

RESEARCH

Julius Holderer

Obstructions in Security-Aware Business Processes

Analysis, Detection, and Handling

OPEN ACCESS




Springer Vieweg

Obstructions in Security-Aware Business Processes

Julius Holderer

Obstructions in Security-Aware Business Processes

Analysis, Detection, and Handling

Julius Holderer 
University of Freiburg
Freiburg im Breisgau, Germany

The present text is the publication of the dissertation for the award of the degree of Doctor of Engineering (Dr.-Ing.) of the Faculty of Engineering of the Albert Ludwigs University of Freiburg im Breisgau with the title “Obstructions in Security-Aware Business Processes – Analysis, Detection, and Handling”.

Place of the dissertation: Faculty of Engineering, University of Freiburg, Germany

Date of the disputation: July 26th, 2021

Dean: Prof. Dr. Rolf Backofen

Reviewers: Prof. Dr. h.c. Günter Müller, Prof. Dr. Josep Carmona, Prof. Dr. Bernd Becker



This publication was supported by the Open Access Publication Fund of the Albert Ludwigs University of Freiburg.

ISBN 978-3-658-38153-0

ISBN 978-3-658-38154-7 (eBook)

<http://doi.org/10.1007/978-3-658-38154-7>

© The Editor(s) (if applicable) and The Author(s) 2022. This book is an open access publication.

Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this book are included in the book’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Responsible Editor: Stefanie Eggert

This Springer Vieweg imprint is published by the registered company Springer Fachmedien Wiesbaden GmbH, part of Springer Nature.

The registered company address is: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Acknowledgements

First and foremost, I want to thank my advisor, Prof. Dr. Dr. h.c. Günter Müller, for continuously supporting, challenging, and mentoring me in a unique and ever-encouraging way throughout all the phases of this dissertation, my doctorate, and beyond. It is hard to summarize all his help and what I learned from him in research and as a person in brief. I feel most thankful for all of it!

Also, I want to thank my second advisor, Prof. Dr. Josep Carmona, for being part of the journey, especially for taking over the task as the second reviewer and examiner. The inspiring collaboration with him, my stay at Barcelona Tech, and the resulting research were crucial for me and my work.

I further want to thank Prof Dr. Bernd Becker for his time and readiness to be the third reviewer and examiner, which was necessary due to the reviewer constellation.

Much appreciation goes to my colleagues at the Institute of Computer Science and Social Studies and the Institute of Sustainable Systems Engineering and to further persons directly or indirectly involved with this work. Many thanks in particular to Adrian, Anja, Arnt, Björn, Christian B., Christian Z., Claus, Farbod, Georg, Markus, Nadine, Thomas S., Thomas K., Tomasz, Rafael, Richard, and Véronique for the support, good discussions, fruitful collaboration, and good times.

Finally, I am deeply grateful for my private environment and its invaluable and unconditional support, patience, and confidence!

Abstract

The growing regulatory pressure on increasingly digitized businesses, for example to combat the growing number of corporate fraud cases, can have an obstructive effect on the execution of automated business processes. Such security-related obstructions occur when the implementation of regulations, that is, the enforcement of so-called safety properties, blocks the execution of business processes—in particular, the so-called liveness property of process completion. Those obstructions exemplify the conflicting goals between business processes and classic IT security. This thesis addresses this problem by describing how to widen this restricted behavior of business processes resulting from security controls to the broader scope that compliance provides as part of corporate governance. By handling obstructions, security in business processes is supposed to be improved. For this purpose, an indicator-based view of security that extends the classic IT security controls will be introduced.

The SecANet approach, which will be presented in the course of this work, will allow putting this field of tension into a well-founded, formal representation of security-aware processes that opens up acting within such a desired security-sensitive realm of maneuver. The SecANet encoding will create an extendable framework that addresses workflow obstructions and will support their comprehensive analysis and handling. Furthermore, the OLive-M and OLive-L approaches will illustrate how the SecANet approach can be applied to solve obstructions based on a process model or a process log, respectively. In order to complete the workflow in that context, based on indicators captured as costs, a certain yet still compliant degree of violation of safety properties is tolerated. The solutions allow for additional, eventually live, process execution sequences because they widen the behavioral framework restricted by classic IT security in a security-sensitive way.

For evaluation, a set of example models and process logs will be generated using sample models with different degrees of complexity. Experiments based on these inputs will empirically show characteristics and functionality of the proposed approach. Because of its practical setting, this approach is applicable to a range of practical applications. For example, it could recommend who shall perform which tasks in a so-called break-glass situation, or act as a delegation assistant to suggest potential best delegates (with fewest violations) to the delegator. A corresponding process-aware information system could automate these delegations and provide additional mitigating techniques to prioritize audits of affected cases. Moreover, the graphical view of obstruction analysis could help policy designers to deepen their understanding of security policies and to improve their own security policies.

Zusammenfassung

Der wachsende regulatorische Druck auf immer stärker digitalisierte Unternehmen, der sich beispielsweise aus der Bekämpfung der zunehmenden Zahl von Betrugsfällen ergibt, kann sich hinderlich auf die Ausführung automatisierter Geschäftsprozesse auswirken. Es können sicherheitsbedingte Obstruktionen (i.S.v. Blockaden) auftreten, wenn die Implementierung von Regulierungen, insbesondere die Durchsetzung von Sicherheitseigenschaften (i.S.v. engl. safety), die Ausführung von Geschäftsprozessen, insbesondere die sogenannte Lebendigkeitseigenschaft (engl. liveness) des Prozessabschlusses, blockiert. Solche Obstruktionen verdeutlichen das Spannungsfeld zwischen den Zielen von Geschäftsprozessen und klassischer IT-Sicherheit. Die vorliegende Arbeit adressiert dieses Problem, indem sie beschreibt, wie das durch Sicherheitskontrollen eingeschränkte Verhalten von Geschäftsprozessen auf den Handlungsspielraum erweitert werden kann, der durch Einhaltung (engl. Compliance) der Grundsätze der Unternehmensführung und -kontrolle (engl. Corporate Governance) gesetzt ist. Auf diese Weise soll durch die Behandlung von Obstruktionen die Sicherheit in Geschäftsprozessen verbessert werden. Dabei wird eine indikatorbasierte Sicht auf Sicherheit als Erweiterung der Zugriffskontrolle eingeführt.

Der SecANet-Ansatz, der im Rahmen dieser Arbeit vorgestellt wird, erlaubt es, dieses Spannungsfeld in eine fundierte, formale Repräsentation sicherheitsorientierter Prozesse zu überführen, die ein Agieren innerhalb solch eines sicherheitssensiblen Handlungsraumes eröffnet. Die SecANet-Kodierung schafft ein erweiterbares Framework, das Obstruktionen in Arbeitsabläufen adressiert und deren umfassende Analyse und Behandlung ermöglicht. Darüber hinaus werden die Ansätze OLive-M und OLive-L als Anwendungsfälle vorgestellt, die veranschaulichen wie der SecANet-Ansatz zur Lösung von Obstruktionen auf

der Grundlage eines Prozessmodells bzw. eines Prozessprotokolls (Prozess-Log) genutzt werden kann. Um eine obstruierten Prozess abzuschließen, wird dabei auf Basis der als Kosten erfassten Indikatoren ein immer noch konformer Grad der Verletzung von Sicherheitseigenschaften toleriert. So ermöglichen die Lösungen zusätzliche, schließlich "lebendige" Prozessausführungssequenzen, weil sie den Verhaltensraum, den die klassische IT-Sicherheit vorgibt, sicherheitssensitiv erweitern. Zur Evaluation wird eine Reihe von Beispielen von Prozessmodellen und Prozessprotokollen mit unterschiedlichen Komplexitätsgraden erzeugt. Experimente, die auf diesen Eingaben basieren, zeigen empirisch Charakteristika und Funktionalität des vorgeschlagenen Ansatzes.

Aufgrund der aus der Praxis abgeleiteten Ausrichtung der Arbeit, ergibt sich aus dem Ansatz auch eine Reihe von praktischen Anwendungen. Er könnte zum Beispiel verwendet werden, um zu empfehlen, wer welche Aufgaben in einer so genannten Break-Glass-Situation übernehmen soll, oder um als Delegationssassistent zu fungieren, welcher Delegierenden die potenziell besten Kandidaten zur Delegation (mit den wenigsten Sicherheitsverletzungen) vorschägt. Ein entsprechendes prozessorientiertes Informationssystem könnte diese Delegationen automatisieren und zusätzliche Verfahren zur Risikomitigation zur Verfügung stellen, welche die Prüfung der betroffenen Fälle priorisieren. Darüber hinaus könnten die durch den graphbasierten Modellierungsansatz gegebenen Visualisierungsmöglichkeiten der Obstruktionsanalyse Policy-Designern helfen, das Verständnis von Sicherheits-Policies zu vertiefen und diese zu verfeinern.

Contents

1	Why the Automation of Regulation Can Obstruct Business	
	Processes	1
1.1	Fraud in a Digitally Enabled World of Processes	4
1.1.1	Towards Process-Awareness and Process Automation	6
1.1.2	IT Security Applied on Processes	10
1.1.3	Security Requirements Obstructing Process Execution	14
1.2	Conventional Anti-Fraud Measures	16
1.2.1	Post-9/11 and Post-Financial Crisis Regulation	17
1.2.2	Corporate Governance	18
1.2.3	The Need to Support Regulation by Automation	22
1.3	Obstruction by Security?	23
1.3.1	The Dilemma of Process Security in Following upper Policies	24
1.3.2	Indicator-Based Process Security	25
1.4	Contribution and Structure of Thesis	27
2	Security-Related Obstructability in Process-Aware Information Systems	31
2.1	Preventive Process Analysis: Obstructability by Design	38
2.1.1	Process Specification	39
2.1.1.1	The Process Model	40
2.1.1.2	The Enforceable Policy	42

2.1.2	Satisfiability	49
2.1.2.1	Initial Publications	50
2.1.2.2	Process Structure	52
2.1.2.3	Order of Assignment	52
2.1.2.4	Constraint Types	53
2.1.2.5	Complexity and Fixed-Parameter Tractability	55
2.1.3	Resilience	56
2.1.3.1	Synthesizing Execution Plans	57
2.1.3.2	Quantification	58
2.1.3.3	Feasibility and Change of Policies	58
2.1.3.4	Policy Violation	59
2.1.4	Obstructability: The New Factor	60
2.1.4.1	Requirements for Obstructability Analysis (ROA)	62
2.2	Process Execution Monitoring: The Case of Obstruction at Runtime	66
2.2.1	Process Enactment	67
2.2.1.1	Process Execution	68
2.2.1.2	PAIS Steering Execution	68
2.2.2	Avoiding Obstructions	70
2.2.2.1	Enforcing Obstruction-Free Workflows	70
2.2.3	Handling Obstructions	71
2.2.3.1	“Breaking” the Policy	71
2.2.3.2	Delegation	72
2.2.3.3	Changing the Policy	73
2.2.3.4	Violating the Policy	73
2.2.4	Completeness	74
2.2.4.1	Requirements for Specification-Based Completeness (RSC)	74
2.3	Detective Process Analysis: The Case of Obstructability by Incompleteness	76
2.3.1	Process Logs	79
2.3.1.1	Formats	80
2.3.1.2	Filtering	82
2.3.2	Process Mining	84
2.3.2.1	Process Discovery	84
2.3.2.2	Conformance Checking	85
2.3.2.3	Enhancement	87

2.3.2.4	Offline and Online Process Mining	89
2.3.2.5	Predictive Monitoring	90
2.3.3	Log-Based Completeness Requirements (RLC)	93
2.4	Security-Sensitive Detection and Handling of Obstructions	95
2.4.1	Main Deficits in Obstructability Research	97
2.4.2	New Spheres of Action: Expanding the Behavioral Space	99
2.4.2.1	General Framework for Requirements	100
3	Obstruction Modeling	103
3.1	Ways to Model Processes	105
3.1.1	Petri Nets: The Method	108
3.1.1.1	Petri Net	109
3.1.1.2	Basic Modeling Examples	116
3.1.2	Petri Net Subclasses	119
3.1.2.1	Place- and Transition-Related Systems	120
3.1.2.2	Choice-Related Nets	121
3.1.2.3	Workflow Net	124
3.1.2.4	Workflow Soundness	125
3.1.2.5	Policy-Aware Workflow Net	132
3.2	The SecANet Solution	135
3.2.1	The Principle of “Flattening”	137
3.2.1.1	Modeling Authorization	138
3.2.1.2	Modeling Constraints	139
3.2.1.3	Example SecANet Analysis	141
3.2.2	Generalizing Flattening	143
3.2.2.1	Flattening of User-Task Authorization	143
3.2.2.2	Flattening of SoD Constraints	145
3.2.2.3	Flattening of BoD Constraints	146
3.2.2.4	SecANet Obstructions	147
3.2.2.5	Capturing Indicators by Costs	150
3.2.3	SecANet Properties	151
3.2.3.1	Net Properties of Modeled User-Task Authorization	151
3.2.3.2	Net Properties of Modeled Constraints	154
3.2.4	Language-Based SecANet Properties	156
3.2.4.1	Language-Related Operators and Nets	156
3.2.4.2	SecANet Language Types	162
3.2.4.3	Decomposition	164

3.2.4.4	Composition	167
3.2.4.5	SecANet Language	173
3.2.4.6	Language Preservation	187
3.2.5	Cyclic Behavior and Policy Re-Enactment	200
3.2.5.1	Points and Scopes of Policy Re-Enactment	203
3.2.5.2	Flattening Policy Re-Enactment	211
3.2.6	SecANet Analysis: Satisfiability and Obstructability	223
3.2.6.1	SecANet Soundness	225
3.2.6.2	SecANet Reversibility and Deadlock Analysis	229
3.2.6.3	SecA-WF-Net: Security-Aware Workflow Net	231
3.2.7	Experimental Evaluation	237
3.2.8	Discussion	240
3.2.8.1	Method of Modeling	241
3.2.8.2	Time, Space, and Output Memory Requirements	242
3.2.8.3	More Efficient Analysis	246
3.2.8.4	Common and Further Constraints	250
3.3	SecANet+: Extension of the SecANet Approach	252
3.3.1	Create and Combine Policy Nets for Workflows	254
3.3.1.1	Creating Policy Nets	255
3.3.1.2	Combining Policy-Related Nets and the Workflow	259
3.3.2	Resilience Extension: Modeling User Absence	263
4	OLive-M: A SecANet Use Case for Model-Based Obstruction	
	Solving	267
4.1	Methods and Realization of the OLive-M Approach	270
4.1.1	Structural Theory of Petri nets	271
4.1.2	Linear Programming	278
4.2	OLive-M Experiments for Model-Based Obstruction Solving	281
4.2.1	Implementation	282
4.2.2	Remarks on Costs and Results	282
4.2.3	Experiment Preparation	282
4.2.4	Experiment Setup and Solution	283
4.2.5	Experiments with More Extensive Nets	284
4.2.6	Replayability	285
4.2.7	Obstruction Position	285
4.2.8	Differing Costs	286

