Description:*

The delay between execution of two subsequent activities  $A$  in all instances of a control flow pattern, must be within  $[\alpha, \beta]$  time units since time  $t$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1 has  $n = 1$  transition. Therefore the event  $A$  is mapped to the  $X_1$ -labeled transitions in the temporal Petri net pattern.

*Guard:*

*delay between activities*  $(A, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_A\} \wedge t_A \neq \text{Undefined}$

The guard *delay between activities* is assigned to the  $X_1$ -labeled transition in Fig. B.1, to which  $A$  is mapped.

### Repetition Time Constraint. Delay Between Activities of Different Kinds

*Description:*

The delay between execution of two subsequent activities  $A$  and  $B$  in all instances of a control flow pattern, must be within  $[\alpha, \beta]$  time units since time  $t$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1 has  $n = 2$  transitions. Therefore the events  $A$  and  $B$  are respectively mapped to the  $X_1$ -labeled and  $X_2$ -labeled transitions in the temporal Petri net pattern.

*Guard:*

*delay between different activities<sub>1</sub>*  $(A, B, t, \alpha, \beta) \equiv t'_A \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_B\} \wedge t_B \neq \text{Undefined}$

The guard *delay between different activities<sub>1</sub>* is assigned to the  $X_1$ -labeled transition in Fig. B.1 to which  $A$  is mapped.

*Guard:*

*delay between different activities<sub>2</sub>*  $(A, B, t, \alpha, \beta) \equiv t'_B \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_A\} \wedge t_A \neq \text{Undefined}$

The guard *delay between different activities<sub>2</sub>* is assigned to the  $X_2$ -labeled transition in Fig. B.1 to which  $B$  is mapped.

Please note that we can limit the *start* of one activity to *completion* of another activity, in this case we can use the same guard explained above with this

difference that the parameters of the guard would be instantiated for *start* event of one activity and *completion* event of another activity.

## B.6 Time Dependent Variability

This category of temporal compliance rules includes one temporal compliance rule and it specifies that the control flow of the process will vary during execution with respect to time aspects.

*Description:*

Within all instances of a control flow pattern, activity *B* must be executed within  $[\alpha, \beta]$  time units since time  $t'$  if *A* has occurred within  $[\gamma, \zeta]$  time units since time  $t''$ .

For this temporal rule, the temporal Petri net pattern in Fig. B.1 has  $n = 2$  transitions. Therefore the events *A* and *B* are respectively mapped to the  $X_1$ -labeled and  $X_2$ -labeled transitions in the temporal Petri net pattern.

*Guard:*

*time dependent variability*  $(A, B, t', t'', \alpha, \beta, \gamma, \zeta) \equiv t'_B \in M \wedge M = [t' + \alpha, t' + \beta] \wedge t'_A \in [t'' + \gamma, t'' + \zeta]$

where  $t'$  and  $t''$  can be chosen by the user from  $t' \in \{t_{I_{st}}, t_{I_{cmp}}, t_{Start}, t_{Calendar}, t_{X_i}\}$  and  $t'' \in \{t_{I_{st}}, t_{I_{cmp}}, t_{Start}, t_{Calendar}, t_{X_i}\}$

The guard *time dependent variability* is assigned to the  $X_2$ -labeled transition in Fig. B.1 to which *B* is mapped. Please note that if  $t' \in \{t_{X_i}\}$ , or  $t'' \in \{t_{X_i}\}$  then the temporal Petri net pattern in Fig. B.1 will have more than 1 transitions; Where  $X_i$  transition is mapped to the additional transition.

## B.7 Overlap

This category of temporal compliance rules, limits the start and completion of one activity (*A*) to start and completion of another activity (*B*). The rules in this category are an extension of the '*Repetition Time Constraint.Delay Between Activities of different kinds*' temporal rule which is explained in Sect. B.5. For this temporal category, the temporal Petri net pattern in Fig. B.1, has ( $n = 4$ ) transitions. Therefore the events  $A_{st}, A_{cmp}, B_{st}$  and  $B_{cmp}$  are mapped respectively into  $X_1, \dots, X_4$ -labeled transitions in the temporal Petri net pattern.

**Overlap. Start After-Complete Before***Description:*

Within all instances of a control flow pattern, activity  $B$  must start within  $[\alpha, \beta]$  time units after activity  $A$  starts and activity  $B$  must complete within  $[\gamma, \zeta]$  time units before activity  $A$  completes. This temporal rule has two guards as follows.

*Guard:*

$$\text{start after-complete before}_1(A_{st}, B_{st}, t, \alpha, \beta) \equiv t'_{B_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{A_{st}}\} \wedge \{t_{A_{st}}\} \neq \text{Undefined}$$

The guard *start after-complete before*<sub>1</sub> is assigned to  $X_3$  transition in Fig. B.1 into which  $B_{st}$  is mapped.

*Guard:*

$$\text{start after-complete before}_2(A_{cmp}, B_{cmp}, t, \gamma, \zeta) \equiv t'_{A_{cmp}} \in M \wedge M = [t + \gamma, t + \zeta] \wedge t \in \{t_{B_{cmp}}\} \wedge \{t_{B_{cmp}}\} \neq \text{Undefined}$$

The guard *start after-complete before*<sub>2</sub> is assigned to the  $X_2$ -labeled transition in Fig. B.1 to which  $A_{cmp}$  is mapped.

**Overlap. Start Before-Complete Before***Description:*

Within all instances of a control flow pattern, activity  $B$  must start within  $[\alpha, \beta]$  time units before activity  $A$  starts and activity  $B$  must complete within  $[\gamma, \zeta]$  time units before activity  $A$  completes. This temporal rule has two guards as follows.

*Guard:*

$$\text{start before-complete before}_1(A_{st}, B_{st}, t, \alpha, \beta) \equiv t'_{A_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{B_{st}}\} \wedge \{t_{B_{st}}\} \neq \text{Undefined}$$

The guard *start before-complete before*<sub>1</sub> is assigned to the  $X_1$ -labeled transition in Fig. B.1 into which  $A_{st}$  is mapped.

*Guard:*

$$\text{start after complete before}_2(A_{cmp}, B_{cmp}, t, \gamma, \zeta) \equiv t'_{A_{cmp}} \in M \wedge M = [t + \gamma, t + \zeta] \wedge t \in \{t_{B_{cmp}}\} \wedge \{t_{B_{cmp}}\} \neq \text{Undefined}$$

The guard *start after complete before*<sub>2</sub> is assigned to the  $X_2$ -labeled transition in Fig. B.1 into which  $A_{cmp}$  is mapped.

**Overlap. Start After-Complete After***Description:*

Within all instances of a control flow pattern, activity  $B$  must start within  $[\alpha, \beta]$  time units after activity  $A$  starts and activity  $B$  must complete within  $[\gamma, \zeta]$  time units after activity  $A$  completes. This temporal rule has two guards:

*Guard:*

*start after-complete after<sub>1</sub>* ( $A_{st}, B_{st}, t, \alpha, \beta$ )  $\equiv t'_{B_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{A_{st}}\} \wedge \{t_{A_{st}}\} \neq \text{Undefined}$

The guard *start after-complete after<sub>1</sub>* is assigned to the  $X_3$ -labeled transition in Fig. B.1 into which  $B_{st}$  is mapped.

*Guard:*

*start after-complete after<sub>2</sub>* ( $A_{cmp}, B_{cmp}, t, \gamma, \zeta$ )  $\equiv t'_{B_{cmp}} \in M \wedge M = [t + \gamma, t + \zeta] \wedge t \in \{t_{A_{cmp}}\} \wedge \{t_{A_{cmp}}\} \neq \text{Undefined}$

The guard *start after-complete after<sub>2</sub>* is assigned to the  $X_4$ -labeled transition in Fig. B.1 into which  $B_{cmp}$  is mapped.

### Overlap. Start Before-Complete After

*Description:*

Within all instances of a control flow pattern, activity  $B$  must start within  $[\alpha, \beta]$  time units before activity  $A$  starts and activity  $B$  must complete within  $[\gamma, \zeta]$  time units after activity  $A$  completes. This compliance rule is indeed similar to the compliance rule explained in Sect. B.7 with the difference that position of the parameters of the rule is swapped. This temporal rule also has two guards:

*Guard:*

*start before-complete after<sub>1</sub>* ( $A_{st}, B_{st}, t, \alpha, \beta$ )  $\equiv t'_{A_{st}} \in M \wedge M = [t + \alpha, t + \beta] \wedge t \in \{t_{B_{st}}\} \wedge \{t_{B_{st}}\} \neq \text{Undefined}$

The guard *start before complete after<sub>1</sub>* is assigned to the  $X_1$ -labeled transition in Fig. B.1 into which  $A_{st}$  is mapped.

*Guard:*

*start before-complete after<sub>2</sub>* ( $A_{cmp}, B_{cmp}, t, \gamma, \zeta$ )  $\equiv t'_{B_{cmp}} \in M \wedge M = [t + \gamma, t + \zeta] \wedge t \in \{t_{A_{cmp}}\} \wedge \{t_{A_{cmp}}\} \neq \text{Undefined}$

The guard *start before complete after<sub>2</sub>* should be assigned to the  $X_4$ -labeled transition in Fig. B.1 into which  $B_{cmp}$  is mapped.

# Appendix C

## Repository of Configurable Control-Flow Compliance Rules

This appendix presents a repository of configurable compliance patterns. These patterns are built upon the repository of control-flow compliance rules (presented in Appendix A). This repository combines different category of control-flow compliance rules and extends the rules of each category with more variants. This repository reduced the number of compliance patterns from over 50 pattern to 18 configurable patterns. Together with all their configuration options, the configurable patterns can specify more than 1000 different compliance rules. We organize the configurable patterns in five groups based on the similarity of the rules.

Each configurable pattern has a core behavior and a set of configuration options. We call these configuration options “components” of a pattern. The components of a pattern may include Petri net nodes (i.e., transitions and places) and the edges that connect the nodes. The components of a pattern can include numeric parameters as well. For detailed discussion about different configuration options in configurable Petri net patters, see Sect. 4.4.

Next, we illustrate all the configurable patterns and their components in our repository and explain the range of compliance rules that can be specified using each configurable pattern. In addition, for the first configurable pattern (*Existence and Bounded Existence*), we describe its different components. We explain how configuring these components changes the behavior that is allowed by the pattern. For the remaining patterns in the repository, we describe only

the range of compliance rules that can be specified by the respective pattern.

## C.1 Existence

This group includes three configurable rules that specify a range of compliance rules related to execution of activities and the number of their executions. These configurable rules combine two categories: *Existence* and *Bounded Existence* of control-flow compliance rules and extends them with more variations of rules in each category. See Sect. A.1 and Sect. A.3 for details of the two categories.

Note that, the mapping:  $\#_{act}(e) \rightarrow \text{transition label}$  and  $\#_{label_N}(e) \rightarrow \text{transition label}$  hold for all the patterns in the repository. Therefore, we do not show these mapping in the each figure separately.

### Existence and Bounded Existence

This configurable pattern (shown in Fig. C.1) can specify a range of compliance rules that specify the execution of an activity and the number of its executions. Configuring this pattern, it is possible to set an upper limit and lower limit for the number of executions of an activity and generate new rules.

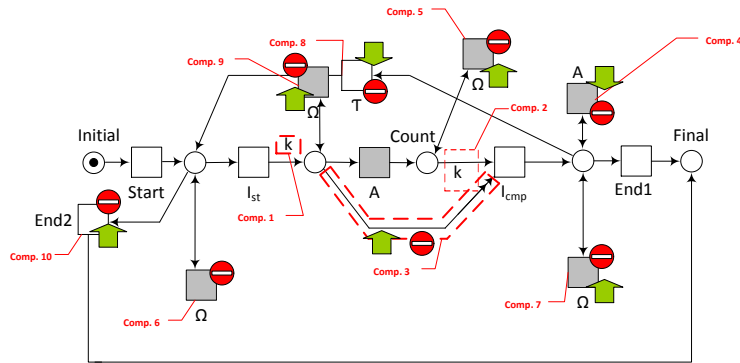


Figure C.1: Configuration options in *Bounded Existence-Upper Bound* configurable rule.

This configurable pattern has *ten* components. From these *ten* components, *two* are numeric parameters (*Comp.7* and *Comp.8*) and the remaining include Petri net nodes and edges.

The core behaviour of this pattern models the execution of activity  $A$  between  $I_{st}$  and  $I_{cmp}$ . The components ( $Comp.1$ ), ( $Comp.2$ ) and ( $Comp.3$ ) together specify the lower and upper bounds for executions of activity  $A$ . By setting a value for  $k$  and activating ( $Comp.1$ ), the execution of activity  $A$  is enabled for  $k$  times. To specify the lower limit for executions of activity  $A$  or the exact number of executions of activity  $A$ , ( $Comp.2$ ) needs to be activated. This component specifies that activity  $A$  should occur  $k$  times before the rule instance can complete by executing transition  $I_{cmp}$ . By activating ( $Comp.3$ ) and deactivating ( $Comp.2$ ), we can specify the upper bound for executions of activity  $A$ . Activating ( $Comp.4$ ) allows for executions of activity  $A$  more than  $k$  times.