4.3	Discussion and Potentials of the OLive-M Approach	286
4.3.1	Limitations in Solutions	288
4.3.2	Efficiency	288
4.3.3	Replayability	289
4.3.4	Emergency-SecANet	289
4.3.5	Extending the OLive-M Principle to Analyze Satisfiability and Resilience	291
4.3.5.1	Analyzing Satisfiability	292
4.3.5.2	Analyzing Resilience	293
5	OLive-L: A SecANet Use Case for Log-Based Obstruction Solving	295
5.1	Methods and Realization of the OLive-L Approach	297
5.1.1	Trace Replay	298
5.1.1.1	Trace-Related Notations	298
5.1.1.2	Replay-Based Partitioning of Traces	299
5.1.2	Nearest Match	300
5.1.2.1	k-Nearest Neighbor	300
5.1.2.2	kNN-Based Completion Trace	301
5.1.2.3	Security-Sensitive Costing of Candidates	303
5.2	OLive-L Experiments for Log-Based Obstruction Solving	305
5.2.1	Implementation	305
5.2.2	Experiment Preparation: Obtaining Traces	306
5.2.3	Experiment Setup and Solution	306
5.2.4	Experiments with Extensive Logs	307
5.3	Discussion and Potentials of the OLive-L Approach	307
5.3.1	Log and Partitioning	308
5.3.2	kNN and Selection of Completion Segment	308
5.3.3	Security-Sensitive Costing	309
5.3.4	SecANet Discovery	310
5.3.5	Log- and Model-Based OLive Extensions	310
5.3.5.1	OLive-LM: Refining the Log-Based approach with the Model-Based approach	311
5.3.5.2	OLive-ML: Refining the Model-Based Approach with the Log-Based Approach.	311
6	Towards Intelligent Security- and Process-Aware Information Systems	315
6.1	Application	318
6.2	Extension	320

6.2.1	Beyond Security-Sensitivity: Multi-Objective Solutions	320
6.2.2	Beyond the Case: Inter-Instance- and Inter-Process-Related Obstructions	321
6.2.3	Beyond Predictions: Corrective Monitoring upon Occurring Obstructions	321
Bibliography	323

List of Figures

Figure 1.1	Sketch of the Anglo fraud process	5
Figure 1.2	Business process management lifecycle	7
Figure 1.3	Example of a collateral evaluation workflow (CEW)	8
Figure 1.4	Behavioral scope of (unregulated) business	11
Figure 1.5	Enterprise architecture layers and regulation	12
Figure 1.6	Access control = authentication + authorization	13
Figure 1.7	Behavioral scope of IT security on processes	15
Figure 1.8	PAIS with policies and contextual influences	16
Figure 1.9	Corporate governance	19
Figure 1.10	Behavioral scope of enterprise bounded by regulation (circle)	21
Figure 1.11	Structure of thesis	28
Figure 2.1	Obstructive sequence of activities	32
Figure 2.2	Example of property classification	34
Figure 2.3	Cause of obstruction and desired outcome	36
Figure 2.4	Terminology related to the process lifecycle	37
Figure 2.5	Highlighted sub-process to determine the market value of a collateral	41
Figure 2.6	DMV process model and policy	49
Figure 2.7	A satisfiable assignment of users to tasks	50
Figure 2.8	An obstructed execution	61
Figure 2.9	Standard transactional life-cycle model	80
Figure 2.10	Positioning of the three main types of process mining	85
Figure 2.11	Using logs to deduce resource indicators	88
Figure 2.12	Recommendation	90
Figure 2.13	Predictive Process Monitoring	91

Figure 2.14	Process behavior sketching satisfiable and resilient executions	96
Figure 2.15	Potential behavioral gain in handling obstructions	98
Figure 2.16	Main deficits regarding completability	99
Figure 2.17	Contribution of work to analyze, detect and resolve obstructions	101
Figure 3.1	Petri net of the Determine Market Value (DMV) process	109
Figure 3.2	DMV Petri net with initial marking	112
Figure 3.3	DMV Petri net with new marking after firing t_1	113
Figure 3.4	Parallel activities in a Petri net	117
Figure 3.5	Conflict, choice, or decision Petri net structure	118
Figure 3.6	Symmetric confusion	119
Figure 3.7	Asymmetric confusion	119
Figure 3.8	Three manifestation of the free-choice property	122
Figure 3.9	(Extended) free-choice	123
Figure 3.10	Sequential pattern	127
Figure 3.11	Exclusive pattern	127
Figure 3.12	Parallel pattern	128
Figure 3.13	Loop pattern	128
Figure 3.14	Example of a live, safe, FC, but not well-structured WF-Net	129
Figure 3.15	Example of a live, safe, AC, well-structured, but not FC WF-Net	129
Figure 3.16	Example of a live, safe, well-structured, but neither FC nor AC WF-Net	130
Figure 3.17	Example of a live, safe, well-structured, and now also FC WF-Net	131
Figure 3.18	Basic BPMN to WF-net patterns	133
Figure 3.19	CEW process in BPMN	133
Figure 3.20	CEW WF-Net	135
Figure 3.21	Determine the market value	135
Figure 3.22	The SecANet approach	136
Figure 3.23	Flattening authorization and SoD into the DMV WF-net	137
Figure 3.24	Obstructed marking in flattened WF-net	141
Figure 3.25	Generalized SoD with application to simplified payment workflow	148

Figure 3.26	Generalized BoD with application to simplified payment workflow	148
Figure 3.27	Basic choice net for different user assignments for a single task t	153
Figure 3.28	Basic choice net with complement	154
Figure 3.29	Determine Market Value workflow as WF-net with initial marking	162
Figure 3.30	Example decomposition of the SecANet example	166
Figure 3.31	Example decomposition of N_{TA+SoD}	167
Figure 3.32	Synchronization example $\dot{N}_{TASy\{u_1t_1, u_1t_2\}}\dot{N}_{SoD}$	169
Figure 3.33	SecANet decomposition	173
Figure 3.34	User-task subnet for t_2	177
Figure 3.35	Language Preservation	189
Figure 3.36	Flattened net with silent transitions	192
Figure 3.37	DMV process with loop and re-enactment point	203
Figure 3.38	DMV Petri net with loop	204
Figure 3.39	Idea of policy re-enactment	207
Figure 3.40	CEW re-enactment points	208
Figure 3.41	Nesting of re-enactment blocks in parallel	210
Figure 3.42	DMV example with re-enactment, cancellation, and enactment controls	216
Figure 3.43	DMV loop and re-enactment	221
Figure 3.44	SoD cancellation example	221
Figure 3.45	User-task assignment	222
Figure 3.46	The collateral evaluation SecANet	224
Figure 3.47	SecANet that fulfills the option to complete	226
Figure 3.48	SecANet that does not fulfill the option to complete	229
Figure 3.49	Obstructability analysis example	230
Figure 3.50	SecA-WF-Net encapsulation and extension with t^*	232
Figure 3.51	DMV with loop and WF structure	234
Figure 3.52	SecA-WF-Net that can be used to check for liveness	236
Figure 3.53	Impression of SecANet (6 tasks, 60 users, and 1 SoD constraint)	238
Figure 3.54	Impression of SecANet (10 tasks, 100 users, and 4 SoD constraints)	238
Figure 3.55	DMV free-choice SecANet with initial marking	245
Figure 3.56	DMV free-choice SecANet representation of the obstruction marking	247
Figure 3.57	DMV free-choice SecANet variant	248

Figure 3.58	DMV free-choice SecANet with further variant	249
Figure 3.59	Variation of the DMV SecANet with constraints causing an obstruction	251
Figure 3.60	Obstruction marking in the variation of the DMV SecANet	251
Figure 3.61	The SecANet+ approach	253
Figure 3.62	SecANet+ for security-aware process specification, with user-task authorization and SoD	255
Figure 3.63	Example of user-activity net $N_{a_1}^{UA}$	257
Figure 3.64	Example of a user-specific SoD Petri net $N_{u_1}^{(a_1, a_2, \neq)}$	258
Figure 3.65	Combination of the two example user-activity Petri nets N_A^{UA}	260
Figure 3.66	Example DMV SecANet $N_{wf}^{UA, sy, SoD}$ after combining all nets	261
Figure 3.67	Compact representation of user-availability net	263
Figure 4.1	Overall OLive-M approach to resolve obstructions	268
Figure 4.2	DMV SecANet with costs and token to escape the obstructed state	269
Figure 4.3	Incidence matrix of DMV workflow net	272
Figure 4.4	Incidence matrix of DMV SecANet	273
Figure 4.5	(a) Petri net. (b) Potential reachability graph. (c) Marking equation	275
Figure 4.6	OLive-M state equations applied on an obstructed DMV SecANet	276
Figure 4.7	OLive-M state equation to resolve obstruction in DMV SecANet	277
Figure 4.8	Δ and X to escape obstructed state	278
Figure 4.9	Experiment runtimes	285
Figure 4.10	Impression of six concatenated nets	287
Figure 4.11	“Emergency-SecANet” with costly additional user-task assignment	290
Figure 5.1	The OLive-L approach	296
Figure 5.2	Log-based approach with example traces	296
Figure 5.3	Sketch of obstructed and successful traces in n-dimensional space	301

List of Tables

Table 1.1	Example event	9
Table 1.2	Comparison of Freiburg approaches addressing security	26
Table 2.1	Terminology related to the process lifecycle	38
Table 2.2	Example DMV trace	80
Table 3.1	Runtimes for satisfiability and obstructability check	240
Table 4.1	ILP statistics	284
Table 5.1	Encoding of successful traces in 12-dimensional space	305
Table 5.2	Solution for $k = 5$ with highlighted partial sequence	306



Why the Automation of Regulation Can Obstruct Business Processes

1

The recurring question as to whether regulation is appropriate or obstructive is becoming more and more urgent due to its ever-increasing pressure on businesses in a digitally enabled world. In the year 2017, a decade after the financial crisis, 56,321 regulatory alerts from more than 900 bodies were tracked worldwide. This is highlighted by the chief compliance officer (CCO) of HSBC bank who recently stated that “you have to build an industrial-scale operation just to digest all the regulatory changes” [83].

Certainly, the question if regulation is appropriate or rather obstructive for business is not new. It is valid for many instances in business, with specific attacker models. The attention for business models was first taken care off by the Sarbanes Oxley Act (SOX) of 2002. The so-called SOX paradox describes a situation where the need for compliance and internal controls can reduce the ability of a company to be agile and dynamic in the marketplace. This scenario is paradoxical because the internal controls that must be documented under SOX are meant to help a company perform well and meet its financial goals [196]. The preceding American Patriot Act, which targeted money flowing to terrorists and other perpetrators after the September 11 attacks, had comparable side effects. The question emerged if requirements of costly “Know Your Customer” initiatives were disproportionate to the risk of money laundering and were rather obstructing business with normal bank customers [96]. Similar problems have recently been observed with respect to the General Data Protection Regulation of the European Union (GDPR), by which especially small and mid-sized businesses were overstrained by a plethora of regulatory requirements. Indeed, there are different ways how regulation can obstruct business. The sheer personnel effort and working hours needed for checking and ensuring compliance to regulation, or the burden of keeping track of the abundance of regulations and their implementation are just common examples. However, basic regulatory concepts and control functions in a business, especially internal controls

may act obstructive as well. Here, **authorization** or the assignment of permissions, for instance the right of a process participant to execute a certain process activity, internal control should ideally apply the principle of least privilege. According to this principle, process participants should only have access to the resources that are indispensable to perform their job responsibilities. By this for example access from unwanted staff can be avoided and the risk of fraud can be reduced without being too restrictive. **Separation of duties (SoD)** is known to be the quintessential internal control practice to avoid fraud. The institute of internal auditors (IIA) calls SoD a fundamental element of internal control for the segregation of certain key duties. The basic idea underlying SoD is that neither one employee nor a group of employees should be in a position to perpetrate or conceal errors or fraud in the normal course of their duties. In general, the principal incompatible duties to be segregated are [20] (1) the custody of assets, (2) authorization or approval of related transactions affecting those assets, and (3) recording or reporting of related transactions. For instance, such controls restricting access to business process activities can lead to staff shortages which can directly obstruct behavioral possibilities and restrict the scope of action of a business. Instead of a SoD rule, authorization can also be chosen so restrictively that an SoD conflict cannot occur at all. This however also restricts the flexibility, namely the possible user-task assignments. Hence, both authorization and further rules such as SOD are in control of the process executors. Their utilization expresses the risk a company is willing to take.

Foremost, regulation originates from combating financial crime and fraud. The scale and impact of corporate fraud has increased significantly in today's digital world. Regulation, fraud and digitization are in a feedback loop in which the latter is the driving force. Its prerequisites are data and processes [152]. Digitization or the digital transformation are often directly connected with the digitization of business processes and (robotic) process automation, i.e., processes are more and more computer-supported. A current McKinsey survey of almost 1,000 top managers supports this. Those surveyed rank "automation and/or improvement of business processes" (besides the digitized integration of customers and digitized innovation processes) among the most prioritized opportunities in the digitized economy. Hence, operations in enterprises are ultimately all attributable to digital representations of processes, be it healthcare, finance, transportation or manufacturing. Therefore, regulation must operate on these processes and the information systems that enact them.

With the digitization and the automation of business processes, it seems as if the "burden" of regulation could be eased if regulation was also automated. Automating the implementation of regulation is seen as the panacea against the ever-pressing regulatory pressure on businesses. In finance it is even reported that today's "biggest

question for bank controllers is how many humans they can replace with bots without compromising compliance” [83]. The idea of also automating the implementation of regulation certainly makes sense in order to keep pace with process automation and make appropriate use of the accumulation of large amounts of data. This development has led to a research focus in the field of business process security at the Department of Telematics of the Institute of Computer Science and Social Studies at the University of Freiburg. Related lectures, research projects, widely published articles and a range of doctoral theses also resulted in the development and publication of the Security Workflow Analysis Toolkit (SWAT) [9, 216]. SWAT is a tool for security analysis of business processes, providing methods for model- and log-based analyses and verification of security properties either from a preventive or a forensic point of view.

Automating regulation can still obstruct the execution of business processes, for instance, when there are no employees authorized to perform a pending process activity at hand. Hence, despite the many benefits of automation, the problem of regulation obstructing business is only shifted to the technical level. Regulation and its related concepts of governance provide ways how to resolve obstructions in order to support business operation. However, such solutions are not easy to find on a technical level. The problem here is that information technology (IT) security aims to remain in a secure state rather than allow the continuation of a business process.

In summary, if a business process violates higher societal or human rights, the process is illegal. Moreover, a business process could also support crime, for instance, money laundering. In both cases, law or regulation is supposed to be enforced in such a way that such processes are detected and stopped. However, as seen above, regulation can also disturb business processes, and its implementation can cause an unjust burden between the owners of the processes or their participants. The automation of the implementation of regulation can finally cause obstructions by inequality and costs. Thus, this thesis does not aim to avoid the legitimate interception of illegal activities but to harmonize the obstructive part of regulation and automated business processes in a security-sensitive way. Here, in particular, the conditions to detect and prevent obstructions in business processes will need to be identified.

This chapter aims to give an in-depth understanding on where these security-related obstructions come from and introduces the context in which to solve them. To motivate the need to automate the application of regulation and show its problems in terms of obstruction, the developments in fraud, regulation and towards process automation will be related in loose chronology to each other. In this way reasons will be identified why regulation acts obstructively and its application needs

to be automated. Since big parts of regulation have indeed been automated on the infrastructure and application layer, the focus of this work will be on how to apply regulation on the business layer. It is the layer that allows for an aggregated view on activities that would otherwise appear isolated in the lower layers. In this way, dots can be connected so that fraud schemes can be revealed. In doing so, classic concepts of IT security will be introduced, which, however, can lead to such obstructions on the technical level. The way classic IT security handles obstructions here cannot work in processes, therefore the concept of indicator-based security will be introduced. Based on these findings, the section on the contribution and structure of the thesis will indicate how the problem of obstruction is tackled.

1.1 Fraud in a Digitally Enabled World of Processes

In retrospect, the 2007-08 financial crisis is often seen as the event that brought about today's plethora of regulatory and compliance measures. The following case of a European Bank involved in the financial crisis represents an exemplary reason for the drastic increase in regulation to fight illegal, criminal, or fraudulent processes after the crisis and also the huge effort and costs needed in solving the case: Roughly a decade after the global financial meltdown, in June 2018, the former CEO of the Anglo Irish Bank (AIB), David Drumm, was found guilty of dishonestly inflating the size of the bank's deposits by 7.2 billion Euro before its collapse and subsequent bail-out during the financial crisis. He was accused of conspiring to carry out the fraudulent transaction process with his former finance director, his former treasury department manager as well as the former CEO of the Irish Life and Permanent (IL&P) Bank, and others. These men were involved in setting up a circular scheme of billion Euro transactions in which AIB moved 1.2, 2 and 4 billion Euro to IL&P which then sent the money back via their assurance firm "Irish Life Assurance" to AIB (see Figure 1.1).

The scheme was designed in order to achieve that the deposits come from the assurance company such that they will be treated as customer deposits, which are considered a better measure of a bank's strength than inter-bank loans. Therefore, it acts as an indicator of the health of the bank, since such deposits show that corporate entities have faith in it [54, 102]. So, in the face of the financial crisis in fall 2008 and the end of the fiscal year on 30th September, the books were faked to mimic achievement of financial goals and palliate the state of the bank to still appear good enough to the government and the European Union (EU) in order to get financial support. Finally, apart from other related legal proceedings, the prosecution that led

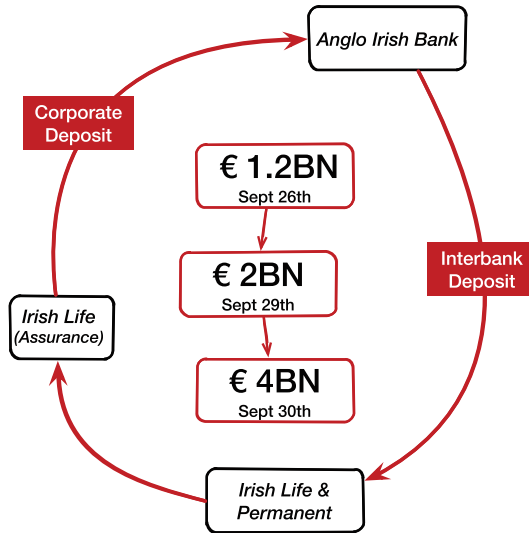


Figure 1.1 Sketch of how the Anglo “Back To Back” fraud process worked

to the imprisoning of the former CEO of the AIB cost 1.123 million Euro [72] and was one of the longest-running criminal trials in the history of the Irish State.

Still today, “statistics show that economic criminals are increasingly targeting businesses. However, businesses need to be aware that wrongdoing can also come from within.” What the Irish Garda National Economic Crime Bureau conclusively stated on the AIB case is often connected to the notion of “white-collar crime”¹. Indeed such crime or fraud within the operational procedures in enterprises recorded a tremendous increase in the last decades. Besides the AIB, the Swiss bank UBS suffered from a rogue trader scandal in 2011, which led to an estimated loss of 2 billion US Dollar. Earlier the French Société Générale had a loss of nearly 5 billion Euro caused by shuffling transactions. Similar constellations with inside perpetrators or internal attackers can also be seen in the fraud cases involving the WorldCom, Parmalat or Enron cases, or in more recent scandals, for instance, in the automotive industry or the financial technology industry (FinTech), for example, the “Dieselgate” or “Wirecard”, respectively. Today’s presence of fraud is drastically stressed by the 2018 fraud report of the Association of Certified Fraud Examiners

¹ Whereas in “corporate crime” the company or the corporation are beneficiaries, in “white-collar crime” the individual involved benefits from its actions.

(ACFE), analyzing 2690 cases of occupational fraud that were investigated between January 2016 and October 2017. The total loss caused by the cases in the study exceeded 7.1 billion US Dollars, while these cases only represent a small fraction of the frauds committed against organizations worldwide during that time. The real cost of global fraud is probably significantly higher, especially when the indirect costs are considered, such as reputational harm and the loss of business during the aftermath of a scandal but also simply undetected or unreported frauds. Based on these limitations anti-fraud experts estimate that organizations worldwide lose on average 5% of their annual revenues to fraud (i.e., a total 4 Trillion US Dollars of the Gross World Product in 2017) [41].

One driver for this increase and ever-pressing issue of fraud is digitization. With the digital transformation, the risk of damage, its amount and the velocity of its growth significantly rise, which can be seen from growth in financial processes including private home banking. Clearly, fraud cases would also happen in an analog environment. Indeed, there are records of accounting fraud at Medici Bank dating back to the 15th century [131]. Nevertheless, the increase and size of recent scandals has, at least to some extent, made use of digital technologies, or were only made possible by them. Just by looking at the AIB case, the circular scheme's high frequency of transactions would not have been possible without the respective digital transaction systems. Even the exchange-traded fund transactions at UBS or the accumulation of transactions at Société Générale that operated directly below the risk threshold would not have been possible to such an extent in an analog system.

1.1.1 Towards Process-Awareness and Process Automation

On a positive side, however, digitization is the driver for comprehension and adaptation of business in a process-oriented way. Information systems of organizations are moving more and more from a traditionally data-centric direction that supports business processes, towards a fully process-oriented view. This is because of the necessity to standardize procedures and adapt to changing environments in the course of globalization. The conception of standardized software for steering business processes follows this development. In general, a business process realizes business goals by orchestrating business activities in coordinated sequences. Process-orientation in software-systems already dates back to 1993 when SAP R/3 ERP-System adapted to a process-oriented view by introducing the Business Process Reference Model. However, no explicit control of the process was supported. Since the mid 90s, methods for process automation have created a series of systems, which supported the operation of business processes by an automated execution of

partial steps and shaped the notion of “workflow management“. Business process management (BPM) extends workflow concepts with a holistic view on processes in a business context. Workflow and BPM Systems can be subsumed under the notion of process-aware information systems (PAIS). A PAIS is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models [3]. They include all software systems that support processes and not just isolated activities [82].

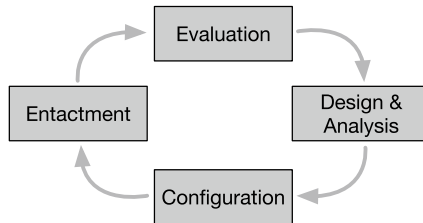


Figure 1.2 Business process management lifecycle

While workflow management can be understood as a supporting technology, BPM is a process-oriented management discipline, which is based on the premise that efficient management and systematic process optimization contribute to the success of an enterprise. Thus, in addition to the (semi-)automated execution of processes, the entire process life-cycle is considered. Figure 1.2 illustrates its different phases. It starts with the specification that includes the modeling of planned process behavior, and contains the implementation in the system landscape of the enterprise, followed by process enactment as the phase in which the process is actually executed. It finalizes with the diagnosis to identify optimization potentials after execution. Along these phases, the process can be captured in different process entities, specifically, the process model, the process execution and the event log. As such, a process is executed by instantiating activities, whereas the resulting activity executions are coordinated. Activities are conducted sequentially or concurrently to each other; their execution is dependent on explicit decisions; and parts of a process may even be repeated multiple times.

The following example connects known facts of the AIB investigation to different process entities. Since IL&P was not willing to give unsecured loans because the cash limit of 100 Mio Euro was exceeded, they required a collateral to secure each loan. Therefore, AIB provided cash collateral that usually is safeguarded by several transactions. The involved process defines how a collateral evaluation for a secured

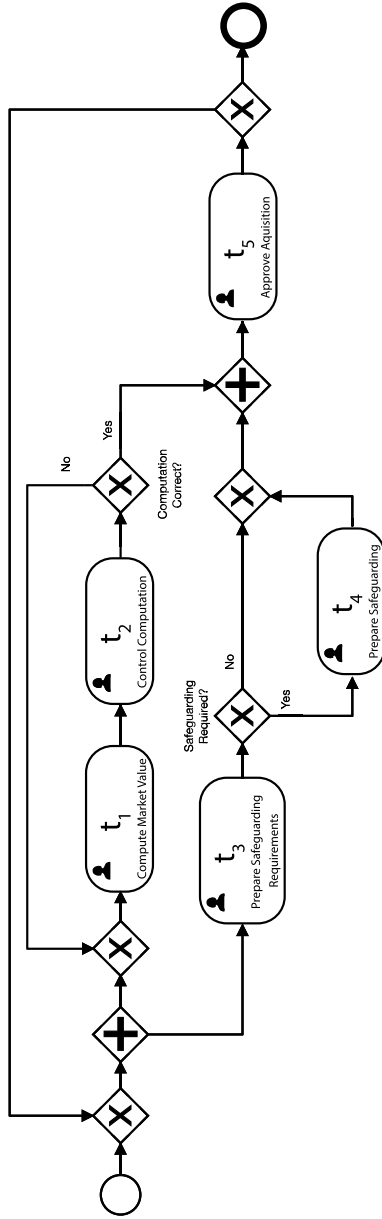


Figure 1.3 Example of a collateral evaluation workflow (CEW)

loan is handled, with the outcome being the approval of the acquisition. A first activity is to determine the market value of the respective collateral. A decision is then made as to whether the value is correct. If it is correct, the acquisition of the collateral can be approved.

Figure 1.3 illustrates a process model that describes how a collateral evaluation in a financial institution is handled (based on the process description in [24]). It is executed to evaluate, accept, and prepare the safeguarding of the collateral that a borrower pledges in return for a secured loan and originates from the IBM Information Framework on Loans' process templates on the evaluation of collateral. This model, captured in the Business Process Model and Notation (BPMN), serves for illustration in the further course of this thesis. The execution of the process activities is coordinated within a specific scope, the so-called case. A case represents an instance of the process and is defined by all activity executions that refer to a particular trigger or input for the system whose behavior is described by the process. An event log fulfills the function of a business process tachograph. It records how the process is actually "lived". Once this process is supported by a PAIS, event data on the execution of this process is collected and may comprise information on the time a particular activity was executed for a case. The latter is related to a user, the requested amount, and the prospective duration of the loan deposit. Each collateral evaluation then represents a case, as part of which the activities of the process are executed. An example of such an event is given in Table 1.1.

Hence, processes capture the behavioral scope of an enterprise. Figure 1.4 sketches this scope of action. The points schematically symbolize the set of states of a PAIS resulting from performing its business activities, which can involve different users and data and belong to specific processes. The arcs represent these executions of business activities which change the system state. Hence, the arcs correspond to an event or a performed process activity and illustrate possible activity sequences. All combinations of activities, users and involved data are possible here. This could also involve fraudulent behavior.