By activating or blocking ( $Comp.5$ ), we allow or constrain the executions of other activities between executions of activity  $A$ . Activating ( $Comp.6$ ) allows the process to start with activities other than activity  $A$ . Blocking this component, specifies that the process may only start with activity  $A$ . Similarly, activating ( $Comp.7$ ) allow the execution of other activities after executions of  $A$ .

Activating ( $Comp.8$ ) allows for cyclic executions of activity  $A$ . Consider the compliance rule: “activity  $A$  must be executed in cycles of *three* occurrences”. In this case, if we set  $k=3$  and activate ( $Comp.1$ ), ( $Comp.2$ ), and ( $Comp.8$ ), we instantiate the pattern for cycles of *three* occurrences of activity  $A$ . Activating ( $Comp.9$ ) requires executions of at least one other activity between two cycle of executions of activity  $A$ .

Activating ( $Comp.10$ ) allows that the process ends without an execution of activity  $A$ .

Note that, activating or blocking any of the components (or combinations of these components) in the pattern changes the behavior specified by the pattern.



**Bounded Existence. Lower Bound**

By deciding about the configuration options in this pattern (shown in Fig. C.3), we can specify the lower limit for the number of occurrences of activity A in a row. This pattern only contains numeric configuration options.

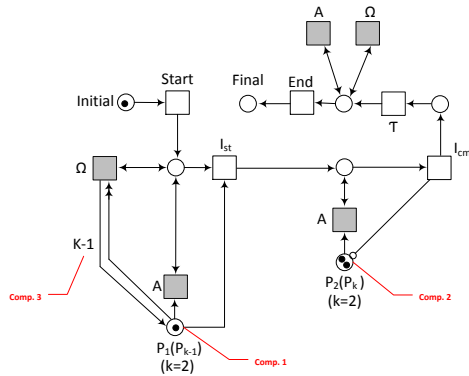


Figure C.2: Configuration options in *Bounded Existence. Lower Bound* configurable rule.

**Bounded Existence. Upper Bound**

By deciding about the configuration options in this pattern (shown in Fig. C.3), we can specify the upper limit for the number of occurrences of activity A in a row. This pattern only contains numeric configuration options.

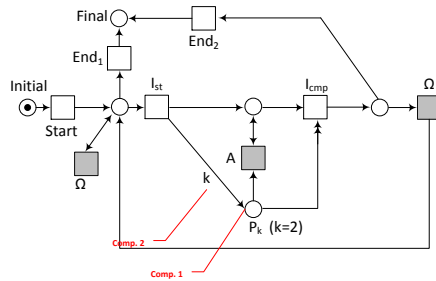


Figure C.3: Configuration options in *Bounded Existence. Upper Bound* configurable rule.

### Cyclic Occurrence

This configurable pattern (shown in Fig. C.4) can specify a compliance rule with the number of activity executions in a cycle and number of cycles. This pattern also include only numeric configuration options, i.e.,  $k$  can be set to specify the number of executions of activity  $A$  within one cycle and  $n$  can be set to specify the number of cycles of  $k$  occurrences of activity  $A$ .

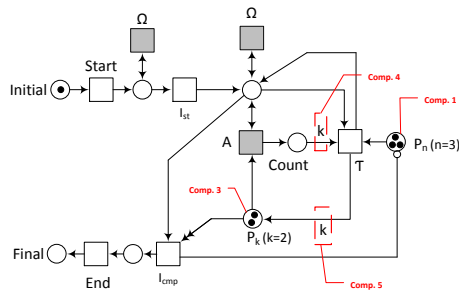


Figure C.4: Configuration options in *Cyclic Occurrence* configurable rule.

## C.2 Sequence

This group of configurable compliance rules combines and extends rules from three category of control-flow compliance rules including *Bounded Sequence Category*, *Response*, and *Precedence*. See Sect.A.4, Sect. A.8, and Sect. A.6 for details of the rules in each category.

### Sequence of (Multiple) Activities

The two configurable patterns (shown in Fig. C.5 and Fig. C.6) can specify a range of compliance rules that specify a sequence of activities.



**Negative Precedence or Response**

This configurable pattern (shown in Fig. C.7) can specify a range of compliance rules that forbids a sequence of activities.

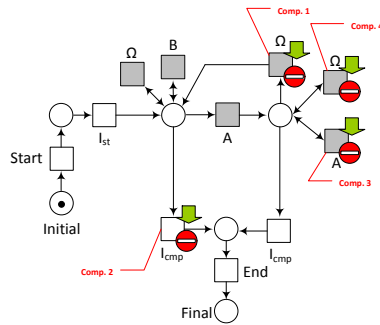


Figure C.7: Configuration options in *Negative Precedence or Response* configurable rule.

**Bounded Sequence**

This configurable pattern (shown in Fig. C.8) can specify a range of compliance rules that specify a sequence of activities and the number of their executions. This pattern contains both numeric configuration options and Petri net nodes and edges.

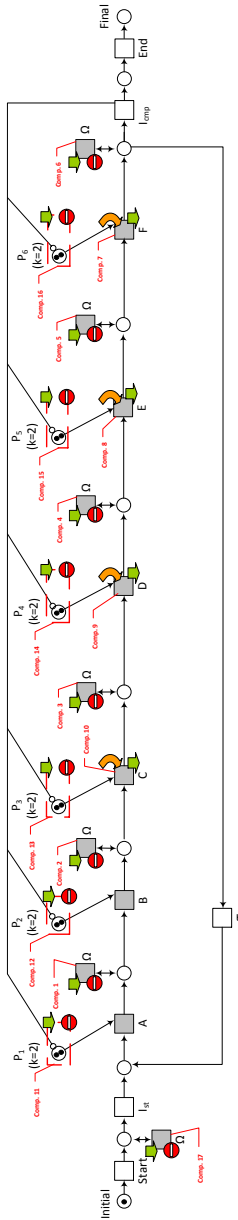


Figure C.8: Configuration options in *Bounded Sequence* configurable rule.

## C.3 Chain Precedence and Response

This group of configurable patterns combine and extends rules from two categories of *Chain Precedence*, and *Chain Response*. See Sect.A.7 and Sect. A.9 for details of the rules in each category.

### Chain Precedence

This configurable pattern (shown in Fig. C.9) can specify a range of compliance rules that specify a sequence of activities that precede each other and the number of the sequence execution.

### Chain Response

This configurable pattern (shown in Fig. C.10) can specify a range of compliance rules that specify a sequence of activities that succeed each other and the number of the sequence execution.

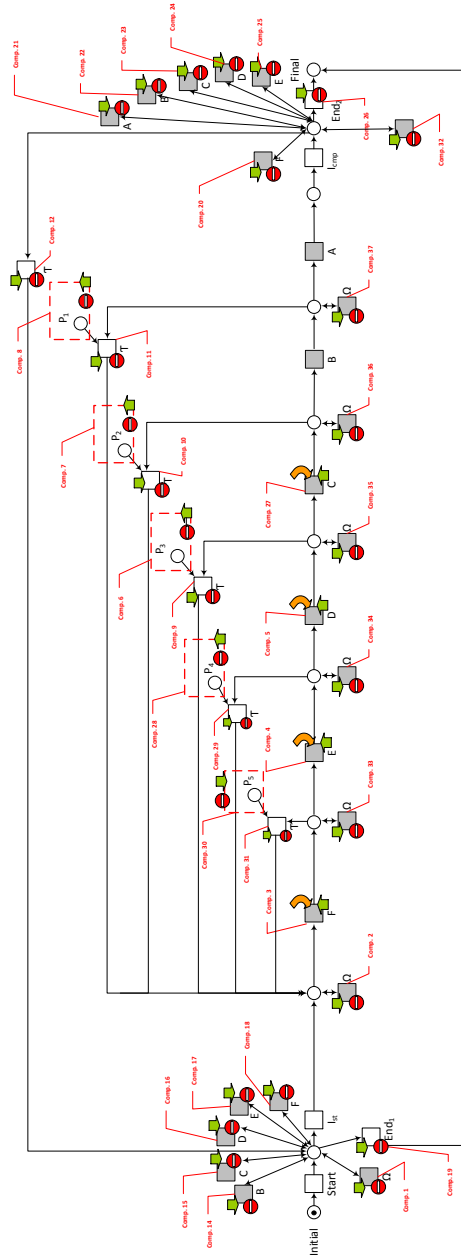


Figure C.9: Configuration options in Chain Precedence configurable rule.

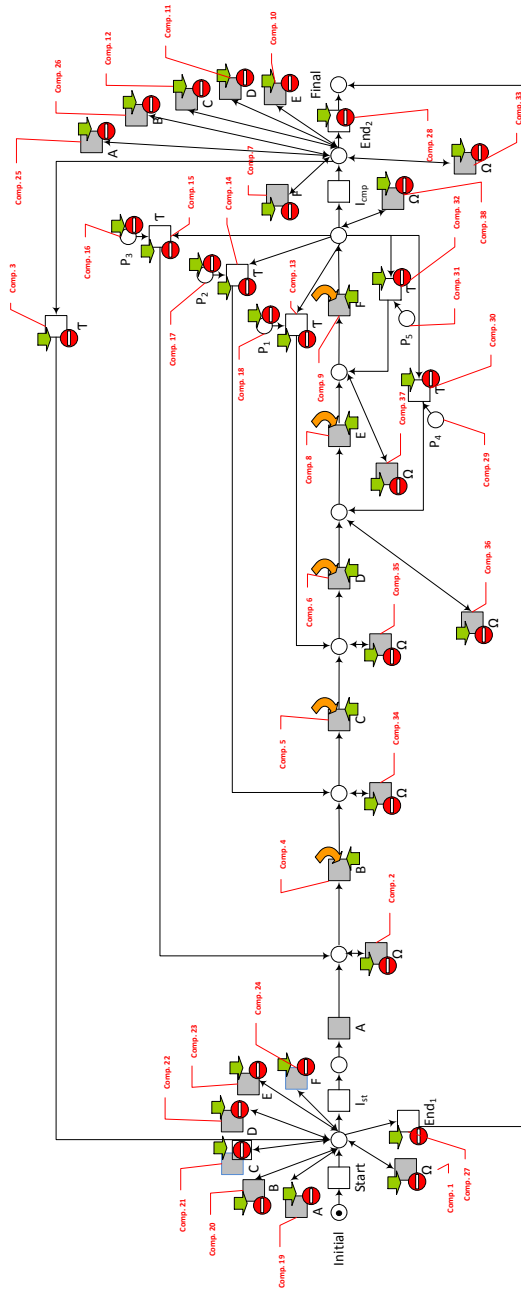


Figure C.10: Configuration options in Chain Response configurable rule.



## C.4 Parallel and Between

This group of configurable patterns, combine and extends rules from two categories of *Parallel*, and *Between*. See Sect.A.5 and Sect. A.10 for details of the rules in each category.

### Between

This configurable pattern (shown in Fig. C.11) can specify a range of compliance rules that constrain occurrence of an activity in between two activities.

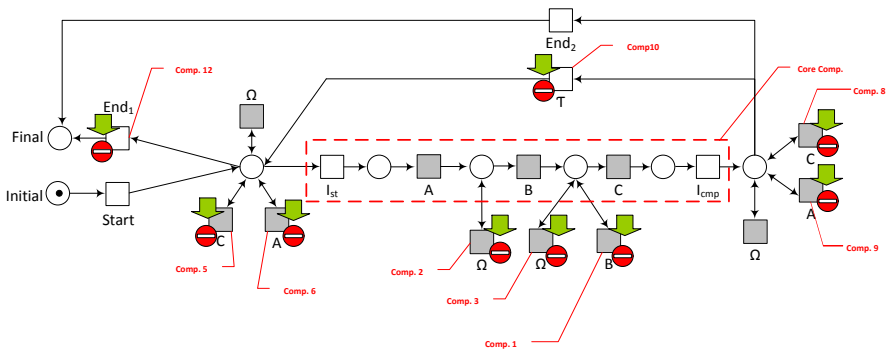


Figure C.11: Configuration options in *Between* configurable rule.

### Not-in-Between

This configurable pattern (shown in Fig. C.12) can specify a range of compliance rules that forbid the occurrence of an activity in between two activities.

### Parallel. During (Activity)

This configurable pattern (shown in Fig. C.13) can specify a range of compliance rules that constrain the occurrence of an activity during the execution of another activity.

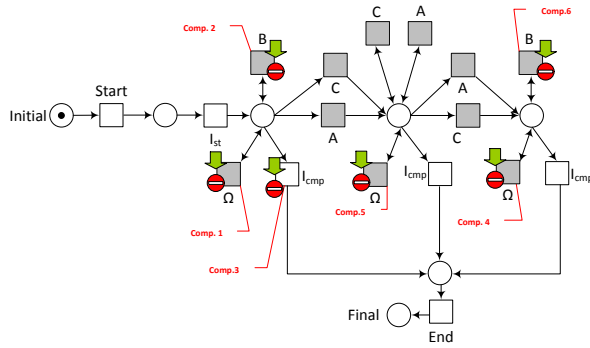


Figure C.12: Configuration options in *Not-in-Between* configurable rule.

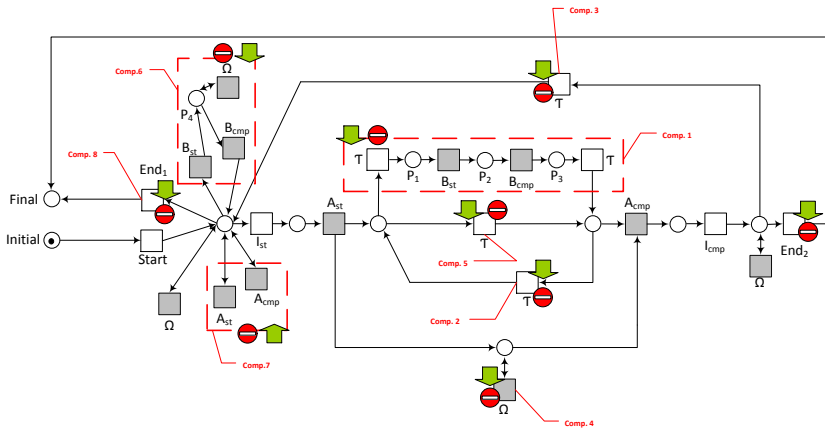


Figure C.13: Configuration options in *Parallel. During* configurable rule.

### Parallel. During (Sequence of Activities)

This configurable pattern (shown in Fig. C.14) can specify a range of compliance rules that constrain the occurrence of a sequence of activities between two other activities.

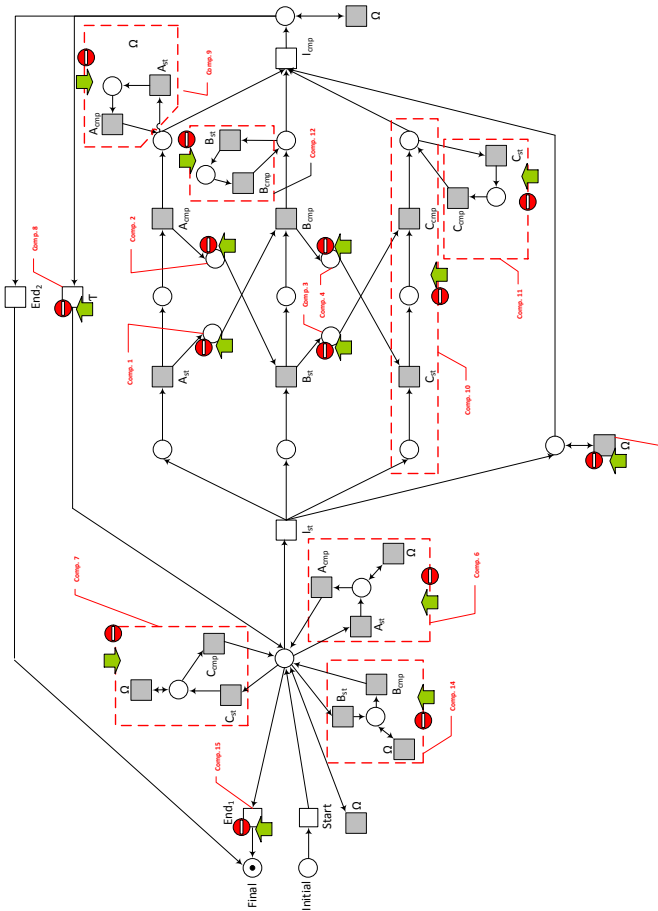


Figure C.14: Configuration options in *Parallel. During (Sequence of Activities)* configurable rule.

**Parallel. Simultaneous**

This configurable pattern (shown in Fig. C.15) can specify a range of compliance rules that constrain occurrences of several activities at the same time.

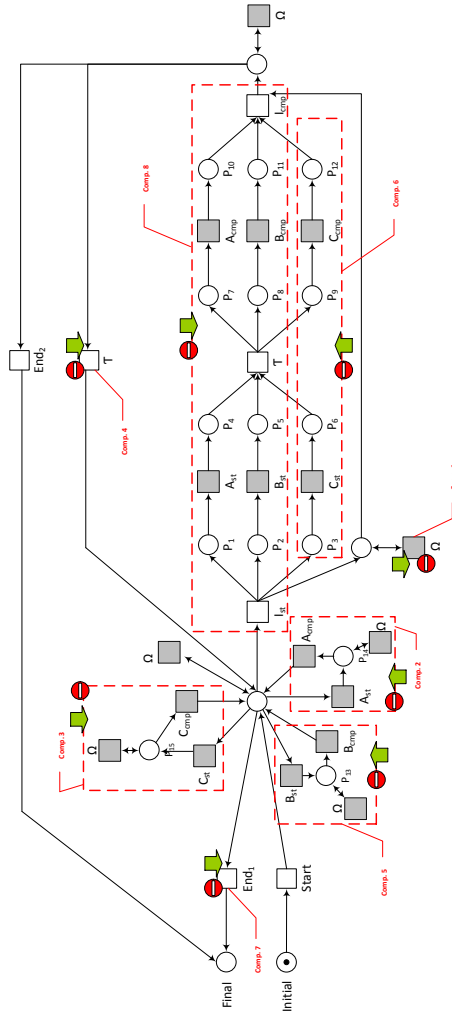


Figure C.15: Configuration options in *Parallel. Simultaneous* configurable rule.

## C.5 Dependent Existence

This group of configurable patterns, combine and extends rules from the category of *Dependent Existence*. See Sect.A.2 for details of the rules in this category.

### Inclusive, Pre-requisite and Co-requisite

This configurable pattern (shown in Fig. C.16) can specify a range of compliance rules related to inclusive, pre-requisite and co-requisite relations between activities. See Sect. A.2, Sect. A.2, and Sect. A.2 for details of these relations.

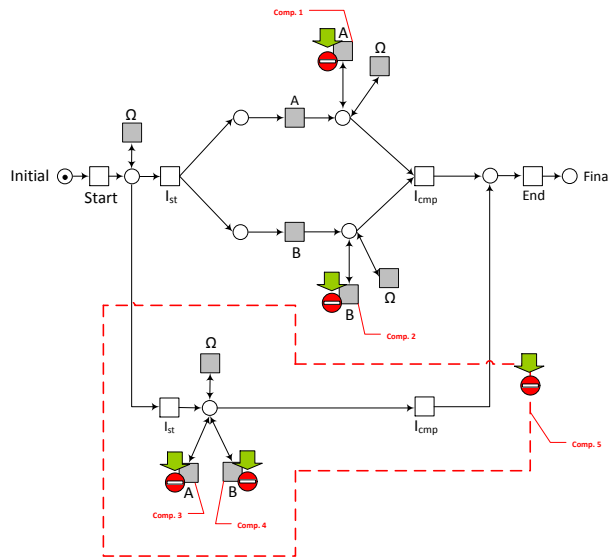


Figure C.16: Configuration options in *Inclusive, Pre-requisite and Co-requisite* configurable rule.

### Exclusive

This configurable pattern (shown in Fig. C.17) can specify a compliance rule related to the exclusive relation between activities. See Sect. A.2, and Sect. A.2 for details of the relation.

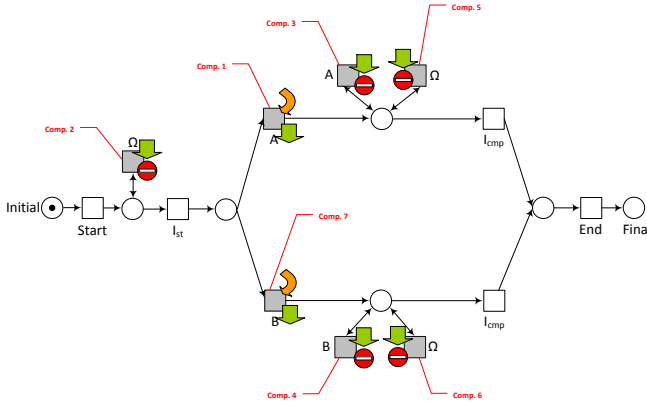


Figure C.17: Configuration options of *Exclusive Compliance* rule

**Substitute**

This configurable pattern (shown in Fig. C.18) can specify a compliance rule related to the substitute relation between activities. See Sect. A.2 for details of the relation.

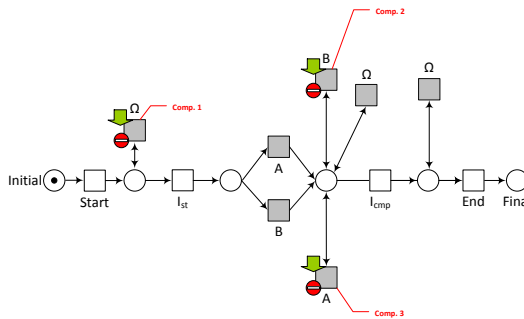


Figure C.18: Configuration options of *Substitute Compliance* rule

For some patterns including *Absence*, *Other Activity in Between*, we did not make any configurable rule because the rules were very specific and we could not find any configuration options. Details of these rules are explained in A.1 and A.10.



# Appendix D

## Results of the Case Study within UWV

In this appendix we describe the analyses performed for UWV case study in more detail. A summary of this analysis is discussed in Chap. 9.

### D.1 Analyzing Process Related Association Rules

Figure D.1 depicts the association rules (after filtering) that we mined between different violations and process related attributes in *Business Unit 1*. The brown ellipses in the middle, titled with the compliance rule numbers, show the violations of respective compliance rules. The other nodes refer to values of different attributes and they are grouped based on their attribute.

Next we will discuss the associations that were not discussed previously in Chap. 9.

**Associations between *application phases* and different violation types.** As we discussed earlier in Sect. 9.1, an application goes through different phases. It is interesting to see in Fig. D.1 that violations of *Rule 1* are connected to *continuation phase* in *Business Unit 1*. We can also observe that some letters that were supposed to be sent in *continuation phase Rule (5)-LA(E)* were sent in *claim* and *after closure phase*, and some letters *Rule (5)-LC(A)* that were supposed to be sent in *claim phase* were actually sent in *continuation phase*.



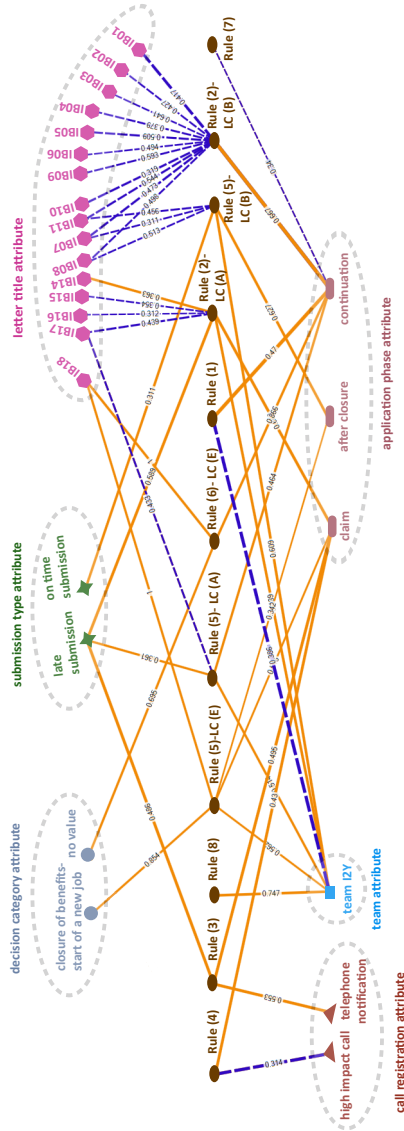


Figure D.1: Filtered association rules mined between different violations and process related attributes in *Business Unit 1*. The brown ellipses in the middle indicate violations related to different rules. The blue edges show associations of type *attribute value to violation* ( $A \rightarrow V$ ) and yellow edges indicate associations of type *violation to attribute value* ( $V \rightarrow A$ ).

By checking the association rules between different violations and application phases in other business units, we can observe the same pattern we found for *Business Unit 1*. The right part of Table D.1 summarizes the connections and their directions between *application phases* and different violation types for all business units. As can be seen in this table, the violating patterns are almost repeated in all the three units except violations of *Rule (4)* that are different in *Business Unit 3*.

**Associations between *application submission types* and different violations.**

Figure D.1 also shows interesting association rules between *application submission types* and violations of *Rule (5)-LC (B)*, *Rule 2-LC (A)*, *Rule 5-LC (A)*, and *Rule (3)*. Applications belong to one of the three different types: early submissions, on time submissions, and late submissions. However, we see that all association rules between violations and submission types are found for *late submissions* and *on time submissions*. These associations are both directed from violation type to the submission type, i.e., the violation implies the specified values for attribute *submission type*.

Our analysis of association rules between different violations and submission types reveals almost the same pattern in other business units as the pattern we observed for *Business Unit 1*. The left part of the Table D.1 summarizes these associations for all the three business units.

**Associations between *decision category* and different violations.** 46 different *decision categories* are recorded in *Business Unit 1*. Figure D.1 shows that violation of *Rule (6)* are connected with *no value* for *decision category* attribute, i.e., many of the violations of *Rule (6)* refer to situations where no value was registered for *decision category*. Similarly, violations of type *Rule (5)-LC (E)* are connected to *decision category = closure of benefits due to start of a new job*.

Table D.2 summarizes associations and their directions between decision category and different violations for the three business units. Note that decision category attribute is an *event attribute* for activity 'send letter' to clients. Hence, only associations can be found between different decision categories and violations of *Rules (2), (5), (6), and (7)*. The column denoted by "... " in this table represents other decision categories that where not connected with any violation. As can be seen in the table, the violating patterns between the three business units are almost similar. We found an association between violations of *Rule (5)* and *closure of benefits due to start of a new job* in the three business units. Associations were also found between violations of *Rule (6)* and situations where no value was registered for decision category attribute in *Business Unit 1* and *3*.