Table 1.1 Example event

Case	Timestamp	Activity	User	Amount	Duration
248	16:17, September 28, 2008	Approve Acquisition (Aa)	David Drumm	1.2 BN Euro	10 days

1.1.2 IT Security Applied on Processes

Hence, in order to also use the potentials of digitization to secure such processes and combat fraud, computer security is applied on processes. Security in business processes represents an interdisciplinary field, which links security mechanisms from the security and business process management research communities [133]. Subsequently, security requirements on business processes are introduced, which allow to be enforced on business processes in an automated manner. Engineering secure processes is a multi-faceted problem, resulting from the different layers and services an enterprise contains. The architecture of an enterprise from an engineering perspective is captured with the so-called enterprise architecture meta-model [176], which can be used to characterize the level of abstraction a PAIS application exhibits [134]. This model divides the support of IT in modern enterprise system architectures in the subsequent inter-layers: The **infrastructure layer** provides the software and hardware needed to automate the execution of services or business processes respectively. This includes, for example, the required databases, the operating system, virtual machines and protocols (e.g. transport protocols, networks). On top of that, the **application layer** contains the services and data schemes that are required for the execution of the processes. In this context, service-oriented architectures (SOA) or higher service modes of Cloud and Grid computing are relevant. Although regulatory controls may be implemented on all three layers, such scenarios for the application layer and infrastructure layer have been investigated thoroughly and may be adapted and carried over to new architectures [153]. The upper **business layer** is the abstraction layer, containing business processes, which codify business goals and its process participants, the business objects and the assets of the company, as well as its organizational structure and guidelines to be followed. It allows for an aggregated view on the enterprise activities such that coherencies and interconnections can be identified, which is eminent for fraud prevention, e.g., to reveal fraudulent schemes. Enforcing the described subsequent notion of regulation and compliance can only be done and justified from this higher business level. Figure 1.5 relates these layers of the enterprise architecture to the unregulated business world, regulation, and its impact on security requirements.

Security requirements are standard principles to enforce security in information systems (IS). Hence, to secure business processes in a PAIS, the security requirements on the business level are subsequently deduced. IT security, or equivalently “computer security”, is basically information security applied to technology. It is based on the three security requirements (tantamount to “security goals” or “protection goals”), which are confidentiality, integrity, and availability. “Control” in classic computer security is basically introduced by access control, consisting of

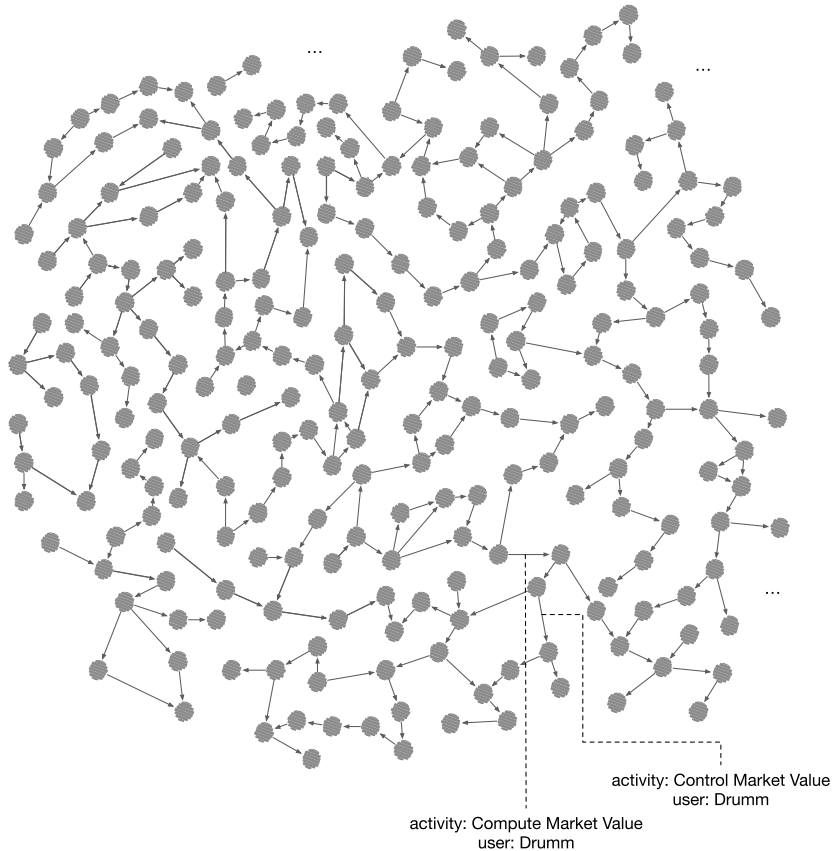


Figure 1.4 Behavioral scope of (unregulated) business symbolized as a set of states (points) resulting from performed process activities (arcs)

authentication and authorization. An access control mechanism supports *confidentiality*. It restricts access by authorizing only certain subjects, such as people or (software) agents with the right to access the desired object. An object may be a file, a process or also a process task, activity or transaction. Before authorization, authentication is used to verify *integrity*, whether the subject is what or who it claims to be. Based on Gollmann, Figure 1.6 depicts this concept, in which a subject, here the *user*, requests access. After authentication, the reference monitor checks by means of the given access control list (ACL), if the requesting user is allowed to access

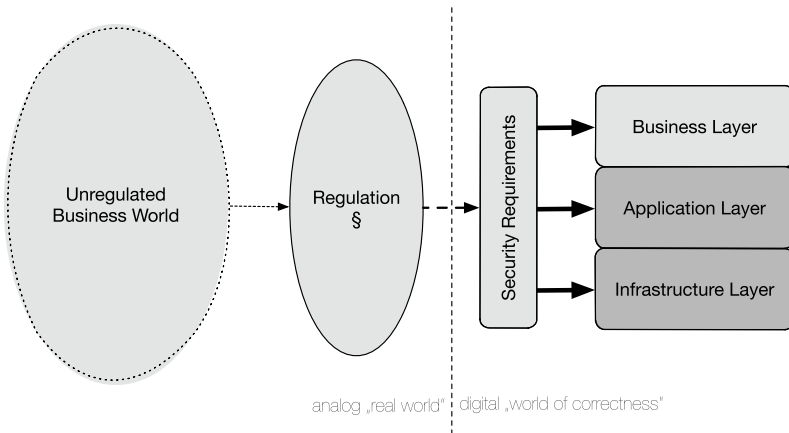


Figure 1.5 Enterprise architecture layers and regulation

the desired object [98]. Besides ACL, a further common concept is the Role-Based Access Control (RBAC), which organizes users in groups with well-defined permissions, which is also represented in several NIST standards (cf. RBAC-Level-2 also including SoD). The basis for access control to function properly is, that the corresponding systems and mechanisms are *available*, such that there is no denial of service (DoS).

A security policy is a statement that partitions the states of a system into a set of authorized (secure) states and a set of unauthorized (no secure) states [33]. It considers all relevant aspects of confidentiality, integrity, and availability. Regarding fraud, unauthorized access may violate integrity. Security policies mostly define safety properties to “keep bad things from happening”, for instance to avoid unauthorized access or that the same person performs two conflicting duties (SoD). In contrast, liveness properties try to “make good things happen”, for example all duties can eventually be performed and required resources are available [13].

What is critical to security is the distinction between policy and mechanism. Whereas a security policy is a statement of what is and what is not allowed, a security mechanism is a method, tool, or procedure for enforcing a security policy (e.g., the introduced access control mechanism). Safety properties are typically enforceable, whereas liveness properties are not. Regulation may formulate rules or procedures addressing a range of different aspects and may concern monetary aspects or people authorized to perform specific transactions. To capture these different dimensions

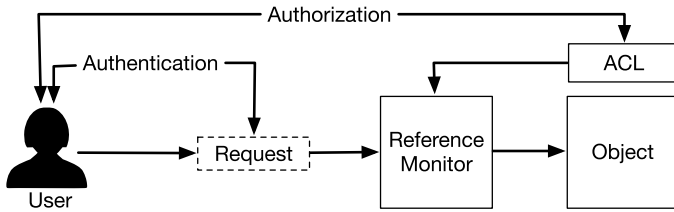


Figure 1.6 Access control = authentication + authorization

and its security requirements, a more fine grained view on processes need to be introduced as well.

The deduction of security requirements on the business process level is structured on the basis of different viewpoints on a business process, namely process aspects. These aspects are connected with security facets defined in several standards, e.g., the ITU-T Recommendation standard for communication security X.810 or ISO/IEC 15816:2002, among which are the more detailed requirements authentication, access control, non-repudiation, confidentiality, integrity, security audit and alarms, and key management, denial of service and availability [154]. Based on the work of Charfi and Mezini [42], relevant process aspects are fourfold. While the *behavioral and functional aspect* relates to causal and temporal relations between process activities together with structural compositions of processes (often named as the “control flow”), the *organizational aspect* considers organizational structures and process participants. The *informational aspect* is about the utilization of data and data flow in processes. Although these aspects are clearly separated by definition, security requirements often comprise several aspects at once.

Based on these aspects, related policies can be classified according to the subsequent general types [153]:

- **Authorization** means enforcing access control to ensure that only authorized individuals or roles are allowed to execute activities within a process [180].
- **Usage control** expresses conditions that must hold after the access to a resource [179]. Usage control policies can be used to capture regulatory compliance, privacy and data protection requirements.
- **Separation of duties** expresses and subsumes constraints associated with the process to limit the abilities of agents to perform activities, eventually reducing risk of fraud [35]. The **four-eye rule** defines that a critical activity in a business

process cannot be carried out by a single subject. **Binding of duties** (BoD) characterizes the very opposite of separation of duties.

- **Isolation** generally says that information must remain confidential during the execution of a process. In other words, there is no information or data leak [11].

Literature of relates to workflows of business processes that encompass authorization and further constraints as “security-aware workflows”.

1.1.3 Security Requirements Obstructing Process Execution

Given these classes of policies and the limitations of their enforceability (depending on whether they encode safety or liveness properties), security can be automated. However, Figure 1.7 partitions the entrepreneurial behavioral space into secure and non-secure states. It shows that the behavioral scope of automated implementation of regulation drastically restricts the behavior. This is because of the identified notion of classic IT security whose policies can only allow authorized and secure or non-secure states of the process execution (the parts on compliance will be explained in detail in Section 1.2).

To illustrate this problem in more depth, the basic means against fraud, namely authorization and SoD, are applied. Figure 1.8 indicates such exceptional situation in a PAIS with authorization. It illustrates how the introduction of security requirements such as authorization and SoD are applied onto a business process and how even further environmental constraints, for example the absence of users due to illness or vacation, significantly minimize the set of users that are able to execute the process.

This may finally result in a situation during process enactment, in which no user is authorized to execute a pending activity; the workflow is obstructed. It is key to distinguish between workflow and the overall process here, since a workflow aims at the flow of the process activities, rather than considering all process aspects. Indeed, the obstruction is caused by the organizational process aspect that restricts user access to execute process activities and finally block the workflow. This represents an exceptional situation, in which the process execution, or the respective case, is obstructed because security constraints and user availability block further execution. The enactment or execution of a process becomes impossible because there is a lack of subjects who can execute the process activities and are authorized to do so at the same time. Hence, although the business process security requirements can be automated, their enforcement creates a new kind of obstruction. These requirements can obstruct acting in the behavioral scope of business set by corporate governance (compliant behavior). Figure 1.7 indicates such an obstruction by an arc crossing the

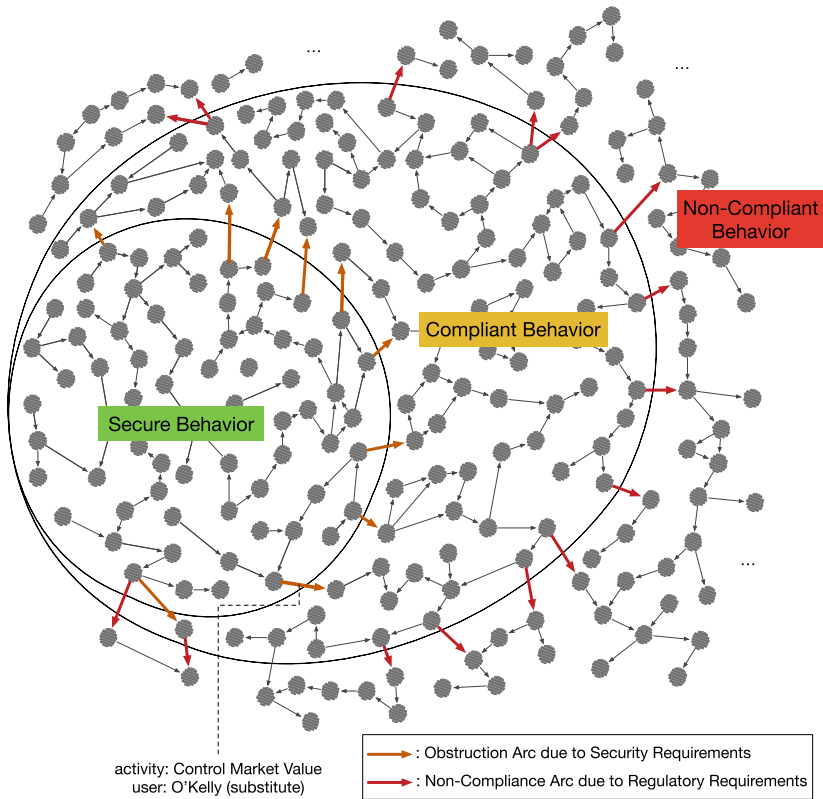


Figure 1.7 Behavioral scope of IT security on processes

border between the secure and the compliant behavior (see obstruction arc), indicating that it would violate security requirements. More specifically, the obstruction arc actually escapes the obstruction. The obstructed state itself is represented by the state preceding the obstruction arc. Similar constellations can also be established with isolation or usage control policies, for example, if no user is available to close an open file. However, due to its common application (e.g., in internal controls), the focus of this work is on security-aware workflows that consider authorization and SoD or BoD constraints, which is sufficient to cause such obstructions.