		Submission type			Application phase				
		Early submission	On time submission	Late submission	Orientation	Claim	Continuation	After Closure	After Rejection
Rule (1)	BU 1						V→A		
	BU 2						V→A		
	BU 3		A→V				V→A		
Rule (2)	BU 1			V→A		V→A	V→A		
	BU 2			V→A		V→A	V→A		
	BU 3		A→V	V→A		V→A	V→A		
Rule (3)	BU 1			V→A		V→A			
	BU 2			V→A		V→A			
	BU 3			V→A		V→A			
Rule (4)	BU 1					V→A			
	BU 2					V→A			
	BU 3								
Rule (5)	BU 1		V→A	V→A		V→A	V→A		
	BU 2		V→A	V→A		V→A	V→A	V→A	
	BU 3			V→A		V→A	V→A	V→A	
Rule (6)	BU 1						V→A		
	BU 2						V→A		
	BU 3						V→A		
Rule (7)	BU 1						A→V		
	BU 2						A→V		
	BU 3			V→A			V→A		
Rule (8)	BU 1								
	BU 2								
	BU 3								

Table D.1: Summary of connections (between submission types and different violations) and (between application phases and different violations) in three business units. The associations are of type *attribute value to violation* ( $A \rightarrow V$ ) or of type *violation to attribute value* ( $V \rightarrow A$ ).

		.....	Closure of benefits-start of a new job	Closure of benefits-moving abroad	No value
Rule (2)	BU 1 BU 2 BU 3			V→A	A→V
Rule (5)	BU 1 BU 2 BU 3		V→A V→A V→A		
Rule (6)	BU 1 BU 2 BU 3				V→A V→A
Rule (7)	BU 1 BU 2 BU 3		V→A		

Table D.2: Summary of connections between decision categories and different violations in the three business units. The associations are of type *attribute value to violation* ( $A \rightarrow V$ ) or of type *violation to attribute value* ( $V \rightarrow A$ ).

## D.2 Analyzing Client Related Association Rules

Figure D.2 depicts the association rules (after filtering) that we mined between different violations and client related attributes in *Business Unit 1*. The brown ellipses in the middle, titled with the compliance rule numbers, show the violations of respective compliance rules. The other nodes refer to values of different attributes and they are grouped based on their attribute. Next we will discuss the details of the mined association rules.

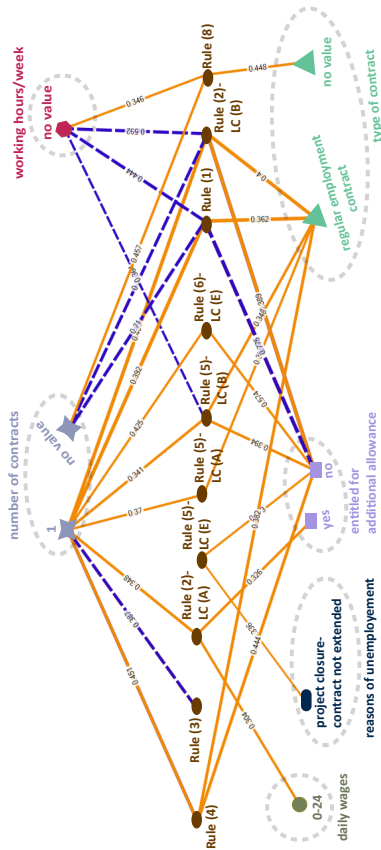


Figure D.2: Association rules mined between different violations and client related attributes in *Business Unit 1*.

**Associations between the *number of contracts* and different violation types.**

The attribute *number of contracts* records the information on the number of contracts a client held prior to un-employment. As can be seen in Fig. D.2, we found connections between different violations and value 'one' for the number of contracts. We believe that these connections mostly represent the absolute frequency of this value. Especially because most of these connections are directed from violations towards this attribute value, except violations of *Rule (3)*. In many situations this information is not recorded. Hence, we observe a connection between 'no value' for number of contracts and different violations.

We checked the association rules between different violations and the same attribute in other two business units. The same pattern is detected in other units as well. Many violations are connected to value 'one' as number of contacts.

**Associations between the *number of working hours per week* and different violation types.** The attribute *working hours per week* provides information on how many hours a client used to work before un-employment. As it is illustrated in Fig. D.2, many violations are connected to 'no value' for this attribute. This observation shows that we cannot really conclude a meaningful association between this attribute and violations of different types. However, we can conclude that in situations where an activity is violating, this information about the clients is often missing.

In *Business Unit (2)* we detected no association rules between *average working hours per week* and different violation types that have the frequency higher than 40 and a value more than 0.3 for its significance metric. The situation is different in *Business Unit (3)*. We found association rules between *average working hours per week*: '[0-3]' and violations of *Rules (1), (5)* directed from the attribute value towards violations. We discussed this result with domain experts in *Business Unit (1)* but we could not come to a clear conclusion about it.

**Associations between *daily wages* and different violation types.** As can be seen in Fig. D.2, violations of *Rule (2)* are connected to value [0-24] for attribute *daily wages* in *Business Unit 1*. We see the same pattern in *Business Unit 2*, i.e., *Rule (2)* is connected to value [0-24] for attribute *daily wages*. However, in *Business Unit 3* no important association between attribute *daily wages* and violations are observed. Domain experts also could not provide us a concrete explanation about these associations.

**Associations between *reasons of unemployment* and different violation types.** In total, 10 different reasons are recorded as unemployment reasons in the three business units. As can be seen in Fig. D.2, violations of *Rule (5)* in *Business Unit 1*

is connected only to '*project closure-contract not extended*' being the reason for unemployment.

The left part of the Table D.3 summarizes these associations for the three business units. We did not observe important associations between different violations and values of attribute reasons for unemployment. However, we see more associations between this attribute value and different violations in *Business unit 3*. As can be seen in this Table, the associations are found between '*project closure-contract not extended*' and different violations in the three business units.

**Associations between *being entitled for additional allowance* and different violation types.** Figure D.2 shows several connections between values of attribute '*entitled for additional allowance*' and different violation types. In general most of the clients are not entitled for an additional allowance. Hence, several connections of value '*no*' of this attribute with different violations may seem to reflect the absolute frequency of clients with this characteristic. Nevertheless, the association directed from value '*no*' of this attribute towards violations of *Rule (1)* deserves further investigation.

The right side of the Table D.3 compares the results of the three business units. Common patterns are observed among different business units, for instance w.r.t. *Rules (5), (6)* and value '*no*' for this attribute.

It is indicated in this table that most the associations are between different violations and value '*no*' for attribute *entitled for additional allowance*. Nevertheless, the direction of this associations varies.

**Associations between *type of contract* and different violation types.** Figure D.2 illustrates the values for the *type of contract* that are connected to different violations in *Business unit 1*. As can be seen many violations are associated with '*regular employment contracts*'. We assume these connections reflect the absolute frequency of this type of contract. Violations of *Rule (8)* as well is connected to '*no value*'. It is interesting to see that violations of *Rule (8)* are mostly connected to situations where information about the client is missing, which may be an indicator of a case to be more complex.

A similar pattern is observed in *Business unit 2*. Different violations are connected to '*regular employment contracts*'. Unlike *Business unit 1*, we do not observe an important association between '*no value*' for attribute *type of contract* and violations.

We observed for *Business unit 3* associations between *regular employment contracts* and different violations. Particularly associations between violations of *Rule (1), (2), and (4)* are interesting. Both of these associations are directed

		Reasons for unemployment		Entitled for additional allowance	
		...	Project closure- contract not ex- tended	Yes	No
Rule (1)	BU 1				A→V
	BU 2				V→A
	BU 3		A→V		A→V
Rule (2)	BU 1			V→A	V→A
	BU 2				V→A
	BU 3		A→V		A→V
Rule (3)	BU 1				
	BU 2				
	BU 3		A→V		A→V
Rule (4)	BU 1				V→A
	BU 2			V→A	
	BU 3				A→V
Rule (5)	BU 1		V→A		V→A
	BU 2			V→A	V→A
	BU 3		V→A	V→A	V→A
Rule (6)	BU 1				V→A
	BU 2				V→A
	BU 3				V→A
Rule (7)	BU 1				
	BU 2				
	BU 3				V→A
Rule (8)	BU 1				
	BU 2				
	BU 3				

Table D.3: Summary of connections between different violations and values of attributes entitled for additional allowance and reasons of unemployment. The associations are of type attribute value to violation ( $A \rightarrow V$ ) or of type violation to attribute value ( $V \rightarrow A$ ).



from the attribute value towards the violation type. Associations between this attribute value and violations of *Rule (1)* have a very high frequency (2430).

**Associations between *age* and different violation types.** We did not observe any important association between values of attribute *age* and different violations in *Business unit 1* and *Business unit 2*. However, we detected an association between '[25 – 29]' for attribute *age* and violations of *Rules (5)* and *(2)*; both directed towards the attribute value.

**Associations between *business sector* and different violation types.** We observed important associations between violations of *Rule (1)*, *Rule (2)* and value '*retail and whole sale*'<sup>1</sup> of attribute *business sector* in *Business unit 3*. Both associations are directed towards the attribute value. We did not find any important associations between values of this attribute and different violations in *Business unit 1* and *2*.

---

<sup>1</sup>In Dutch 'Winkelbedrijf en groothandel'

# Bibliography

- [1] BPIChallenge2011, [dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffc54](https://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffc54). (Cited on page 144.)
- [2] *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France*. IEEE, 2011. (Cited on pages 476, 483, and 484.)
- [3] 107th United States Congress. Sarbanes oxley act of 2002, July 2002. (Cited on pages 65, 68, 198, and 199.)
- [4] Nouha Abid, Silvano Dal-Zilio, and Didier Le Botlan. Real-time specification patterns and tools. In Stoelinga and Pinger [126], pages 1–15. (Cited on pages 9 and 149.)
- [5] Nouha Abid, Silvano Dal-Zilio, and Didier Le Botlan. Real-time specification patterns and tools. In Stoelinga and Pinger [126], pages 1–15. (Cited on pages 65, 67, 156, and 193.)
- [6] Joao Abreu, Paula Ventura Martins, Silvia Fernandes, and Marielba Zacarias. Business processes improvement on maintenance management: A case study. *Procedia Technology*, 9:320–330, 2013. (Cited on page 354.)
- [7] Rafael Accorsi, Thomas Stocker, and Günter Müller. On the exploitation of process mining for security audits: the process discovery case. In Shin

- and Maldonado [120], pages 1462–1468. (Cited on pages 65, 68, 198, and 199.)
- [8] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment based precision checking. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, volume 132 of *Lecture Notes in Business Information Processing*, pages 137–149. Springer, 2012. (Cited on page 10.)
- [9] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Towards robust conformance checking. In Michael zur Muehlen and Jianwen Su, editors, *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Hoboken, NJ, USA, September 13-15, 2010, Revised Selected Papers*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer, 2010. (Cited on page 10.)
- [10] 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. IEEE Computer Society, 2011. (Cited on pages 10, 89, 115, and 151.)
- [11] Charu C. Aggarwal and Philip S. Yu. A new framework for itemset generation. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 18–24. ACM Press, 1998. (Cited on page 260.)
- [12] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 207–216. ACM Press, 1993. (Cited on pages 255 and 359.)
- [13] Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors. *Business Process Management, 5th International Conference, BPM 2007, Brisbane,*

- Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*. Springer, 2007. (Cited on pages 472 and 479.)
- [14] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008. (Cited on pages 9, 65, 66, 109, and 370.)
- [15] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Specification, verification and explanation of violation for data aware compliance rules. In *ICSOC/SW*, volume 5900 of *LNCS*, pages 500–515, 2009. (Cited on pages 9 and 227.)
- [16] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Visually specifying compliance rules and explaining their violations for business processes. *J. Vis. Lang. Comput.*, 22(1):30–55, 2011. (Cited on page 9.)
- [17] Alistair P. Barros, Avigdor Gal, and Ekkart Kindler, editors. *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, volume 7481 of *Lecture Notes in Computer Science*. Springer, 2012. (Cited on pages 469 and 477.)
- [18] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Analysis of patient treatment procedures. In *BPM Workshops*, volume 99 of *LNBIP*, pages 165–166, 2012. (Cited on page 144.)
- [19] Reinhardt A. Botha and Jan H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001. (Cited on pages 65, 68, 198, and 199.)
- [20] Melike Bozkaya, Joost Gabriels, and Jan Martijn E. M. van der Werf. Process diagnostics: A method based on process mining. In Andrew Kusiak and Sang-goo Lee, editors, *International Conference on Information, Process, and Knowledge Management, eKNOW 2009, Cancun, Mexico, February 1-7, 2009*, pages 22–27. IEEE Computer Society, 2009. (Cited on page 356.)
- [21] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. (Cited on page 264.)