Etymologically, obstruction means that someone or something is prevented from moving in the direction it is trying to go. The Latin word “obstruere” means blocking

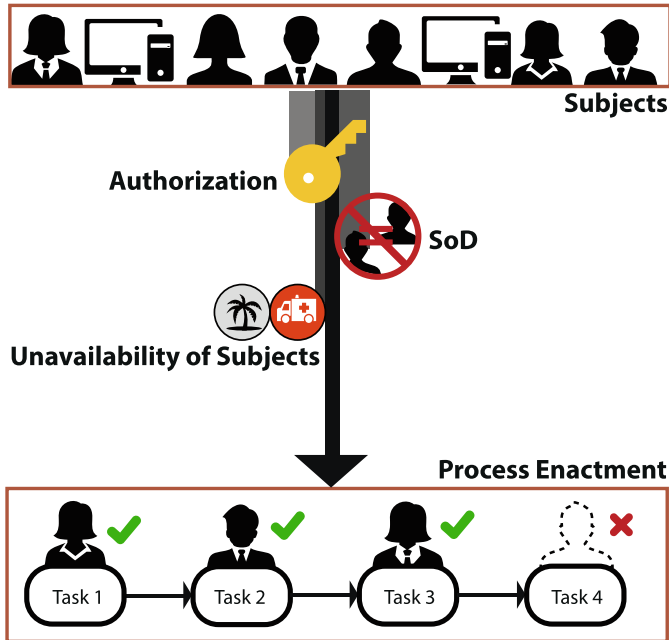


Figure 1.8 PAIS with policies and contextual influences

an access, to block up, to barricade, or building an obstacle. Interestingly, at the same time this means the object that is obstructed is not totally out of reach.

1.2 Conventional Anti-Fraud Measures

The introduction of IT security on processes can cause an obstruction on the technical level. However, this section shows that regulation itself does not necessarily create such obstructive situations or, if so, offers solutions or ways of circumventing them. This is due to the difference between the digital “world of correctness”, which also embodies classic IT security, and the analog “real world” from which regulation originates (cf. Figure 1.5).

The basic solution to the problem of fraud is clearly that there is a legislation that defines laws and penalizes unwanted deceitful behavior. In Germany, for instance, which has a civil law system (like most of EU countries), fraud and related facts

are codified in article 263 and the following of the criminal code (StGB §263 et seq.) that build the foundation for further regulation. On the other hand, U.S. law or further countries with Anglo-Saxon heritage like Ireland developed a law system that originates from the English common law, which is based on precedences.

In the AIB case, the CEO was convicted by unanimous verdict of the two offenses: first, conspiracy to defraud contrary to common law, secondly, false accounting contrary to section 10 of the Criminal Justice (Theft and Fraud Offenses) Act 2001 [16]. The latter consolidates the law relating to dishonesty, theft and fraud in Ireland. Hence, the definition of fraud in law is historically grown and depends on the legislative history of each country. Acts are understood as broad laws that are passed and the regulations are guidelines that dictate how the legal provisions of an act should be applied. Hence, the application of law is regulation.

1.2.1 Post-9/11 and Post-Financial Crisis Regulation

The financial crisis of 2007-08 especially underscores the importance of verifying that organizations operate “within their boundaries” [3]. Nowadays, enterprises have to face a continuously growing number of regulations and guidelines. Apart from the Basel Accords, which especially address banking regulations, the American federal statute SOX (Sarbanes-Oxley Act) of 2002 probably has the most significant impact. It requires an internal control system (ICS) for billing and external controllers to attest the effectiveness of the system, for example, to preserve the reliability of published financial data. The stir SOX has caused is evident by considering its worldwide adaptations, for example J-SOX (Japan), CLERP (Australia) or the EuroSOX (EU Directive 2006/43/EC on statutory audits of annual accounts and consolidated accounts). Moreover, international standards such as the ISO27k series of the International Organization for Standardization (ISO) have equivalents in SOX and vice versa. Regarding the AIB case, Irish legislation and regulation constantly adapted to the requirements arising from the financial crisis [147]. The offence of intentional falsification of accounts is also captured in EU market-abuse rules (with penalties up to ten years in prison and a fine of up to 10 million Euro) or the American Code of Laws (18 USC Sec. 1005, with penalties up to 30 years in prison and a fine of up to 1 million Dollar). One of the currently most prominently debated regulations is the General Data Protection Law of the European Union (GDPR), which came into force in 2018. It also emphasizes the need for companies to be able to demonstrate compliance with accountability and further principles regarding the handling of information. Enterprises, such as banks, have the duty to act upon these regulations and the governing law relevant in their economic areas and legal space.

This means that they also have the duty to take action for themselves to adhere to regulation and ensure that there is no violation of it. They need to integrate it into the way they lead and control their operations. Otherwise, legal action may be taken. In this respect, the increasing regulatory pressure in finance can be stressed as regulators have fined financial firms at least 28.4 billion Dollars for money-laundering and sanctions violations since 2008 and another 9.5 billion Dollars for aiding tax evaders.

Looking at the fraudsters in an enterprise, the ACFE identifies three main aspects that influence a potential perpetrator:

- Pressure, that is, financial or emotional motivation forcing towards fraud (e.g., AIB has a very negative balance acting as a deterrent for Government and EU),
- Rationalization, that is, personal justification of dishonest actions (e.g., “I will save the bank (and my position) and keep the Government and EU supporting the bank”), and
- Opportunity, that is, the ability to execute a fraud scheme without being detected (e.g., the management can override given fraud controls).

Certainly, entrepreneurial measures, for instance fair and proper salaries and a good code of conduct, may lessen financial pressure, strengthen ethics in a company and may make rationalization harder. These two aspects are mainly based on situational or personal reasons. However, the “opportunity” to commit fraud, underlines the need for regulation and its effective implementation in enterprises to establish enough measures and control such that the opportunity to commit fraud is ruled out or at least minimized. Hence, since corporate fraud takes place in the enterprise, regulation provides the following general concepts and controlling functions in business under the overall roof and concept of corporate governance.

1.2.2 Corporate Governance

Governance unifies the regulatory concepts on the entrepreneurial side and sets the frame how fraud measures are established in order to still support business operation. As identified, enterprises not only have an intrinsic interest in security to prevent fraud, but also an extrinsic motivation by given law and regulation. However, enterprises are also keen to comply with accepted standards in industry so they can claim to be compliant. Hence, besides implementing regulation and standards, organizations want to protect themselves and may profit by going beyond what is only “necessary” or “required from outside”.

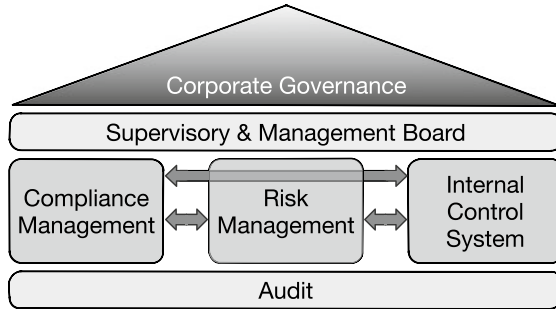


Figure 1.9 Compliance as part of corporate governance

Based on Schröder et al. [185], Figure 1.9 shows the bigger context in which subsequently elaborated components can be put, namely corporate governance. In general, corporate governance defines how enterprises and specifically processes are conducted in terms of leadership and control. It is the combination of culture, policies, processes, laws, and institutions that define the structure by which the organization is directed and managed. Compliance management, risk management and the system of internal controls build the three pillars of corporate governance, which aims at an successful supervision of an enterprise. These three pillars are monitored by the board of directors as the senior supervisory body [130]. Moreover, corporate governance requires the board to also establish an audit committee to review and assess the effectiveness of the internal control system, including financial reporting, but also risk and compliance management.

Internal Control Systems play an important role for fraud prevention. The U.S. Committee of Sponsoring Organizations of the Treadway Commission (COSO), which became renowned especially in the course of the SOX, defines: “Internal control is a process — effected by an entity’s board of directors, management, and other personnel — designed to provide reasonable assurance regarding the achievement of objectives in the following categories: (a) reliability of financial reporting, (b) effectiveness and efficiency of operations, and (c) compliance with applicable laws and regulations.” An important component of internal control is the control activities. They represent the specific policies and procedures that shall be applied on or extend business processes. Here, the ACFE report’s set of guidelineing

questions stresses the procedures and practices of internal control that are key against fraud, indicating especially the importance of authorization and SoD principles².

The essence of each regulation, the legal semantics of law, is decided through the courts, where regulation goes from theoretical legislation to practical rules for running an enterprise. **Compliance** means the fulfillment of such rules and obligations, resulting from law and regulation, but also guidelines, norms or corporate governance. Compliance rules initiate the respective policies and procedures resulting from internal controls. For instance, an SoD rule initiates the respective SoD control activity. Regarding the origin of these rules, the “Code of practice for information security controls” from the ISO 27002 standard postulates that it is essential that an organization identifies its security requirements. More specifically, this concerns “the legal, statutory, regulatory and contractual requirements that an organization, its trading partners, contractors and service providers have to satisfy, and their socio-cultural environment.” As with all regulation and compliance requirements, this underlines that organizational compliance is not considered to be an objective, impartial measure. In contrast, it is rather influenced or even biased by different stakeholders, economic interests and, more generally, politics and stems from the context or socio-cultural environment in which the law has evolved.

To assess how to handle the danger of fraud with controls, risk assessment is key. Trust, the (brand) reputation, shareholder value, regulatory compliance (fines, jail time), customer retention, privacy, staff morale or the ability to offer and fulfill business transactions are crucial factors which can be at risk for an enterprise. **Risk** is commonly understood as the product of the occurrence probability and the impact (or damage) of a given event. The ISO 27002 standard requires “the assessment of risks to the organization, taking into account the organization’s overall business strategy and objectives.” Moreover, “through a risk assessment, threats to assets are identified, vulnerability to and likelihood of occurrence is evaluated and potential impact is estimated.” Identified risk may be accepted (based on minor probability of occurrence or low expected damage), outsourced (for instance by insurance) or handled otherwise. Handled risks are relevant for the system of internal controls. An adequate control to reduce the probability of the risk to occur has to be implemented here, for instance, by adding a respective control activity to a process. The calculation of risks can be done with different methods, such as the failure mode and effects analysis (FMEA).

Auditing has the aim to evaluate enterprises and their processes. Audits are carried out to ascertain the validity and reliability of information about these

² Other measures identified therein such as job rotations and vacations finally also aim at the change of duties.

companies and the associated processes. In this way, it is checked whether the execution of business processes is done within certain boundaries set by managers, governments and other interest groups and stakeholders. Specific rules can be enforced by law or company policies, and the auditor should check whether those rules are being followed or not. Violations of these rules can indicate fraud, misconduct, risk and inefficiency.

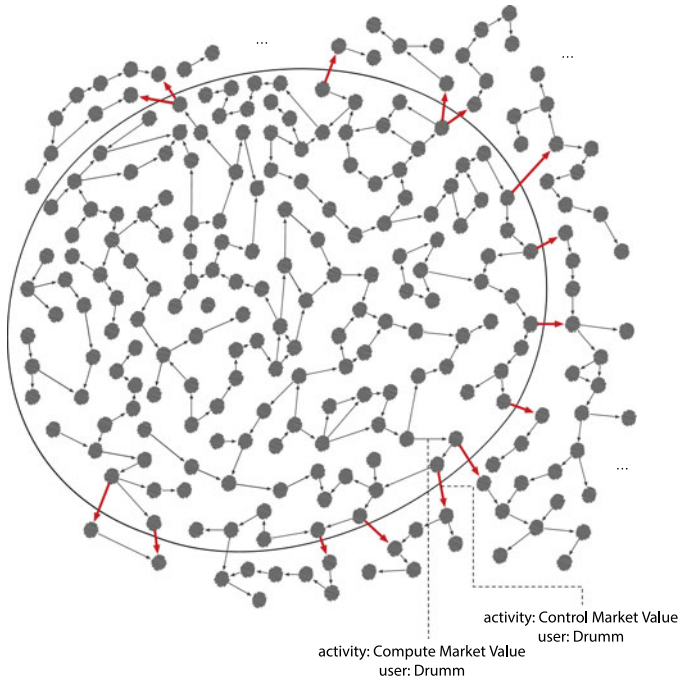


Figure 1.10 Behavioral scope of enterprise bounded by regulation (circle)

Given the concepts of regulation, the behavioral possibilities of an enterprise can be divided into compliant (see the area within the circle in Figure 1.10) and non-compliant behavior. Figure 1.10 indicates this behavioral scope restricted by regulation and complements the explanation on Figure 1.7. Thus, an arc crossing the respective border represents a non-compliant business activity. The German financial regulator, the Federal Financial Supervisory Authority (BaFin), for example, expressly requires the separation of duties in scenarios such as the approval of a loan

deposit, for example between activities for calculating and controlling the market value, because assets are affected (cf. IAA SoD Definition). Hence, performing this activity by the same person would represent non-compliant behavior (indicated as arcs crossing the circle). As indicated before, the area of secure behavior is indicated in Figure 1.7. Each obstruction arc indicates a part of a process execution here, in which IT security would block the process execution, whereas corporate governance would allow a solution to continue process in the scope of compliant behavior. Hence, from a compliance perspective, in such an obstruction the process can still be completed by a rational decision, balancing the potential harms of neglecting security against missing the business goals. Hence, an obstruction is the point in the process execution when a decision has to be made whether to stop the process execution or to find a solution based on regulation, risk, and corporate governance to still complete the execution. The frame set for such a solution on the business level is set by corporate governance; it is not about security.

1.2.3 The Need to Support Regulation by Automation

Despite the concepts of regulation to restrict the scope of behavior to avoid fraud, their implementation in businesses is deficient. In other words, the indicated boundary in Figure 1.10 between compliance and non-compliant behavior often does not exist in practice because the intended compliant behavioral scope is often not adequately implemented and controlled. In this respect, the ACFE report states that internal control weaknesses were responsible for nearly half of the frauds that were investigated. In particular, survey respondents were asked for their perspective on the internal control weaknesses at the victim organization that contributed to the fraudster's ability to perpetrate the fraud scheme. More than 30 percent cited a clear lack of internal controls as the primary issue, another 19 percent stating that internal controls were present but had been overridden by the perpetrator. Here, overriding may set a so called "red flag" indicating a potential suspicious fact for further fraud investigation. In this respect, management review describes the process of management reviewing organizational controls, processes, accounts, or transactions for adherence to company policies and expectations. The other half is led by another 18 percent that were owed to a lack of such management review.