- [22] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA.*, pages 255–264. ACM Press, 1997. (Cited on pages 255, 260, and 359.)
- [23] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*, pages 1–8. IEEE, 2012. (Cited on page 266.)
- [24] Toon Calders, Christian W. Günther, Mykola Pechenizkiy, and Anne Rozinat. Using minimum description length for process mining. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 1451–1455. ACM, 2009. (Cited on pages 10 and 267.)
- [25] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001. (Cited on page 24.)
- [26] Rachel L. Cobleigh, George S. Avrunin, and Lori A. Clarke. User guidance for creating precise and accessible property specifications. In *SIGSOFT FSE*, pages 208–218. ACM, 2006. (Cited on pages 9, 136, and 149.)
- [27] Jonathan E. Cook and Alexander L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999. (Cited on pages 10 and 151.)
- [28] Nadja Damija, Talib Damijb, Janez Gradc, and Franc Jelencd. A methodology for business process improvement and is development. *Information and Software Technology*, 50:1127–1141, 2008. (Cited on page 354.)
- [29] Florian Daniel and Federico Michele Facca, editors. *Current Trends in Web Engineering - 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*, volume 6385 of *Lecture Notes in Computer Science*. Springer, 2010. (Cited on pages 478 and 480.)
- [30] Florian Daniel, Jianmin Wang, and Barbara Weber, editors. *Business Process Management - 11th International Conference, BPM 2013, Beijing*,

- China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*. Springer, 2013. (Cited on pages 469 and 478.)
- [31] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In Barros et al. [17], pages 82–97. (Cited on pages 10 and 151.)
- [32] Massimiliano de Leoni, Suriadi Suriadi, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Turning event logs into process movies: animating what has really happened. *Software & Systems Modeling*, pages 1–26, 2014. (Cited on page 279.)
- [33] Massimiliano de Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In Daniel et al. [30], pages 113–129. (Cited on pages 10, 115, 179, 180, 194, 228, and 267.)
- [34] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In Shin and Maldonado [120], pages 1454–1461. (Cited on page 27.)
- [35] Andrea Delgado, Barbara Weber, Francisco Ruiz, Ignacio García Rodríguez de Guzmán, and Mario Piattini. An integrated approach based on execution measures for the continuous improvement of business processes realized by services. *Information & Software Technology*, 56(2):134–162, 2014. (Cited on page 354.)
- [36] Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Access control: principles and solutions. *Softw., Pract. Exper.*, 33(5):397–421, 2003. (Cited on pages 65, 68, 198, and 199.)
- [37] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *FMSP*, pages 7–15, 1998. (Cited on pages 9 and 149.)
- [38] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999.*, pages 411–420. ACM, 1999. (Cited on pages 65, 66, 109, 193, and 370.)

- [39] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *CAiSE*, volume 1626 of *LNCS*, pages 286–300. Springer, 1999. (Cited on page 192.)
- [40] Amal Elgammal, Oktay Türetken, and Willem-Jan van den Heuvel. Using patterns for the analysis and resolution of compliance violations. *Int. J. Cooperative Inf. Syst.*, 21(1):31–54, 2012. (Cited on page 9.)
- [41] Amal Elgammal, Oktay Türetken, Willem-Jan van den Heuvel, and Mike P. Papazoglou. On the formal specification of regulatory compliance: A comparative analysis. In E. Michael Maximilien, Gustavo Rossi, Soe-Tsy Yuan, Heiko Ludwig, and Marcelo Fantinato, editors, *Service-Oriented Computing - ICSOC 2010 International Workshops, PAASC, WE-SOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers*, volume 6568 of *Lecture Notes in Computer Science*, pages 27–38, 2010. (Cited on pages 65, 68, 198, and 199.)
- [42] Amal Elgammal, Oktay Türetken, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Root-cause analysis of design-time compliance violations on the basis of property patterns. In Paul P. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *ICSOC*, volume 6470 of *Lecture Notes in Computer Science*, pages 17–31, 2010. (Cited on pages 65, 66, 109, 149, 228, and 370.)
- [43] Dennis Schunselaar Felix Mannhardt, Niek Tax and Eric Verbeek. *Importing CSV files Using the ProM 6.5 Log package*. Eindhoven University of Technology, Den Dolech 2, 5612 AZ Eindhoven P.O. Box 513, 5600 MB Eindhoven The Netherlands, 6.5.121 edition, June 2015. (Cited on page 325.)
- [44] Hajo A. Reijers Wil M. P. van der Aalst Felix Mannhardt, Massimiliano de Leon. Balanced multi-perspective checking of process conformance. *Computing*, 98:1–31, 2016. To Appear. (Cited on pages 10, 43, 183, and 218.)
- [45] François Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*, 5:1531–1555, 2004. (Cited on pages 265 and 338.)
- [46] Daniel Fötsch, Elke Pulvermüller, and Wilhelm Rossak. Modeling and verifying workflow-based regulations. In Régine Laleau and Michel

- Lemoine, editors, *Proceedings of the CAISE\*06 Workshop on Regulations Modelling and their Validation and Verification ReMo2V '06, Luxemburg, June 5-9, 2006*, volume 241 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. (Cited on pages 9 and 149.)
- [47] Ning Ge, Marc Pantel, and Xavier Crégut. Formal specification and verification of task time constraints for real-time systems. In *ISoLA (2)*, volume 7610 of *LNCS*, pages 143–157. Springer, 2012. (Cited on page 193.)
- [48] Aditya Ghose and George Koliadis. Auditing business process compliance. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, volume 4749 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2007. (Cited on pages 9 and 149.)
- [49] Christopher Giblin, Alice Y. Liu, Samuel Müller, Birgit Pfitzmann, and Xin Zhou. Regulations expressed as logical models (REALM). In Marie-Francine Moens and Peter Spyns, editors, *Legal Knowledge and Information Systems - JURIX 2005: The Eighteenth Annual Conference on Legal Knowledge and Information Systems, Brussels, Belgium, 8-10 December 2005*, volume 134 of *Frontiers in Artificial Intelligence and Applications*, pages 37–48. IOS Press, 2005. (Cited on pages 8, 9, 65, 66, 109, and 370.)
- [50] 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. (Cited on page 10.)
- [51] Stijn Goedertier, Jochen De Weerd, David Martens, Jan Vanthienen, and Bart Baesens. Process discovery in event logs: An application in the telecom industry. *Appl. Soft Comput.*, 11(2):1697–1710, 2011. (Cited on page 355.)
- [52] Guido Governatori, Jörg Hoffmann, Shazia Wasim Sadiq, and Ingo Weber. Detecting regulatory compliance for business process models through semantic annotations. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, volume 17 of *Lecture Notes in Business Information Processing*, pages 5–17. Springer, 2008. (Cited on pages 9 and 148.)



- [53] Guido Governatori, Zoran Milosevic, and Shazia Wasim Sadiq. Compliance checking between business processes and business contracts. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China*, pages 221–232. IEEE Computer Society, 2006. (Cited on pages 9 and 24.)
- [54] Vladimir Gromov. Diagnostics in compliance checking. Master’s thesis, Eindhoven University of technology, January 2014. (Cited on pages 264, 273, 277, and 278.)
- [55] Volker Gruhn and Ralf Laue. Patterns for timed property specifications. *Electr. Notes Theor. Comput. Sci.*, 153(2):117–133, 2006. (Cited on pages 65, 66, 109, and 370.)
- [56] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In Alonso et al. [13], pages 328–343. (Cited on pages 266 and 270.)
- [57] Christian W. Günther and Eric Verbeek. Extendible event stream (xes) standard, March 2014. (Cited on pages 22 and 325.)
- [58] Kosmas Hatzidimitris. Using visual analytics for conformance checking and compliance rules. Master’s thesis, Eindhoven University of Technology (TU/e), 2013. (Cited on page 279.)
- [59] Airlangga Adi Hermawan, Massimiliano de Leoni, and B Skoric. Context analysis of business processes based on event logs. Master’s thesis, Eindhoven University of Technology (TU/e), 2013. (Cited on page 279.)
- [60] Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In Kohei Honda and Alan Mycroft, editors, *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010.*, volume 69 of *EPTCS*, pages 59–73, 2010. (Cited on page 70.)
- [61] Anne-Maria Holma, Anu Bask, and Katri Kauppi. Ensuring corporate travel compliance: Control vs. commitment strategies. *Tourism Management*, 51:60–74, 2015. (Cited on page 354.)
- [62] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29, No 2:119–127, 1980. (Cited on page 263.)

- [63] Uzay Kaymak, Ronny Mans, Tim van de Steeg, and Meghan Dierks. On process mining in health care. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, SMC 2012, Seoul, Korea (South), October 14-17, 2012*, pages 1859–1864. IEEE, 2012. (Cited on page 356.)
- [64] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. On enabling data-aware compliance checking of business process models. In *ER*, volume 6412 of *LNCS*, pages 332–346. Springer, 2010. (Cited on page 228.)
- [65] George Koliadis, Nimit Desai, Nanjangud C. Narendra, and Aditya K. Ghose. Analyst-mediated contextualization of regulatory policies. In *2010 IEEE International Conference on Services Computing, SCC 2010, Miami, Florida, USA, July 5-10, 2010*, pages 281–288. IEEE Computer Society, 2010. (Cited on page 148.)
- [66] Sascha Konrad and Betty H. C. Cheng. Facilitating the construction of specification pattern-based properties. In *13th IEEE International Conference on Requirements Engineering (RE 2005), 29 August - 2 September 2005, Paris, France*, pages 329–338. IEEE Computer Society, 2005. (Cited on pages 9, 136, and 149.)
- [67] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. (Cited on page 193.)
- [68] Andreas Lanz, Barbara Weber, and Manfred Reichert. Workflow time patterns for process-aware information systems. In Ilia Bider, Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling - 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010, Hammamet, Tunisia, June 7-8, 2010. Proceedings*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, 2010. (Cited on pages 65, 67, and 156.)
- [69] Andreas Lanz, Barbara Weber, and Manfred Reichert. Time patterns for process-aware information systems. *Requir. Eng.*, 19(2):113–141, 2014. (Cited on pages 65, 67, and 156.)
- [70] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Process and deviation exploration with inductive visual miner. In Lior Limonad

- and Barbara Weber, editors, *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014)*, Eindhoven, The Netherlands, September 10, 2014., volume 1295 of *CEUR Workshop Proceedings*, page 46. CEUR-WS.org, 2014. (Cited on page 266.)
- [71] Martin Lehnert, Alexander Linhart, and Maximilian Röglinger. Chopping down trees vs. sharpening the axe - balancing the development of BPM capabilities with process improvement. In Sadiq et al. [115], pages 151–167. (Cited on page 355.)
- [72] Blume LHK, van Weert NJHW, and Delnoij DMJ. How to manage external demands in hospitals: the case of atrium mc. *Healthcare*, 3:157–159, 2015. (Cited on page 353.)
- [73] Hongchen Li and Yun Yang. Verification of temporal constraints for concurrent workflows. In Yu et al. [153], pages 804–813. (Cited on pages 65, 67, and 156.)
- [74] Hongchen Li and Yun Yang. Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications*, 4(2):124–142, 2005. (Cited on page 192.)
- [75] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2nd edition edition, 2002. (Cited on page 162.)
- [76] Niels Lohmann, Minseok Song, and Petia Wohed, editors. *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, volume 171 of *Lecture Notes in Business Information Processing*. Springer, 2014. (Cited on pages 477 and 478.)
- [77] Ruopeng Lu, Shazia Wasim Sadiq, and Guido Governatori. Compliance aware business process design. In Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers*, volume 4928 of *Lecture Notes in Computer Science*, pages 120–131. Springer, 2007. (Cited on page 9.)
- [78] Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking based on partially ordered event data. In Fabiana Fournier and

- Jan Mendling, editors, *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, volume 202 of *Lecture Notes in Business Information Processing*, pages 75–88. Springer, 2014. (Cited on page 21.)
- [79] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring business process compliance using compliance rule graphs. In Robert Meersman, Tharam S. Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng Chin Ooi, Ernesto Damiani, Douglas C. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh K. Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part I*, volume 7044 of *Lecture Notes in Computer Science*, pages 82–99. Springer, 2011. (Cited on pages 149 and 150.)
- [80] Fabrizio Maria Maggi, Marco Montali, and Wil M. P. van der Aalst. An operational decision support framework for monitoring business constraints. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7212 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2012. (Cited on page 150.)
- [81] 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 Rinderle-Ma et al. [103], pages 132–147. (Cited on pages 10, 149, and 150.)
- [82] Ronny Mans, Hajo A. Reijers, Hans Berends, Wasana Bandara, and Rogier Prince. Business process mining success. In *21st European Conference on Information Systems, ECIS 2013, Utrecht, The Netherlands, June 5-8, 2013*, page 89, 2013. (Cited on page 356.)
- [83] 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. (Cited on pages 70 and 193.)

- [84] 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. (Cited on pages 148 and 150.)
- [85] 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. (Cited on pages 10, 149, 150, and 267.)
- [86] Jorge Munoz-Gama and Josep Carmona. A fresh look at precision in process conformance. In Richard Hull, Jan Mendling, and Stefan Tai, editors, *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer, 2010. (Cited on page 10.)
- [87] Jorge Munoz-Gama and Josep Carmona. Enhancing precision in process conformance: Stability, confidence and severity. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France* [2], pages 184–191. (Cited on page 10.)
- [88] Cosmina Cristina Niculae. Time patterns in workflow management systems. Technical Report BPM-11-04,, BPM Center Report, BPMcenter.org, 2011. (Cited on pages 65, 67, and 156.)
- [89] Edward Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Trans. Knowl. Data Eng.*, 15(1):57–69, 2003. (Cited on page 260.)
- [90] Hanchuan Peng, Fuhui Long, and Chris H. Q. Ding. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1226–1238, 2005. (Cited on pages 265 and 338.)
- [91] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. (Cited on page 24.)

- [92] Amir Pnueli. Two decades of temporal logic: Achievements and challenges (abstract). In *38th Annual Symposium on Foundations of Computer Science, FOCs '97, Miami Beach, Florida, USA, October 19-22, 1997*, page 78. IEEE Computer Society, 1997. (Cited on page 9.)
- [93] Heinz Pozewaunig, Johann Eder, and Walter Liebhart. ePERT: Extending PERT for workflow management system. In *ADBIS*, pages 217–224. Nevsky Dialect, 1997. (Cited on page 192.)
- [94] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. (Cited on page 264.)
- [95] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. (Cited on page 264.)
- [96] Lori A. Clarke Rachel L. Smith, George S. Avrunin. From natural language requirements to rigorous property specifications. In *Monterey Workshop 2003 (SEES 2003)*, number UM-CS-2004-019, pages 40–46. Chicago, IL, September 2003. (Cited on page 136.)
- [97] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. Where did I misbehave? diagnostic information in compliance checking. In Barros et al. [17], pages 262–278. (Cited on pages 52, 151, and 267.)
- [98] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. Supporting domain experts to select and configure precise compliance rules. In Lohmann et al. [76], pages 498–512. (Cited on pages 52, 149, and 367.)
- [99] Elham Ramezani, Dirk Fahland, Jan Martijn E. M. van der Werf, and Peter Mattheis. Separating compliance management and business process management. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part II*, volume 100 of *Lecture Notes in Business Information Processing*, pages 459–464. Springer, 2011. (Cited on page 8.)
- [100] Andre D. Read, Ronald J. West, and Brendan P. Kelaher. Using compliance data to improve marine protected area management. *Marine Policy*, 60:119–127, 2015. (Cited on page 353.)
- [101] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012. (Cited on page 70.)

- [102] Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling—an academic dream or the future for bpm? In Daniel et al. [30], pages 307–322. (Cited on pages 70 and 149.)
- [103] Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors. *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*. Springer, 2011. (Cited on pages 475 and 484.)
- [104] Carlos Rodríguez, Patrícia Silveira, Florian Daniel, and Fabio Casati. Analyzing compliance of service-based business processes for root-cause analysis and prediction. In Daniel and Facca [29], pages 277–288. (Cited on pages 10 and 278.)
- [105] Andreas Rogge-Solti. *Probabilistic Estimation Of Unobserved Process Events*. PhD thesis, Hasso Plattner Institute, University of Potsdam, January 2014. (Cited on pages 161 and 162.)
- [106] Andreas Rogge-Solti and Gjergji Kasneci. Temporal anomaly detection in business processes. In Sadiq et al. [115], pages 234–249. (Cited on page 193.)
- [107] Andreas Rogge-Solti, Wil M. P. van der Aalst, and Mathias Weske. Discovering stochastic petri nets with arbitrary delay distributions from event logs. In Lohmann et al. [76], pages 15–27. (Cited on pages 161 and 162.)
- [108] Michael Rosemann and Wil M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007. (Cited on page 138.)
- [109] Mohsen Rouached, Walid Gaaloul, Wil M. P. van der Aalst, Sami Bhiri, and Claude Godart. Erratum: Web service mining and verification of properties: An approach based on event calculus. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *LNCS*, page 2. Springer, 2006. (Cited on page 267.)
- [110] Anne Rozinat, Ivo S. M. de Jong, Christian W. Günther, and Wil M. P. van der Aalst. Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 39(4):474–479, 2009. (Cited on page 356.)

- [111] 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. (Cited on pages 10, 151, and 267.)
- [112] Ksenia Ryndina, Jochen Malte Küster, and Harald Gall. Consistency of business process models and object life cycles. In *MoDELS Workshops*, volume 4364 of *LNCS*, pages 80–90. Springer, 2006. (Cited on page 227.)
- [113] Shazia Wasim Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In Alonso et al. [13], pages 149–164. (Cited on pages 9 and 148.)
- [114] Shazia Wasim Sadiq, Maria E. Orłowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, 2005. (Cited on page 9.)
- [115] Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors. *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*. Springer, 2014. (Cited on pages 474 and 478.)
- [116] Pierangela Samarati and Sabrina De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000]*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. Springer, 2000. (Cited on pages 65, 68, 198, and 199.)
- [117] Daniel Schleicher, Stefan Grohe, Frank Leymann, Patrick Schneider, David Schumm, and Tamara Wolf. An approach to combine data-related and control-flow-related compliance rules. In Kwei-Jay Lin, Christian Huemer, M. Brian Blake, and Boualem Benatallah, editors, *2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011, Irvine, CA, USA, December 12-14, 2011*, pages 1–8. IEEE Computer Society, 2011. (Cited on pages 65, 66, 109, and 370.)
- [118] David Schumm, Frank Leymann, Zhilei Ma, Thorsten Scheibler, and Steve Strauch. Integrating compliance into business processes. In Matthias Schumann, Lutz M. Kolbe, Michael H. Breitner, and Arne



- Frerichs, editors, *Multikonferenz Wirtschaftsinformatik, MKWI 2010, Göttingen, 23.-25.2.2010, Proceedings*, pages 2125–2137. Universitätsverlag Göttingen, 2010. (Cited on pages 65, 66, 109, and 370.)
- [119] David Schumm, Oktay Türetken, Natallia Kokash, Amal Elgammal, Frank Leymann, and Willem-Jan van den Heuvel. Business process compliance through reusable units of compliant processes. In Daniel and Facca [29], pages 325–337. (Cited on pages 65, 66, 68, 109, 149, 198, 199, 228, and 370.)
- [120] Sung Y. Shin and José Carlos Maldonado, editors. *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*. ACM, 2013. (Cited on pages 466 and 469.)
- [121] Natalia Sidorova, Christian Stahl, and Nikola Trcka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.*, 36(7):1026–1043, 2011. (Cited on page 27.)
- [122] Patrícia Silveira, Carlos Rodríguez, Fabio Casati, Florian Daniel, Vincenzo D'Andrea, Claire Worledge, and Zouhair Taheri. On the design of compliance governance dashboards for effective compliance and audit management. In Asit Dan, Frederic Gittler, and Farouk Toumani, editors, *ICSOC/ServiceWave Workshops*, volume 6275 of *LNCS*, pages 208–217, 2009. (Cited on pages 10 and 278.)
- [123] Rachel L. Smith, George S. Avrunin, Lori A. Clarke, and Leon J. Osterweil. Propel: an approach supporting property elucidation. In *ICSE*, pages 11–21. ACM, 2002. (Cited on pages 9, 136, and 149.)
- [124] M. Song and W.M.P. van der Aalst. Towards comprehensive support for organizational mining. *Decision Support Systems*, 46:300–317, 2008. (Cited on page 355.)
- [125] K. Srinivasana, S. Muthub, S.R. Devadasanc, and C. Sugumarand. Enhancing effectiveness of shell and tube heat exchanger through six sigma dmaic phases. *Procedia Engineering*, 97:2064–2071, 2014. (Cited on page 352.)
- [126] Mariëlle Stoelinga and Ralf Pinger, editors. *Formal Methods for Industrial Critical Systems - 17th International Workshop, FMICS 2012, Paris*,

- France, August 27-28, 2012. *Proceedings*, volume 7437 of *Lecture Notes in Computer Science*. Springer, 2012. (Cited on page 465.)
- [127] Jakub Stolfa, Martin Kopka, Svatopluk Stolfa, Ondrej Kobersky, and Václav Snásel. An application of process mining to invoice verification process in SAP. In Ajith Abraham, Pavel Krömer, and Václav Snásel, editors, *Innovations in Bio-inspired Computing and Applications - Proceedings of the 4th International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2013, August 22 -24, 2013 - Ostrava, Czech Republic*, volume 237 of *Advances in Intelligent Systems and Computing*, pages 61–74. Springer, 2013. (Cited on page 357.)
- [128] Marco Stuit and Hans Wortmann. Discovery and analysis of e-mail-driven business processes. *Inf. Syst.*, 37(2):142–168, 2012. (Cited on page 356.)
- [129] Suriadi Suriadi, Ronny Mans, Moe Thandar Wynn, Andrew Partington, and Jonathan Karnon. Measuring patient flow variations: A cross-organisational process mining approach. In Chun Ouyang and Jae-Yoon Jung, editors, *Asia Pacific Business Process Management - Second Asia Pacific Conference, AP-BPM 2014, Brisbane, QLD, Australia, July 3-4, 2014. Proceedings*, volume 181 of *Lecture Notes in Business Information Processing*, pages 43–58. Springer, 2014. (Cited on page 356.)
- [130] Elham Ramezani Taghiabadi, Dirk Fahland, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Diagnostic information for compliance checking of temporal compliance requirements. In Camille Salinesi, Moira C. Norrie, and Oscar Pastor, editors, *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, volume 7908 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2013. (Cited on pages 52, 191, 264, and 267.)
- [131] Elham Ramezani Taghiabadi, Vladimir Gromov, Dirk Fahland, and Wil M. P. van der Aalst. Compliance checking of data-aware and resource-aware compliance requirements. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzocrea, and Timos Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, volume 8841 of *Lecture Notes in Computer Science*, pages 237–257. Springer, 2014. (Cited on pages 52 and 227.)

- [132] Oktay Türetken, Amal Elgammal, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Enforcing compliance on business processes through the use of patterns. In *ECIS*, 2011. (Cited on pages 9, 65, 149, 198, 199, and 228.)
- [133] Wil M. P. van der Aalst. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013(ID 507984):37, 2013. (Cited on page 70.)
- [134] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. (Cited on pages 3, 89, 115, 116, 151, 228, and 261.)
- [135] 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. Rev.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. (Cited on pages 10, 36, 37, 38, 42, 43, 115, 116, 151, and 267.)
- [136] Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2005. (Cited on pages 10, 149, and 150.)
- [137] Wil M. P. van der Aalst, Alexander Dreiling, Florian Gottschalk, Michael Rosemann, and Monique H. Jansen-Vullers. Configurable process models as a basis for reference modeling. In *BPM Workshops*, volume 3812, pages 512–518, 2005. (Cited on page 138.)
- [138] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. (Cited on page 70.)
- [139] Wil M. P. van der Aalst, Hajo A. Reijers, A. J. M. M. Weijters, Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, Minseok Song, and H. M.

- W. (Eric) Verbeek. Business process mining: An industrial application. *Inf. Syst.*, 32(5):713–732, 2007. (Cited on page 355.)
- [140] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475, 2011. (Cited on page 355.)
- [141] Wil M. P. van der Aalst and Christian Stahl. *Modeling Business Processes - A Petri Net-Oriented Approach*. Cooperative Information Systems series. MIT Press, 2011. (Cited on pages 24 and 27.)
- [142] Wil M. P. van der Aalst, Kees M. van Hee, Jan Martijn E. M. van der Werf, Akhil Kumar, and Marc Verdonk. Conceptual model for online auditing. *Decision Support Systems*, 50(3):636–647, 2011. (Cited on pages 65, 66, 109, and 370.)
- [143] Maikel L. van Eck, Natalia Sidorova, and Wil M. P. van der Aalst. Kpi-based activity planning for people working in flexible processes. In Janis Grabis and Kurt Sandkuhl, editors, *Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering co-located with 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015), Stockholm, Sweden, June 10th, 2015.*, volume 1367 of *CEUR Workshop Proceedings*, pages 97–104. CEUR-WS.org, 2015. (Cited on page 147.)
- [144] Jan vom Brocke and Michael Rosemann, editors. *Handbook on Business Process Management 2, Strategic Alignment, Governance, People and Culture, 2nd Ed*, International Handbooks on Information Systems. Springer, 2015. (Cited on page 8.)
- [145] Jochen De Weerd, Manu De Backer, Jan Vanthienen, and Bart Baesens. A robust f-measure for evaluating discovered process models. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France* [2], pages 148–155. (Cited on page 10.)
- [146] Matthias Weidlich, Artem Polyvyanyy, Nirmal Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011. (Cited on page 357.)

- [147] Matthias Weidlich, Holger Ziekow, Jan Mendling, Oliver Günther, Mathias Weske, and Nirmal Desai. Event-based monitoring of process execution violations. In Rinderle-Ma et al. [103], pages 182–198. (Cited on page 150.)
- [148] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (FHM). In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France* [2], pages 310–317. (Cited on page 266.)
- [149] Michael Westergaard and Fabrizio Maria Maggi. Looking into the future. using timed automata to provide a priori advice about timed declarative process models. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I*, volume 7565 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2012. (Cited on pages 10, 65, 67, 156, and 193.)
- [150] Glynn Winskel. Petri nets, morphisms and compositionality. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets, Espoo, Finland in June 1985, selected papers*, volume 222 of *Lecture Notes in Computer Science*, pages 453–477. Springer, 1985. (Cited on page 147.)
- [151] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008. (Cited on page 263.)
- [152] Xindong Wu, Chengqi Zhang, and Shichao Zhang. Efficient mining of both positive and negative association rules. *ACM Trans. Inf. Syst.*, 22(3):381–405, 2004. (Cited on pages 255, 259, 278, and 359.)
- [153] Jeffrey Xu Yu, Xuemin Lin, Hongjun Lu, and Yanchun Zhang, editors. *Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference, APWeb 2004, Hangzhou, China, April 14-17, 2004, Proceedings*,

- volume 3007 of *Lecture Notes in Computer Science*. Springer, 2004. (Cited on pages 65, 67, 156, and 474.)
- [154] Lei Yu and Huan Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224, 2004. (Cited on pages 265 and 338.)