To some extent, this overall lack and override of control and the missing review can be all attributable to the obstructive nature of the implementation and enforcement of regulation in practice. Here, one can differentiate between enforcement from outside the enterprise (e.g., high efforts and costs of law enforcement in the AIB case) or inside the enterprise (e.g., internal audit). The likelihood of regula-

tion obstructing business is even growing with the ongoing digitization. Whereas business processes are more and more automated, regulation is often not. Regulation is frequently still conducted manually or is only partially computer-supported. According to the ACFE, only one percent of the detected fraud is initially detected by IT controls. The majority of fraud schemes is revealed by hints (40 percent), internal audit (15 percent) and management review (13 percent). In practice, internal auditing teams can obstruct operational business for month [92]. Traditionally, auditors can only provide reasonable assurance (cf. ICS) that business processes will be conducted within the specified boundaries. Based on the aims of internal control, they check the controls' operative effectiveness designed to ensure and maintain reliable processing. If these controls are not in place or otherwise do not work as expected, they typically only check samples of factual data, often in the "paper world" [3]. In this way, they are not able to completely check all relevant process data and therefore do not provide the full picture, such that suspicious activities may not be detected. In automated processes, the manual application of regulation intervenes even more drastically in the otherwise automated business operations. Today, event logs or audit trails, transaction logs, databases or data warehouses record detailed information about processes such that implementations of processes produce event logs of considerable size in practice. For example, around 10,000 loan applications were submitted to a major European bank over a period of 6 months. Because the resulting event log contains more than 200,000 events, a manual analysis of the respective data is no longer possible [40]. Moreover, as the ACFE results propose, the huge amounts of data provided by automation are not adequately used in manual checks. Hence, in the face of process automation, manual fraud controls are too slow, obstructive or simply missing.

The fullness of today's regulation, automated processes and big data demands the automation of the implementation of regulation in order to use the potentials that lie in automation, for instance to digest all necessary process data. To speed up its application and counteract its obstructive behavior in terms of delaying business operation and demanding high monetary and temporal effort, the logical and necessary step is to automate regulation that acts upon the business processes as well.

1.3 Obstruction by Security?

Despite the deficits identified in the application of IT security on processes, in particular the appearance of obstructions at the technical level (see Section 1.1.3), there is a need for automation of regulation. Because the behavioral space of corporate

governance permits more behavior than that of IT security, this should also be taken into account in the automation of regulation.

1.3.1 The Dilemma of Process Security in Following upper Policies

IT security originates from controlling access over data, not process-oriented activities. For instance, if access to data in a database has an error, the transaction is rolled-back. In contrast, unlike in database transaction processing, security policies in processes may not be fully checked before granting access, due to missing information of other related processes such as inter-instance process dependencies or further context specific data, such as user availability. Regarding databases, the ACID criteria (atomicity, consistency, isolation and durability) are not applicable for such long-lived transactions like a business process. Although there are even scenarios and also concepts in transaction processing of rolling back long-lived transactions, this cannot be in the interest of a business process execution.

In analogy to this and the need for a business process to run through, a short excursus is given by means of an example from sports. Since 2018, there has been a video referee at the Tour de France, the Spanish Vuelta and other famous cycle races who can intervene retrospectively. This video commissioner has access to all cameras and can review and evaluate any situation at any time, rewind, zoom in and assess if a first suspicion is justified, for example a rider pushes a rival aside. Of course, you cannot stop the race at the Tour de France. Consequences can only be taken after the race; after crossing the finish line. Then, a team of race control and referees looks at the respective selected material and then, it has to decide what to do. Hence, during the race there is only a suspicion. After the race, the decision is made and consequences can be taken. Ideally, one would like prompt, binding decisions here. Analogously, business processes need to operate and only after “crossing the finish line” or reaching the process goal can the security be fully assessed. However, if there is a problem, one would like to promptly identify it and act accordingly. This could mean to change the process or take further steps after an identified violation. This could clearly also mean that damage already occurred, if for example payments have already been done during the process.

Based on these considerations, the assumption security is based on changes. Indeed security in business processes should not only consider the protection of objects like in classic computer security but also take upper policies set by corporate governance into account. This represents a paradigm shift; moving away from trying to achieve or sustain protection goals, such as confidentiality and integrity,

or simply “keeping bad things from happening” towards an end-oriented view on security in business processes where not so much a violation of security is a problem but at the end a compliant result must exist to eventually “make good things happen”. Hence, whereas classic security stresses the safety property, business process security emphasizes the liveness property.

1.3.2 Indicator-Based Process Security

Altogether, the classic understanding of security does not fully apply to the context of business processes. The concept of classic security clearly builds the essential basis for business process security. However, based on the identified reasons, this so far policy-based classic security concept needs to be extended. Common ways of compliance management to resolve obstructive situations are setting red flags or delegation. If there is an obstruction due to an SoD conflict imposed by regulation, it is possible to assign substitutes that do not have the managerial level of the initial assigned users. Then, the delegator indicates an appropriate substitute based on his risk assessment of the delegatee. This represents a compliant solution to resolve an obstruction, for example based on the BaFin regulation (Finanzdienstleistungsaufsicht [94]). Considering best practices of internal auditors, it is a well-known means to set red flags for fraud, for instance when a control is overridden due to an obstruction. Red flags are signs that indicate both the inadequacy of controls in place to deter fraud — in the sense of not being effective but also in obstructing the process — and the possibility that some perpetrator has already overcome these weak or absent controls to commit fraud. These indicators can then be used for detection and prevention and ultimately aim to lessen the risk of the execution of business processes. Hence, indicators allow to reduce the risk of a fraud to happen and at the same time support the completion of the process. This can also be stressed by looking at the etymological meanings of the respective concepts. Etymologically, “to comply” means to fulfill something and originates from completing (lat. *com-ple-re*), i.e., bringing something to its very end. Also the conformity to rules is often named equally to compliance, meaning to be in accordance or strong similarity to something. This notion generally is less strict than the meaning of security, originating from the Latin terms “*se-*” (engl. without) “*cura*” (engl. care) and can be translated as “free of care”. In the literal sense, the inherent goal of security is to accomplish a carefree state, i.e., to avoid a state that may cause worries. Hence, whereas security literally tries to avoid risk, compliance tries to keep given rules while allowing to take a certain amount of risk and worries into account to fulfill the overall goal of completion (or “bringing it to its very end”).

Due to the limitations of classic security in processes, the policy-based security concept is supposed to be enhanced with an indicator-based view, thereby allowing to better represent and capture the frame set by corporate governance. The area of secure behavior is supposed to be widened towards the compliant behavior (or shifting the respective border in Figure 1.7). Using indicators in business processes is not uncommon, for example Process Performance Indicators (PPI) or Resource-Behavior-Indicators (RBI). The focus of this work is to use indicators in order to find a security-sensitive solution to obstructions.

Table 1.2 displays different works in the Business Process Security Group in Freiburg that contribute to this perspective on security. For instance, Lowis analyzes compliance based on given rules and workflow models. Stocker [189] provides a way how to reconstruct process models from event logs that are able to indicate rare deviating suspicious behavior. Moreover, Zahoransky [215] uses simulation to investigate if control activities imposed on single core processes are a problem in performance in the context of entire business process architectures (cf. PPI). Syring [193] defines a framework how to decode legal code into (semi-)formal rule that are machine-readable and can be applied in an indicator-based way. Wonnemann [214] proposes a way to check (unwanted) flow of information based on models. All these approaches aim to automate the implementation of regulation to some extent. However, they have so far not addressed the problem of runtime obstructions during process enactment in processes and its practical consequences and how this conflict between business and security goals should be handled.

Table 1.2 Comparison of Freiburg approaches addressing security before, during and after process execution: ✓ addressed, (✓) partially addressed, ✗ not addressed

	ex-ante	enactment	ex-post
Deviations (Stocker)	(✓)	✗	✓
Performance indicators (Zahoransky)	✓	✗	(✓)
Rule Formalization (Syring)	✓	✗	✗
Information Flow (Wonnemann)	✓	✗	✗
Compliance (Lowis)	✓	✗	✗
Obstruction	(✓)	✓	(✓)

1.4 Contribution and Structure of Thesis

Automation is no panacea against regulation acting obstructive. Adequate and effective application of regulation builds the basis against fraud. However, there is no way to avoid automation when IT methods are used to generate competitive advantages. An obstruction results from introducing IT security on business processes, particularly authorization and further security policies such as separation of duties. Handling situations that obstruct the business process are key, because, besides the business harm, not handling them in an automated manner would contradict the overall intention of automating security to speed up the secured process. Hence, the aim of this thesis is to specify obstructions on a technical level and provide automated security-sensitive solutions that take the frame set by cooperate governance into account. By learning from obstruction handling in corporate governance, indicators are identified and an indicator-based security is proposed, which allows to benefit from process automation and process event data. In detail, this dissertation provides the following contributions³:

- This work will comprehensively examine the conflict between the goals of IT security and business process executions and present the concept of an indicator-based security in business processes.
- With the systematic examination of existing literature on security-related obstructability in PAIS concerning the process design, runtime, and auditing phases, requirements to analyze, detect, and handle obstructions will be derived.
- With the SecANet approach, an encoding technique that can capture all aspects of a security-aware process into a formal representation and integrate indicators as costs to explore security-sensitive behavior will be systematically developed.
- The model-based OLive-M approach represents a SecANet-based technique to revive an obstructed workflow and find a security-sensitive way to complete its execution.
- With OLive-L, this work presents an approach to handle an obstructed execution trace of events based on a SecANet and a log containing information on past executions.

Figure 1.11 shows the structure of this work. Chapter 2 examines the state of the art with regard to security-related obstructability in PAIS, such as workflow sat-

³ Parts of this work (in particular, with regard to the identification of the research field, the basic approach idea of how to detect, capture, and resolve obstructions, and the tools used) have been published and can be found in [9, 106–109, 216].

isfiability and resilience. Thereby, requirements for analyzing, detecting, and handling obstructions will be derived along the phases resulting from the enforcement of security properties and the BPM lifecycle. Besides introducing the notion of “obstructability” and “completability” of security-aware workflows, various possibilities of how, in particular, logs can be used to obtain indicators will be examined.

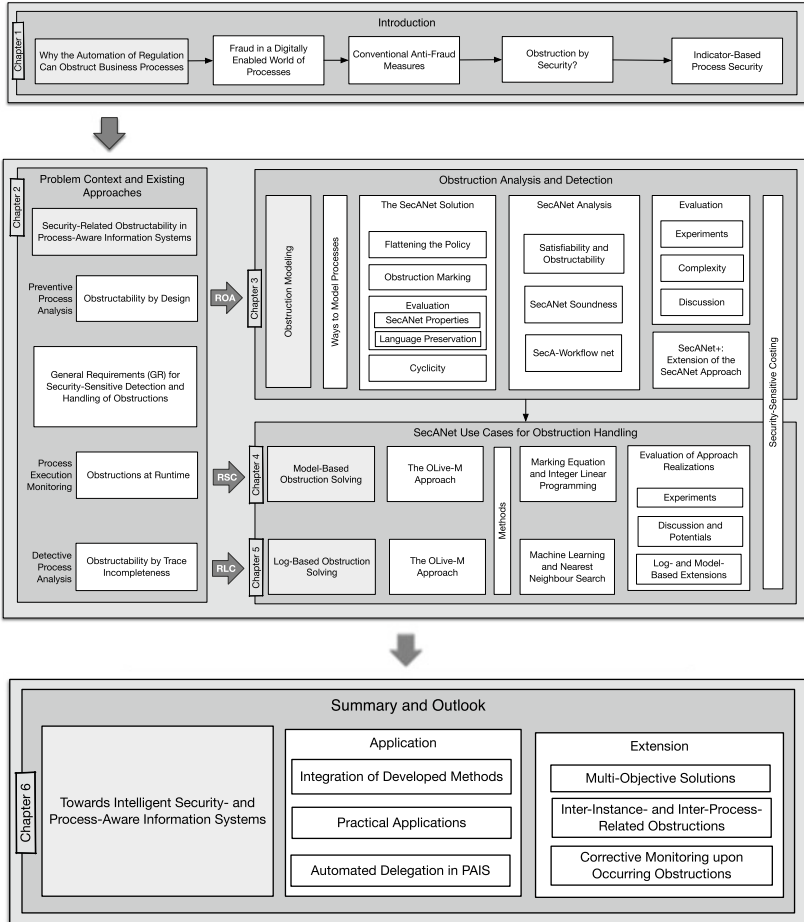


Figure 1.11 Structure of thesis

Chapter 3 builds the central part of the thesis. In addressing the requirements for obstructability analysis (ROA), it is primarily concerned with the modeling of obstructions in security-aware workflows. For this, it will first explore the ways to model processes, in particular, the modeling of the control flow. Based on the selection of an ordinary Petri net model (P/T net) as the method of modeling, a security-aware Petri net representation, the SecANet, will be developed. The general idea of the SecANet modeling is to “flatten” the organizational aspect of a security-aware process specification that subsumes the process model and the policy specification into the model representing the functional and behavioral aspects. That way, a so-called “obstruction marking” can identify and capture obstructions. Moreover, indicators can be assigned to the task- and policy-related Petri net elements as costs. For the evaluation of the SecANet approach, its Petri net-specific properties will be investigated. Moreover, the formal correctness of the SecANet method will be proven by examining the behavioral preservation of the original inputs (i.e., language preservation). To take cyclic workflow behavior into account, the concept of policy re-enactment will add cyclic functionality to the so-far acyclic SecANet encoding. The developed SecANet formalism can be regarded as a target metamodel of security-aware process specifications that creates a basis for further analysis. Accordingly, Section 3.2.6 will subsume SecANet-based satisfiability and obstructability checks as SecANet soundness and show further extensions to facilitate analysis. An experimental evaluation will compare soundness checking runtimes with typical satisfiability-related approaches. The discussion will then examine the computational complexity of the SecANet encoding and sketch possibilities for extensions. With the introduction of SecANet+, the integration of additional inputs that further constrain the execution of tasks, for instance, counting constraints or user absence (resilience), will be made possible. Chapters 4 and 5 will use the SecANet encoding to handle and resolve obstructions. Across all these chapters dealing with approaches and solutions within the described problem context, the security-sensitive costing will play a crucial role in enabling the consideration of indicators. In order to address the requirements for specification-based completability (RSC), Chapter 4 proposes a model-based technique to complete obstructions. Thereby, the OLive-M approach will interpret the question how to complete an obstructed process as an optimization problem. Based on the examination of suitable methods, an integer linear programming model that optimizes the SecANet marking equation and its security-sensitive costs will realize the OLive-M approach as a way to deal with obstructions. In contrast to the exhaustive exploration of all possible markings (i.e., computing the marking graph), the experimental evaluation will underline the “lightness” of this technique to resolve obstructions. The discussion will then indicate how the approach can be used to also analyze sat-

isfiability or resilience. Chapter 5 focuses on log-based completability (RLC) and considers the obstruction as an execution trace in the context of process event logs. The realization of the OLive-L approach will incorporate methods of process mining and machine learning techniques to propose solutions that represent the nearest match of successful executions to escape an obstructed state. To show the applicability of the OLive-L approach, the evaluation offers experimental results based on event data synthesized from SecANet execution sequences. In the course of the discussion it will be sketched how logs may be used for the model-based approach and vice versa.