# Acknowledgments

My interest in pursuing a PhD dates back to the time when I left Iran in 2008 and started my master studies in Germany. Later when I started working as a researcher, I got to learn about process mining and became interested in the topic. Among the papers I read, the papers that most delighted me had Wil's name on them. I met Dirk Fahland in a conference who was working in the group of Wil van der Aalst. I asked him to bring me in contact with Wil. One thing led to another and I received an offer to do a PhD in process mining in TU/e. Although I truly valued the opportunity to do a PhD under supervision of Wil and Dirk, when I joined AIS group in 2012, I was not sure whether I would feel comfortable in the group or not. I studied business and did not really share the same background knowledge with the group. Ever since, they have given me support and resources to learn and explore. And somewhere on the way the doubts of doing a PhD vanished.

Throughout these years, Wil has enlightened me through his wealth of knowledge and deep intuition. He has been a role model for me and his enthusiasm and passion to strive for better have inspired me. He has always challenged me and encouraged me to grow as an independent researcher. His precise and focused view on topics, while he has the bigger picture in mind, taught me how important it is to work an idea to the level that is mature, precise and complete. Only then others can see the added value and the true contribution of your work.

I would not have started or finished this thesis without the help of Dirk. I had the most challenging and fundamental discussions about my research with him. He instantly understood my ideas and helped me to structure my thoughts. He always provided new and interesting perspectives to my thoughts. He taught me how to form and present my ideas to an audience. I would not have been able to demonstrate the feasibility of my research in practice without his support.



Thank you for your trust and patience!

I would like to thank my reading committee members Hajo Reijers, Shazia Sadiq, Sandro Etalle, Stefanie Rinderle-Ma and Michael zur Muehlen for their time, interest and helpful comments. Especially, I would like to thank Hajo for his guidance and help. I very much enjoyed my many discussions and talks with him and learning about his multi-perspective view to the world.

I would like to thank Peter Matheis for encouraging me to pursue the PhD and providing me the resources to realize it. I am gratefully indebted to Christian Stahl for his friendship and help during my PhD project. I would like to thank Eric Verbeek for being there always with an open-door policy to help and clarify any question or issues with ProM. I would like to thank Massimiliano de Leoni and Arya Adriansyah for their outstanding works that formed the basis for my research. I would like to thank Valdimir Gromov for his immediate responses and maintaining several ProM Plugins even long after he left the research community.

I am grateful to the team in UWV who allowed me to test my ideas in a real challenge. I owe much to Marcus Dees and Vincent Jalink. This thesis would not have a successful evaluation without their help and devotion.

Several people at TU/e have extended their help and support in the project. I thank Eduardo, Maikel Leemans, Xixi, Sander, Mohsen, Felix, Ronny, Boudewijn and Maikel van Eck. I would like to thank Anne Rozinat who extended her support during my PhD and after that in numerous ways. Special thanks go to Dennis and Joos for always being there for me. They made the PhD journey for me easier and more fun in many different ways. I enjoyed very much our scientific and not-so-scientific discussions. I would like to thank Maikel, Marie and Guangming for creating a nice atmosphere in the office. Many thanks are due to Riet and Ine who have been wonderful and took care of all the administrative things with a big smile.

I would like to thank my colleagues at KPMG for their understanding and supporting me during the last months of the thesis writing.

I thank my friends who have always been a source of energy. They provided a warm and pleasant environment for me. My friends who never stopped asking me for a little bit of distraction and relaxation and understood that I often had to work. Thank you! Special thanks goes to Mahnaz for the most leveraging discussions and hints how to organize a hectic period of life.

I have had the great fortune to cross paths with many dedicated teachers and benefit from their wisdom, encouragement and friendship. Of them, one had a profound impact on me and I wish he was still among us to see this thesis. I owe my thanks to Mahyar Khalili for his inspiring words and advice which gave

me the courage to strive for better.

My deepest gratitude goes to my beloved parents Zari and Mahmoud for their sacrifices, unconditional love and support throughout my life. They have imbibed in me the importance of education and provided me all that they can towards realizing this. Thank you! I would like to thank my brother Nima. I have always looked up to you as a source of inspiration and a role-model. I would like to thank my sister Nasim for the bundles of joy she brings in my life. I would like to thank her for being caring in so many different ways. Thank you for taking care of everything I could not take care of in the last years. I would like to thank my dear Shirin for believing in me and helping me in surprises that life gives us.

Thank you my dear Amir, for your interest in knowing about my research. I enjoyed very much our discussions and they helped me in better articulation of my thoughts in the thesis. Your interest and support have been a constant source of energy. Most of all thank you for bringing the needed peace and happiness into my life during the final stages of my PhD.

Elham  
Eindhoven, December 2016



# Curriculum Vitae

Elham Ramezani was born in Esfahan, Iran in 1979. She obtained her B.Sc. degree with distinction in business management from University of Tehran, Iran in 2002. She worked for six years in Iran Khodro Industrial Group (IKCO) as a business analyst. In 2008, she moved to Germany to pursue her Master's degree. She received her M.Sc. degree in 2009 with distinction in business consulting from Business Informatics (Wirtschaftsinformatik) department of Hochschule Furtwangen University, Germany. During her master studies she joined IDS-Scheer AG, Germany for an internship. She did her master thesis in IKOR Financials GmbH, Hamburg, Germany. Her master thesis titled *Developing a guideline for outsourcing business intelligence and data warehousing projects in an offshore environment* was awarded for the best thesis of the year (2009) by Hochschule Furtwangen University, Germany. She worked in the same university as a lecturer and research assistant for two years. Following her research interests, in 2012, she moved to The Netherlands to pursue a Doctoral degree in the Information Systems group of Eindhoven University of Technology. During her Ph.D., she developed several techniques for analyzing and improving process compliance. Her research interests include process mining, business process management and data mining. The result of her efforts on these topics are described in this dissertation. Since 2016, she is employed at KPMG, The Netherlands.

## List of Publications

Elham Ramezani Taghiabadi has the following publications:

- Elham Ramezani Taghiabadi, Vladimir Gromov, Dirk Fahland, Wil M. P. van der Aalst: **Compliance Checking of Data-Aware and Resource-Aware Compliance Requirements**. OTM Conferences 2014: 237-257
- Elham Ramezani, Dirk Fahland, Wil M. P. van der Aalst: **Supporting Domain Experts to Select and Configure Precise Compliance Rules**. Business Process Management Workshops 2013: 498-512
- Elham Ramezani Taghiabadi, Dirk Fahland, Boudewijn F. van Dongen, Wil M. P. van der Aalst: **Diagnostic Information for Compliance Checking of Temporal Compliance Requirements**. CAiSE 2013: 304-320
- Elham Ramezani, Natalia Sidorova, Christian Stahl: **Interval Soundness of Resource-Constrained Workflow Nets: Decidability and Repair**. FSEN 2013: 150-167
- Elham Ramezani, Dirk Fahland, Wil M. P. van der Aalst: **Where Did I Misbehave? Diagnostic Information in Compliance Checking**. BPM 2012: 262-278
- Elham Ramezani, Dirk Fahland, Jan Martijn E. M. van der Werf, Peter Mattheis: **Separating Compliance Management and Business Process Management**. Business Process Management Workshops (2) 2011: 459-464

# SIKS dissertations

## 2009

- 2009-01 Rasa Jurgelenaite (RUN) Symmetric Causal Independence Models
- 2009-02 Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques
- 2009-03 Hans Stol (UvT) A Framework for Evidence-based Policy Making Using IT
- 2009-04 Josephine Nabukenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering
- 2009-05 Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
- 2009-06 Muhammad Subianto (UU) Understanding Classification
- 2009-07 Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion
- 2009-08 Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments
- 2009-09 Benjamin Kanagwa (RUN) Design, Discovery and Construction of Service-oriented Systems
- 2009-10 Jan Wielemaker (UVA) Logic programming for knowledge-intensive interactive applications
- 2009-11 Alexander Boer (UVA) Legal Theory, Sources of Law & the Semantic Web
- 2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin) Operating Guidelines for Services
- 2009-13 Steven de Jong (UM) Fairness in Multi-Agent Systems
- 2009-14 Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA) Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT) New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT) Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI) Armada, An Evolving Database System

2009-19 Valentin Robu (CWI) Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU) Adjustable Autonomy: Controlling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM) Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT) Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU) Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA) Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI) "RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU) An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU) Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT) Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT) Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI) Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA) A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiat Truong (UT) How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN) Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context  
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) Service Substitution – A Behavioral Approach Based on Petri Nets  
2009-40 Stephan Raaijmakers (UvT) Multinomial Language Learning: Investigations into the Geometry of Language  
2009-41 Igor Berezhnyy (UvT) Digital Analysis of Paintings  
2009-42 Toine Bogers (UvT) Recommender Systems for Social Bookmarking  
2009-43 Virginia Nunes Leal Franqueira (UT) Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients  
2009-44 Roberto Santana Tapia (UT) Assessing Business-IT Alignment in Networked Organizations  
2009-45 Jilles Vreeken (UU) Making Pattern Mining Useful  
2009-46 Loredana Afanasiev (UvA) Querying XML: Benchmarks and Recursion

## 2010

2010-01 Matthijs van Leeuwen (UU) Patterns that Matter  
2010-02 Ingo Wassink (UT) Work flows in Life Science  
2010-03 Joost Geurts (CWI) A Document Engineering Model and Processing Framework for Multimedia documents  
2010-04 Olga Kulyk (UT) Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments  
2010-05 Claudia Hauff (UT) Predicting the Effectiveness of Queries and Retrieval Systems  
2010-06 Sander Bakkes (UvT) Rapid Adaptation of Video Game AI  
2010-07 Wim Fikkert (UT) Gesture interaction at a Distance  
2010-08 Krzysztof Siewicz (UL) Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments  
2010-09 Hugo Kielman (UL) A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging  
2010-10 Rebecca Ong (UL) Mobile Communication and Protection of Children  
2010-11 Adriaan Ter Mors (TUD) The world according to MARP: Multi-Agent Route Planning  
2010-12 Susan van den Braak (UU) Sensemaking software for crime analysis  
2010-13 Gianluigi Folino (RUN) High Performance Data Mining using Bio-inspired techniques



2010-14 Sander van Splunter (VU) Automated Web Service Reconfiguration  
2010-15 Lianne Bodenstaff (UT) Managing Dependency Relations in Inter-Organizational Models  
2010-16 Sicco Verwer (TUD) Efficient Identification of Timed Automata, theory and practice  
2010-17 Spyros Kotoulas (VU) Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications  
2010-18 Charlotte Gerritsen (VU) Caught in the Act: Investigating Crime by Agent-Based Simulation  
2010-19 Henriette Cramer (UvA) People's Responses to Autonomous and Adaptive Systems  
2010-20 Ivo Swartjes (UT) Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative  
2010-21 Harold van Heerde (UT) Privacy-aware data management by means of data degradation  
2010-22 Michiel Hildebrand (CWI) End-user Support for Access to Heterogeneous Linked Data  
2010-23 Bas Steunebrink (UU) The Logical Structure of Emotions  
2010-24 Dmytro Tykhonov (TUD) Designing Generic and Efficient Negotiation Strategies  
2010-25 Zulfiqar Ali Memon (VU) Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective  
2010-26 Ying Zhang (CWI) XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines  
2010-27 Marten Voulon (UL) Automatisch contracteren  
2010-28 Arne Koopman (UU) Characteristic Relational Patterns  
2010-29 Stratos Idreos (CWI) Database Cracking: Towards Auto-tuning Database Kernels  
2010-30 Marieke van Erp (UvT) Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval  
2010-31 Victor de Boer (UVA) Ontology Enrichment from Heterogeneous Sources on the Web  
2010-32 Marcel Hiel (UvT) An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems  
2010-33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval  
2010-34 Teduh Dirgahayu (UT) Interaction Design in Service Compositions  
2010-35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU) Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification  
2010-37 Niels Lohmann (TUE) Correctness of services and their composition  
2010-38 Dirk Fahland (TUE) From Scenarios to components  
2010-39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents  
2010-40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web  
2010-41 Guillaume Chaslot (UM) Monte-Carlo Tree Search  
2010-42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach  
2010-43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies  
2010-44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain  
2010-45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services  
2010-46 Vincent Pijpers (VU) e3alignment: Exploring Inter-Organizational Business-ICT Alignment  
2010-47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples  
2010-48 Withdrawn  
2010-49 Jahn-Takeshi Saito (UM) Solving difficult game positions  
2010-50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives  
2010-51 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources  
2010-52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention  
2010-53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

## **2011**

2011-01 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models  
2011-02 Nick Tinnemeier (UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU) Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU) Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT) Cloud Content Contention

2011-11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM) Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM) Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU) The Mind 's Eye on Personal Profiles

2011-20 Qing Gu (VU) Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA) System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU) Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein (UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE) Discrimination-aware Classification

2011-30 Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU) Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaïke Harbers (UU) Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU) Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM) Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU) Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU) Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT) Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU) Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU) Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT) Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT) Statistical Language Models for Alternative Sequence Selection  
2011-46 Beibei Hu (TUD) Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work  
2011-47 Azizi Bin Ab Aziz (VU) Exploring Computational Models for Intelligent Support of Persons with Depression  
2011-48 Mark Ter Maat (UT) Response Selection and Turn-taking for a Sensitive Artificial Listening Agent  
2011-49 Andreea Niculescu (UT) Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

## **2012**

2012-01 Terry Kakeeto (UvT) Relationship Marketing for SMEs in Uganda  
2012-02 Muhammad Umair (VU) Adaptivity, emotion, and Rationality in Human and Ambient Agent Models  
2012-03 Adam Vanya (VU) Supporting Architecture Evolution by Mining Software Repositories  
2012-04 Jurriaan Souer (UU) Development of Content Management System-based Web Applications  
2012-05 Marijn Plomp (UU) Maturing Interorganisational Information Systems  
2012-06 Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks  
2012-07 Rianne van Lambalgen (VU) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions  
2012-08 Gerben de Vries (UVA) Kernel Methods for Vessel Trajectories  
2012-09 Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms  
2012-10 David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment  
2012-11 J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Pre-processing, Discovery, and Diagnostics  
2012-12 Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems  
2012-13 Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions  
2012-14 Evgeny Knutov (TUE) Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU) Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU) Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU) Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE) What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT) Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA) Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT) Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT) Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT) Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD) Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU) Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes  
2012-37 Agnes Nakakawa (RUN) A Collaboration Process for Enterprise Architecture Creation  
2012-38 Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms  
2012-39 Hassan Fatemi (UT) Risk-aware design of value and coordination networks  
2012-40 Agus Gunawan (UvT) Information Access for SMEs in Indonesia  
2012-41 Sebastian Kelle (OU) Game Design Patterns for Learning  
2012-42 Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning  
2012-43 Withdrawn  
2012-44 Anna Tordai (VU) On Combining Alignment Techniques  
2012-45 Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions  
2012-46 Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation  
2012-47 Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior  
2012-48 Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data  
2012-49 Michael Kaisers (UM) Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions  
2012-50 Steven van Kervel (TUD) Ontology driven Enterprise Information Systems Engineering  
2012-51 Jeroen de Jong (TUD) Heuristics in Dynamic Scheduling; a practical framework with a case study in elevator dispatching

## **2013**

2013-01 Viorel Milea (EUR) News Analytics for Financial Decision Support  
2013-02 Erietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing  
2013-03 Szymon Klarman (VU) Reasoning with Contexts in Description Logics  
2013-04 Chetan Yadati (TUD) Coordinating autonomous planning and scheduling  
2013-05 Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves (CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT) Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk (VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige (UvT) A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras (TUD) Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian (VU) Knowledge-driven Migration to Services

2013-13 Mohammad Safiri (UT) Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA) Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM) Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU) Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU) The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT) Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD) Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA) Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT) Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN) Causal Discovery and Logic

2013-23 Patricio de Alencar Silva (UvT) Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM) Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM) Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT) Architectural Support for Dynamic Homecare Service Provisioning



2013-27 Mohammad Huq (UT) Inference-based Framework Managing Data Provenance  
2013-28 Frans van der Sluis (UT) When Complexity becomes Interesting: An Inquiry into the Information eXperience  
2013-29 Iwan de Kok (UT) Listening Heads  
2013-30 Joyce Nakatumba (TUE) Resource-Aware Business Process Management: Analysis and Support  
2013-31 Dinh Khoa Nguyen (UvT) Blueprint Model and Language for Engineering Cloud Applications  
2013-32 Kamakshi Rajagopal (OUN) Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development  
2013-33 Qi Gao (TUD) User Modeling and Personalization in the Microblogging Sphere  
2013-34 Kien Tjin-Kam-Jet (UT) Distributed Deep Web Search  
2013-35 Abdallah El Ali (UvA) Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA)  
2013-36 Than Lam Hoang (TUE) Pattern Mining in Data Streams  
2013-37 Dirk Börner (OUN) Ambient Learning Displays  
2013-38 Eelco den Heijer (VU) Autonomous Evolutionary Art  
2013-39 Joop de Jong (TUD) A Method for Enterprise Ontology based Design of Enterprise Information Systems  
2013-40 Pim Nijssen (UM) Monte-Carlo Tree Search for Multi-Player Games  
2013-41 Jochem Liem (UVA) Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning  
2013-42 Léon Planken (TUD) Algorithms for Simple Temporal Reasoning  
2013-43 Marc Bron (UVA) Exploration and Contextualization through Interaction and Concepts

## **2014**

2014-01 Nicola Barile (UU) Studies in Learning Monotone Models from Data  
2014-02 Fiona Tulyiano (RUN) Combining System Dynamics with a Domain Modeling Method  
2014-03 Sergio Raul Duarte Torres (UT) Information Retrieval for Children: Search Behavior and Solutions  
2014-04 Hanna Jochmann-Mannak (UT) Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijssen (UU) Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU) Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE) Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD) Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT) Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU) Service Value Networks

2014-11 Janneke van der Zwaan (TUD) An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU) Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU) Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD) Language Models With Meta-information

2014-15 Natalya Mogles (VU) Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU) Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU) Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (UVA) Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vinicius Ramos (TUE) Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT) Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Cassidy Clark (TUD) Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UU) Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI) Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU) Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN) New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD) What to Bid and When to Stop

2014-27 Rui Jorge Almeida (EUR) Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

2014-28 Anna Chmielowiec (VU) Decentralized k-Clique Matching

2014-29 Jaap Kabbedijk (UU) Variability in Multi-Tenant Enterprise Software

2014-30 Peter de Cock (UvT) Anticipating Criminal Behaviour

2014-31 Leo van Moergestel (UU) Agent Technology in Agile Multiparallel Manufacturing and Product Support

2014-32 Naser Ayat (UvA) On Entity Resolution in Probabilistic Data

2014-33 Tesfa Tegegne (RUN) Service Discovery in eHealth

2014-34 Christina Manteli (VU) The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

2014-35 Joost van Ooijen (UU) Cognitive Agents in Virtual Worlds: A Middleware Design Approach

2014-36 Joos Buijs (TUE) Flexible Evolutionary Algorithms for Mining Structured Process Models

2014-37 Maral Dadvar (UT) Experts and Machines United Against Cyberbullying

2014-38 Danny Plass-Oude Bos (UT) Making brain-computer interfaces better: improving usability through post-processing.

2014-39 Jasmina Maric (UvT) Web Communities, Immigration, and Social Capital

2014-40 Walter Omona (RUN) A Framework for Knowledge Management Using ICT in Higher Education

2014-41 Frederic Hogenboom (EUR) Automated Detection of Financial Events in News Text

2014-42 Carsten Eijckhof (CWI/TUD) Contextual Multidimensional Relevance Models

2014-43 Kevin Vlaanderen (UU) Supporting Process Improvement using Method Increments

2014-44 Paulien Meesters (UvT) Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

2014-45 Birgit Schmitz (OUN) Mobile Games for Learning: A Pattern-Based Approach

2014-46 Ke Tao (TUD) Social Web Data Analytics: Relevance, Redundancy, Diversity

2014-47 Shangsong Liang (UVA) Fusion and Diversification in Information Retrieval

## 2015

- 2015-01 Niels Netten (UvA) Machine Learning for Relevance of Information in Crisis Response
- 2015-02 Faiza Bukhsh (UvT) Smart auditing: Innovative Compliance Checking in Customs Controls
- 2015-03 Twan van Laarhoven (RUN) Machine learning for network data
- 2015-04 Howard Spoelstra (OUN) Collaborations in Open Learning Environments
- 2015-05 Christoph Bösch (UT) Cryptographically Enforced Search Pattern Hiding
- 2015-06 Farideh Heidari (TUD) Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
- 2015-07 Maria-Hendrike Peetz (UvA) Time-Aware Online Reputation Analysis
- 2015-08 Jie Jiang (TUD) Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
- 2015-09 Randy Klaassen (UT) HCI Perspectives on Behavior Change Support Systems
- 2015-10 Henry Hermans (OUN) OpenU: design of an integrated system to support lifelong learning
- 2015-11 Yongming Luo (TUE) Designing algorithms for big graph datasets: A study of computing bisimulation and joins
- 2015-12 Julie M. Birkholz (VU) Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
- 2015-13 Giuseppe Procaccianti (VU) Energy-Efficient Software
- 2015-14 Bart van Straalen (UT) A cognitive approach to modeling bad news conversations
- 2015-15 Klaas Andries de Graaf (VU) Ontology-based Software Architecture Documentation
- 2015-16 Changyun Wei (UT) Cognitive Coordination for Cooperative Multi-Robot Teamwork
- 2015-17 André van Cleeff (UT) Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
- 2015-18 Holger Pirk (CWI) Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
- 2015-19 Bernardo Tabuenca (OUN) Ubiquitous Technology for Lifelong Learners
- 2015-20 Loïs Vanhée (UU) Using Culture and Values to Support Flexible Coordination

2015-21 Sibren Fetter (OUN) Using Peer-Support to Expand and Stabilize On-line Learning  
2015-22 Zhemin Zhu (UT) Co-occurrence Rate Networks  
2015-23 Luit Gazendam (VU) Cataloguer Support in Cultural Heritage  
2015-24 Richard Berendsen (UVA) Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation  
2015-25 Steven Woudenberg (UU) Bayesian Tools for Early Disease Detection  
2015-26 Alexander Hogenboom (EUR) Sentiment Analysis of Text Guided by Semantics and Structure  
2015-27 Sándor Héman (CWI) Updating compressed column stores  
2015-28 Janet Bagorogoza (TiU) KNOWLEDGE MANAGEMENT AND HIGH PERFORMANCE; The Uganda Financial Institutions Model for HPO  
2015-29 Hendrik Baier (UM) Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains  
2015-30 Kiavash Bahreini (OU) Real-time Multimodal Emotion Recognition in E-Learning  
2015-31 Yakup Koç (TUD) On the robustness of Power Grids  
2015-32 Jerome Gard (UL) Corporate Venture Management in SMEs  
2015-33 Frederik Schadd (TUD) Ontology Mapping with Auxiliary Resources  
2015-34 Victor de Graaf (UT) Gesocial Recommender Systems  
2015-35 Jungxao Xu (TUD) Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

## 2016

2016-01 Syed Saiden Abbas (RUN) Recognition of Shapes by Humans and Machines  
2016-02 Michiel Christiaan Meulendijk (UU) Optimizing medication reviews through decision support: prescribing a better pill to swallow  
2016-03 Maya Sappelli (RUN) Knowledge Work in Context: User Centered Knowledge Worker Support  
2016-04 Laurens Rietveld (VU) Publishing and Consuming Linked Data  
2016-05 Evgeny Sherkhonov (UVA) Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers  
2016-06 Michel Wilson (TUD) Robust scheduling in an uncertain environment  
2016-07 Jeroen de Man (VU) Measuring and modeling negative emotions for virtual training

2016-08 Matje van de Camp (TiU) A Link to the Past: Constructing Historical Social Networks from Unstructured Data

2016-09 Archana Nottamkandath (VU) Trusting Crowdsourced Information on Cultural Artefacts

2016-10 George Karafotias (VUA) Parameter Control for Evolutionary Algorithms

2016-11 Anne Schuth (UVA) Search Engines that Learn from Their Users

2016-12 Max Knobbout (UU) Logics for Modelling and Verifying Normative Multi-Agent Systems

2016-13 Nana Baah Gyan (VU) The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

2016-14 Ravi Khadka (UU) Revisiting Legacy Software System Modernization

2016-15 Steffen Michels (RUN) Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

2016-16 Guangliang Li (UVA) Socially Intelligent Autonomous Agents that Learn from Human Reward

2016-17 Berend Weel (VU) Towards Embodied Evolution of Robot Organisms

2016-18 Albert Meroño Peñuela Refining Statistical Data on the Web

2016-19 Julia Efremova (Tu/e) Mining Social Structures from Genealogical Data

2016-20 Daan Odijk (UVA) Context & Semantics in News & Web Search

2016-21 Alejandro Moreno Céleri (UT) From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

2016-22 Grace Lewis (VU) Software Architecture Strategies for Cyber-Foraging Systems

2016-23 Fei Cai (UVA) Query Auto Completion in Information Retrieval

2016-24 Brend Wanders (UT) Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

2016-25 Julia Kiseleva (TU/e) Using Contextual Information to Understand Searching and Browsing Behavior

2016-26 Dilhan Thilakarathne (VU) In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

2016-27 Wen Li (TUD) Understanding Geo-spatial Information on Social Media

2016-28 Mingxin Zhang (TUD) Large-scale Agent-based Social Simulation - A study on epidemic prediction and control

2016-29 Nicolas Höning (TUD) Peak reduction in decentralised electricity systems -Markets and prices for flexible planning

- 2016-30 Ruud Mattheij (UvT) The Eyes Have It
- 2016-31 Mohammad Khelghati (UT) Deep web content monitoring
- 2016-32 Eelco Vriezekolk (UT) Assessing Telecommunication Service Availability Risks for Crisis Organisations
- 2016-33 Peter Bloem (UVA) Single Sample Statistics, exercises in learning from just one example
- 2016-34 Dennis Schunselaar (TUE) Configurable Process Trees: Elicitation, Analysis, and Enactment
- 2016-35 Elham Ramezani Taghiabadi (TUE) Understanding Non-compliance