Chapter 6 summarizes the work and explains its significance by considering the contributions and its applicability. The work concludes with extensions that could be envisaged.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





Security-Related Obstructability in Process-Aware Information Systems

2

“The practicality of any security policy depends on whether that policy is enforceable and at what cost [182].” What Schneider stated in his seminal work on enforceable security policies also applies to the conflict between security and business goals in business processes. In classic IT security, enforcement mechanisms work by monitoring execution steps of some system, often called the target, and terminating the target’s execution if it is about to violate the security policy being enforced. In order to operate a Process-Aware Information System (PAIS) according to given policies, mechanisms that implement the policies and control their enforcement are equally necessary. However, if such mechanisms are used in a PAIS, this is opposed to the achievement of business goals, as enforcement may “cost” the completion of the process execution, such that the process does not generate the expected value to the company. Technically, it is this enforceability that enables the process execution to become obstructed. Hence, an obstruction is the case when enforceable policies conflict with the goal of process completion, or, put differently, enforceability implies obstructability. The first chapter identified that detecting, preventing and handling the obstructability resulting from the implementation of an automated regulation, is an important problem. It showed that a solution must provide an indicator-based view on security that enables obstructed executions to be completed. In this respect, the concepts of governance, risk and compliance allow for greater room for maneuver than classic IT-security and should be considered when solving obstructions. The actual costs of enforcement may therefore take into account not only the financial loss due to process stop but also the risks if the process is nevertheless completed, for example if tasks are delegated, executed by unauthorized users or if SoD rules are violated. Based on such considerations a rational decision has to be taken whether and how to continue the process or to stop, or, in relation to Schneider, to balance enforcement and its costs.

This chapter aims to relate this conflict to the existing state of the art. It will therefore first introduce concepts related to security properties and their enforcement to better grasp the problem of security-related obstructability in PAISs. Based on this and the BPM lifecycle, a structure will be derived along which the analysis of related work will then be conducted.

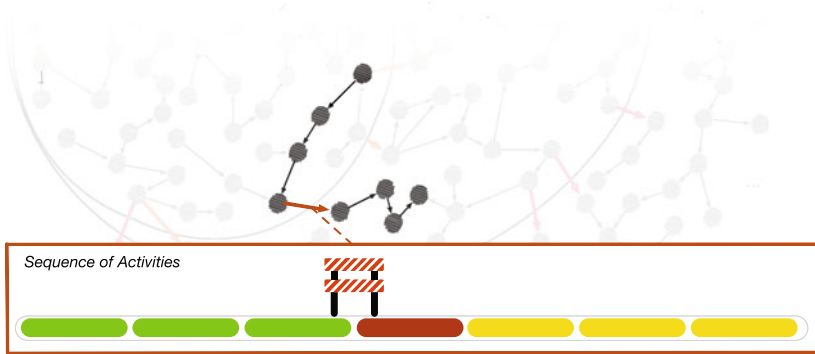


Figure 2.1 Obstructive sequence of activities

To begin with, it can be observed that the identified conflict between security and business goals occurs on the level of a single process execution that eventually becomes obstructed or runs through. Figure 2.1 exemplarily highlights an activity sequence of such a single execution, based on the executions encoded in the representation of the behavioral scope of a process in a PAIS. Based on the introduced behavioral areas of a process in Chapter 1, the sequence encompasses activities in the scope of secure behavior (the first three activities) and in the scope of compliant behavior (the last three activities). Moreover, there is an activity (the fourth activity) that escapes the obstructed state (indicated as a barricade sign). Hence, the representation of the obstructive activity execution provided in Figure 2.1, embodies secure behavior, which would actually not allow to complete the execution of the process securely and would therefore cause the PAIS to block further execution. However, it also contains the complete successful execution that is enabled by escaping the obstruction in a still compliant way. In relation to the BPM lifecycle, such an activity sequence can be determined on the basis of the process specification as a task execution sequence. It occurs during runtime in a case, or can also be found in logs as a so-called trace.

The conflict between security and continuity in operation is not new but it now manifests itself no longer at the infrastructure or application level but at the business level, which involves the business processes, regulatory rules implemented in PAIS and the process stakeholders. The basic security concept that can be applied to these architectural layers was first introduced in the context of language-based security (LBS [183]) by so-called security **property** classes focusing on the application layer. Because a process can also be seen as a small application—in fact, a workflow can be understood as a business process representation on the application layer—this view is adaptable to the business layer as well, which manifests itself in the common notion of an execution sequence or, equivalently, a trace for both layers. In order to be able to reason on whether system states satisfy specific security requirements, the notion of “properties” is used. The focus on these properties is supposed to help to reason on their impact on the process execution and the desired outcomes for the detection and handling of obstructions throughout this chapter.

The notion of trace properties can be used to specify basic types of security classes. Because it will be sufficient for the subsequent analysis of the state of the art to informally show the general concepts and intuitions, it is referred to a more formal definition of properties and the related property classes to the seminal works of Lamport, Alpern and Schneider in this field [13, 132, 182]. Informally, a property itself is encoded by a set of execution sequences. Such trace represents an execution of an abstracted system as a sequence of events. The observation of these execution sequences allows to reason on their properties, such that traces can be identified for which a specific property holds. In other words, the set of sequences that hold this property, builds a subset of the set of all sequences. As introduced in Chapter 1, there are two basic property classes, namely safety and liveness. A **safety** property stipulates that no “bad thing”, i.e., the violation of a property, happens during the execution [13]. It describes states that should never be reached. This means, if there is a certain sequence that can be appended to a given execution sequence, such that the property does not hold anymore, it is a safety property. There is a finite prefix in the execution sequence at which the violation can be detected (discreteness)¹. Typical safety properties are confidentiality or integrity (keeping a private key secret or mutual exclusion). If a violation of the property happens in an execution sequence, this execution can then not be remedied for this property or, in other words, there is no chance to fix it. Hence, for safety considerations, partial sequences are sufficient and, if a safety property is violated, this happens in finite interval of time. For example, suppose the safety property that the two activities "compute market value" and "control market value" shall be executed by

¹ A safety property can be proved using an invariance argument [14].

different people. If there is a sequence in which both activities are performed by the same person, this sequence does not hold the safety property anymore. On the other hand, a **liveness** property, as introduced by Alpern et al. [13], stipulates that a “good thing”, i.e., the fulfillment of a property, eventually happens during the execution. Typical liveness properties relate to availability, for example, guaranteed service, starvation freedom (i.e., a process makes continuous progress) or process termination. This means, if there is a certain sequence that can be appended to a given execution sequence, such that the property holds, it is a liveness property. A liveness property cannot stipulate that a “good thing” always happens, only that it *eventually* happens². Unlike safety, liveness does not require the “good thing” to be discrete. It refers to states that must be reached at some time in the future, and it means that something good will happen sometime. In other words, it is always possible that the “good thing” will still take place and it is not sufficient to consider partial sequences to assess liveness. Given, for example, a partial execution of the collateral evaluation process where a user has not (yet) executed the activity “approved acquisition”, it is always possible to extend a partial sequence in a way that a user executes the activity by appending a sequence with this activity to the existing sequence.

In relation to the identified activity sequence in Figure 2.1, Figure 2.2 sketches a liveness property, in particular the reaching of the end activity, and a safety property, in particular an existing SoD constraint between two events (or activity executions respectively). It depicts that, for a partial trace (here, the first three events) the safety property holds. In contrast, the liveness property holds for the whole execution sequence. In this way, it is now possible to capture secure and compliant behavior with the respective property classes.

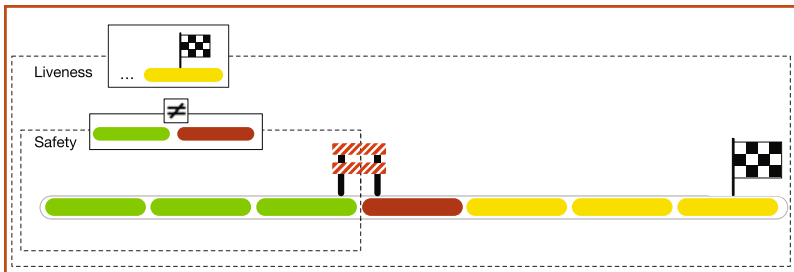


Figure 2.2 Intuitive example of property classification for an obstructive sequence

² A liveness property can be proved using a well-foundedness argument [14].

The differentiation between safety and liveness has further implications. Depending on the property class to which a property belongs, it can be enforced, such that the violation of the property can be prevented, which means the execution of the target system to which it applies, is stopped [14]. As it is this enforcement and its implications that cause the occurrence of obstructions, a closer look at the respective enforcement mechanisms is followed. This step will enable a better insight into the causes, time-points and forms in which obstructions can manifest themselves.

Schneider showed that only safety properties are enforceable whereas liveness properties are not [182]. Mechanisms to enforce safety properties are classically divided into static analysis, execution monitoring and program rewriting [103]. Schneider describes static analysis as an enforcement mechanism that strictly operates prior to running the untrusted program. The idea of static analysis is that, after the analysis, accepted programs are permitted to run unhindered while rejected programs are not allowed to run at all. Reference monitors [15, 210] and other enforcement mechanisms that operate alongside an untrusted program are termed execution monitors [182]. Program rewriting refers to any enforcement mechanism that, in a finite interval of time, modifies an untrusted program prior to execution. This chronological differentiation may be well-applied on the business layer as well. Instead of programs, the object to investigate are processes. Analogously, depending on the time of enforcement, the policies that can actually be enforced, differ. For instance, a separation of duties constraint, which depends on the actual execution history of a process instance (also known as “dynamic SoD”), can only be enforced at runtime. If the set of executions for a security policy is not a safety property, an enforcement mechanism for the class of properties an execution monitor can enforce does not exist. On the other hand, the converse—that all safety properties have enforcement mechanisms for execution monitoring—does not hold (for example for the Maximum Waiting Time (MWT) regarding time interval) [182]. In this respect, Schneider already identified that his conditions for enforceability are necessary but not sufficient. To address this, based on similar monitors that observe system actions and terminate systems to prevent policy violations, Basin et al. [26] distinguish between actions that are only observable and those that are also controllable. Their enforcement mechanism cannot terminate the target system when considering an only observable action. In contrast, it can prevent the execution of a controllable action by terminating the system [26]. However, because the only observable cases are rather encompassing policies related to time, for example meeting deadlines, or administrative changes, such only observable policies will not be in focus of this thesis. In fact, it is the cases of controllable policies that may obstruct executions. Therefore, it is sufficient to stay with the basic notion of enforceability given by Schneider. The terms of Basin et al., however, are occasionally used for

clarification. In this respect, the enforcement of any enforceable safety property can lead to an obstruction, because, in order to avoid a violation a reference monitor per se is always able to block an execution. In this thesis, this obstructability is considered for those enforceable safety properties that are relevant with regard to the security requirements for business processes identified in Chapter 1.

According to the points of time of an enforcement mechanism, enforcement of security properties from a purely chronological point of view only makes sense by means of a preceding preventive analysis before the execution, or monitoring during the execution. After the execution, the result of it must be accepted as it is and properties can not be actively enforced anymore nor change the status quo. However, narrowing the focus only on the phases relevant to enforcement would neglect a significant portion of the possibilities offered by process logs. A log allows to check for safety and liveness properties, and this, for the actually “lived”, thus practically relevant processes. More specifically, the log is also the only way to check whether processes have really been completed in practice, in other words, whether the possibility of completion, the desired liveness property, was actually perceived in the process being performed. Hence, for a holistic detection and handling of obstructions, all three phases of the process execution are important. Then, depending on the regarded process entity, there are different possibilities to treat or avoid potential damage resulting from obstructions: Before the execution, the design of involved processes can be addressed. During the execution, runtime monitors are in control and obstructive situations can be detected and handled, and after the execution, damage may already have occurred but can be determined, for example, by auditing, such that its effects can be mitigated subsequently. Further, the log can also be used to learn for the handling of obstructions based on the history of the process.

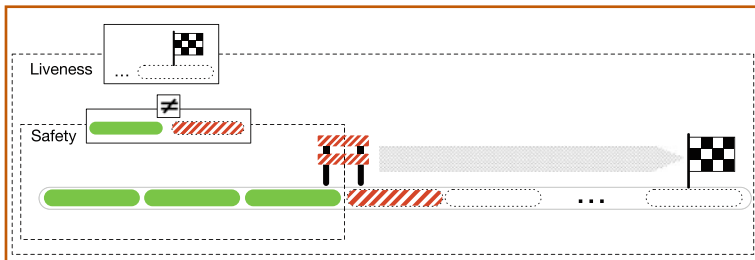


Figure 2.3 Properties encoding the cause of obstruction (safety) and the desired outcome (liveness)

In conclusion, before, during and after the process execution different aspects of safety and liveness manifest themselves. Figure 2.3 abstracts from specific obstructive activity sequences and depicts how these complementary property classes set the frame to analyze and handle obstructions. In a sense, these classes capture the cause and the cure for obstructions at the same time. On the one hand, enforceable safety properties cause obstructions. Their consideration allows to reason on the detection and avoidance of obstructions and how policies may be improved. On the other hand, liveness encodes the property of process completion, which describes the aim to escape and resolve an obstruction. Their consideration allows to find out when obstructions occur, that is when the liveness property is not fulfilled (falsification), or when there are no obstructions, which can give hints on how obstructions can be fixed.

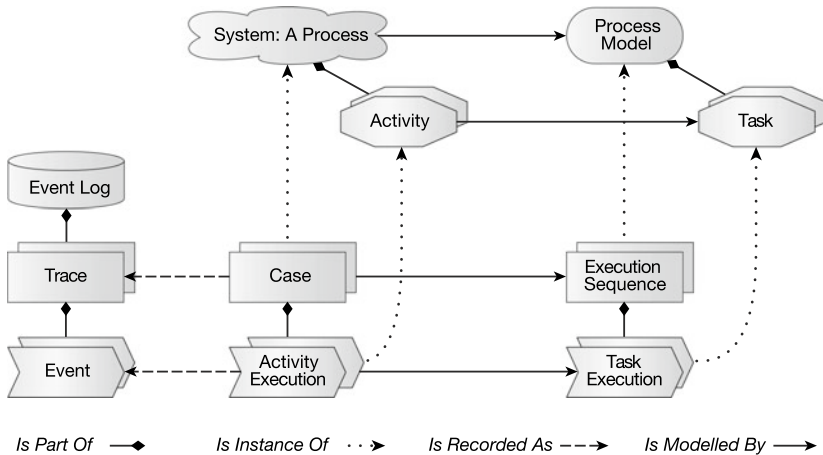


Figure 2.4 Terminology of processes, event logs, and models [40]

Based on Figure 2.4, Table 2.1 relates the identified different kinds of enforcement and security perspectives to the BPM lifecycle and its process entities. Preventive and detective mechanisms [33] work on the process model or the log respectively. Execution monitoring observes the process instance at runtime. Based on this structure, the subsequent systematic analysis of the state of the art determines deficits and potentials for development for the detection and treatment of obstructions, which then lead to their reformulation in the form of requirements, which a

solution should consider, with regards to the goal of this thesis: to escape an obstructive situation that was caused by a safety property and find a solution that restores liveness, that is, to allow for process completion. In all of this, the common terminology and policies, problem cases and developments in the related fields will be grasped, such that they can be applied and adapted or that they can inspire solutions to obstructability. The requirements for the three approaches of this thesis will be derived from the three individual areas arranged according to design, execution and audit time. Firstly, the analysis of the state of the art for preventive analysis will determine the requirements for a specification of a model that covers all relevant process aspects and can be of added value in practice. Based on this, the requirements from execution monitoring will then be identified for the approach to resolve obstructions. Finally, with the analysis of log-based approaches and its potentials regarding obstructability, the basis is laid to beneficially involve logs to identify and resolve obstructions.

Table 2.1 Process terminology from Figure 2.4 [40] related to BPM lifecycle phases and enforcement mechanisms

<i>Mechanism:</i>	Detective	Monitoring	Preventive
<i>BPM-Lifecycle Phase:</i>	Evaluation	Enactment	Design & Analysis
<i>Process Terminology:</i>	Event Log Trace Event	System: A Process Activity Case Activity Execution	Process Model Task Execution Sequence Task Execution

2.1 Preventive Process Analysis: Obstructability by Design

The preventive analysis takes place prior to the actual process execution and is related to the “Design” and “Analysis” phase of the BPM lifecycle. The idea here is to first analyze the effects of policy enforcement a priori execution to find out, if the process is obstructable. For this, this subchapter will first consider the process specification, which embodies the different process aspects. Based on the process model and the process policy, it will be shown that obstructability foremost analyzes possible conflicts between the organizational and the functional and behavioral aspect of the process, namely if there is a user-task assignment that is able to obstruct the execution of the control-flow. This question relates to other questions that arise with the enforcement of policies in security-aware workflows, more specifically, questions

regarding their satisfiability and resilience. The so-called workflow satisfiability problem (WSP) asks whether there is a mapping of users to tasks in a workflow so that each task can be executed and the policy can be followed. Moreover, obstructive situations may also arise due to reasons beyond the policy. For instance, if there are exceptional situations due to illness of staff or further unexpected unavailabilities. The notion of resilience (or resiliency) is used to describe such scenarios, asking how many users can be absent (or are likely to be absent) such that the policy can still be followed and each task is still executable. To some extent, WSP and resilience analysis imply obstructability analyses. For example, an unsatisfiable workflow is obstructable as well. However, a satisfiable workflow can still contain obstructions. The particular differences of these notions will be elaborated in more detail in the subsequent sections. Due to this interrelation, and to allow for comparability, this subchapter will mainly elaborate on satisfiability and resilience problems, which will sometimes implicitly relate to obstructability analysis as well. However, it will also reveal that, so far, explicitly analyzing obstructability has only marginally been considered in related work. The sections on satisfiability and resilience will therefore highlight important findings and deficits from related research and extract criteria that will be key for the preventive analysis of obstructability.

2.1.1 Process Specification

The design of the process manifests itself in its overall specification, because a PAIS is steered by the process specification. It builds the basic reference to preventively analyze policy enforcement in a PAIS. However, strictly speaking, because the term “process specification” is often intended to only describe the control-flow, it does not represent the whole process. Rather, there are different specifications that encode different parts of the different process aspects, which were briefly introduced in Chapter 1.

The process model is typically used to capture the functional and behavioral aspects of the process. In particular, it specifies the control flow to capture the behavior and includes the functional components of the process, for example in the Business Process Model and Notation (BPMN) language (which is used for the example in Figure 1.3). This overall composition encodes the business goal, which is a central part of the functional aspect.

For the other aspects, in particular for the organizational and informational aspect, further specifications are typically used. They describe policies concerning who is involved in the process and how data may be accessed or how it may flow, for

example in the form of security models, such as ACL, RBAC, Bell-La-Padula or Chinese wall. These policies will be subsumed under the overall policy.

2.1.1.1 The Process Model

As outlined in the right-hand part of Figure 2.4 a process model consists of tasks, where each task represents an activity of the process and its respective execution dependencies, that is, an activity literally is a task that is actively executed by a user. Activities themselves are conducted sequentially or in concurrent order. The execution of activities can depend on decisions such that there are branching situations that cause the process execution to follow different paths. The decision on the direction to take can base on process-related or contextual properties, such that process activities can be linked to conditions and obligations in different ways. Further, parts of a process may be conducted multiple times. A process model can also be instantiated. This means that an execution sequence of a process model consists of task executions, which enables a design-time representation of a case of the process.

A process model primarily stands for the business process and the business goal to be achieved. Security requirements of the functional and behavioral aspects mainly focus on the relationship between the activities. For example, they require that for the evaluation of a collateral, a market value computation must always be carried out and that in any case a check of this computation must only happen afterwards. Many such requirements can be covered by the explicit definition of the control flow in the form of process models. They can be checked with so-called patterns to facilitate re-use [169]. Patterns also build the bridge to other security policies, or in general safety or liveness properties, for instance expressed by means of Linear Temporal Logic (LTL). Such patterns can check if a given control flow supports a required behavior. For secure process execution, it is important to strictly comply with the requirements on the control flow layer.

The control flow requirements can be specified in various ways. Rule-based modeling follows an Event-Condition-Action-approach (ECA) to define requirements along business rules. However, business processes are usually defined with the help of graphical models such as UML [116, 117], Event-driven Process Chains (EPCs), BPMN [118] (as in Figure 2.5) or Petri nets. These process instructions explicitly define valid process activities and the order in which they must be executed. Within PAISs, such models can be used to automatically assign users to activities and monitor execution. In BPMN, the de facto standard in process modeling, tasks are represented by rectangles. Immediate events are visualized by circles, for example, that start or end the process as shown in Figure 2.5. Execution dependencies are modeled by control flow arcs and diamond-shaped nodes, which are called

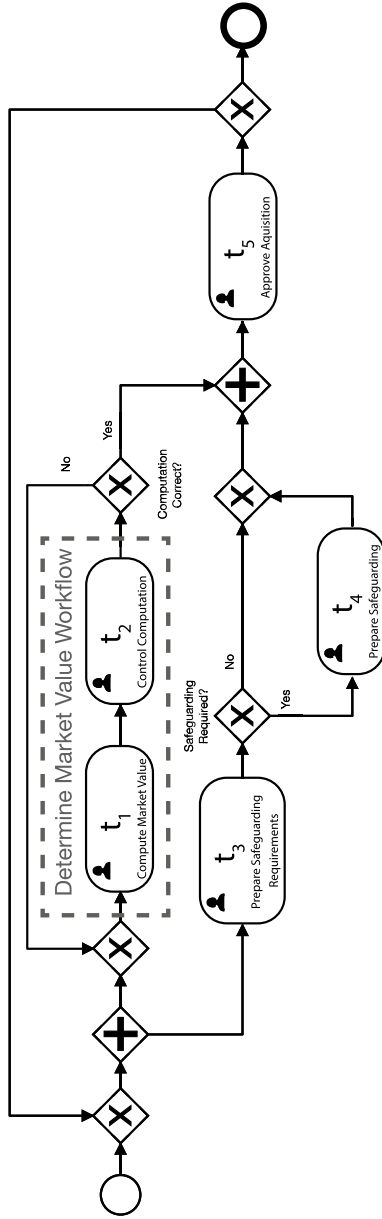


Figure 2.5 Highlighted sub-process to determine the market value of a collateral

gateways. The gateway semantics determines the exact process behavior. For example, they determine whether or not the incoming arcs are synchronized (AND or XOR gateway with a plus, or cross symbol respectively). Further, for outgoing arcs, they determine whether they are activated concurrently or mutually exclusive (AND or XOR gateway) [40]. For further illustration, the CEW example is simplified and only considers the subprocess of determining the market value of a collateral, namely its computation and control. This sub-process will be called the “Determine Market Value” (DMV) process depicted on the left in Figure 2.6b. A subject computes the market value (t1) and afterwards, computation has to be controlled (t2).

2.1.1.2 The Enforceable Policy

A security policy is used to specify, either directly or indirectly, which interactions are authorized in the process. The first chapter introduced the policy classes that are typically used to specify security requirements in business processes. These four policy classes are authorization, separation of duties, usage control and isolation and can be incorporated into the specification in different ways. As mentioned at the beginning of this chapter, only enforceable safety properties can cause a reference monitor to block the execution of a workflow, thus cause an obstruction. However, not every security policy is a property in the sense that was described at the beginning of this chapter. In fact, some security policies cannot be defined using the criteria that individual executions must each satisfy in isolation. If the set of executions for a security policy is not a controllable safety property, then an enforcement mechanism for execution monitoring does not exist. Therefore, the different policy types need to be examined for their enforceability in the sense of the introduced trace properties.

In general, “every property which is formalizable as a set of traces can be written as the intersection (or conjunction) of a safety and a liveness property” [148]. This means, a sequence or trace can fulfill liveness and safety properties and encode different policies. At the same time, execution monitoring mechanisms for enforceable safety properties compose in a natural way as well. In the case where several such mechanisms are used simultaneously, the policy enforced by the aggregate is the combination of the policies enforced by each mechanism in isolation. This is of interest because it enables complex policies to be decomposed into conjunctions with a separate mechanism used to enforce each of the component policies [182]. Hence, because the policy composition does not affect enforceability, it is possible to subsequently examine each of the different policy classes separately to identify the ones that are relevant for obstructability. The policy classes therefore will be elaborated in greater detail to identify to which extent they are enforceable. Moreover, examples of typical specifications are given, which often represent models

well-established in practice. In combination with the policy classes, this section will examine the related process aspects in greater detail in order to comprehensibly identify relevant policies. To the best of our knowledge, this is the first systematic attempt to comprehensively analyze the enforceability of all given classes of business process security requirements in terms of trace properties.

Authorization: Access control regulates the authorization of individual subjects, namely users or IT-systems, within processes or also the granting of access to corresponding resources or data. It centers on the organizational process aspect. As introduced in chapter 1, authorization means enforcing access control to ensure that only authorized individuals are allowed to execute activities within a process. Because this allows to define authorized as well as unauthorized trace properties, authorization represents an enforceable safety property [182]. Authorization forms the core of access control along with authentication. The latter is usually part of the supporting IT-Infrastructure. The information system that guides the process execution usually gets the information about authorized users from the implemented access control model. For this, Access Control Lists (ACL) or also Role-based Access Control (RBAC) are used. Corresponding roles are created for different tasks or functions, which can also be structured in hierarchies and allow for the delegation of rights. Hence, PAISs can use authorization concepts to enforce user activity assignments (e.g., an SAP-System with Authorization or similar), i.e., the mapping of process tasks to stakeholders with respective permissions and enough capacity. The informational aspect complements authorizations that purely assign process participants to activities with requirements related to properties of data elements, for instance how they can be used or produced. In particular, through the consideration of data elements, such as databases, documents or variables, certain subjects can be involved in the process or excluded, for example, on the basis of a loan amount. Such data-oriented conditions are however usually encoded or annotated in the control flow specification as refinements of branching conditions. The focus, however, is on policies that are not specified in the control flow but in a separate policy, such that situations arise in which the policy enforcement monitor can block the control flow execution.

Separation of Duties: A further concept that embodies the organizational aspect, is the separation, or, its dual, the binding of duties to avoid conflicts of interest or reduce the risk of fraud (cf. Chapter 1). It generally states that some activities in the processes cannot (SoD) or must be (BoD) executed by the same subject or by the same role. The specification of the authorization already allows for a structural separation or binding of duties in using appropriate access control concepts.

For example, separations are taken into account when designing the organizational structure of a company. Suitable role and authorization concepts are defined to ensure that processes meet the requirements of functional separation, for instance, that different departments are involved in a process. However, role hierarchies can be very complex with the result that individuals can avoid the intended separation by acting in different roles. Thus, the separation or binding of duties needs to be implemented on an individual level as well, for example with the Four-Eye Principle, which requires that always two people are involved in a process. Such policies represent constraints that act on top of authorization. Here, based on the execution of specific activities of a user, or in other words, the execution history of a process case, the user is forced (binding) or not allowed (separation) to execute other process activities. This is how the assignment of actually authorized users is further constrained. Because such an assignment can also be encoded as a trace property, allowing or denying a respective access can be enforced during execution. Hence, similarly to authorization, the separation of duties and the binding of duties encode enforceable safety properties. Regarding the specification of authorization and SoD policies, the NIST RBAC standards provide three levels of RBAC. It allows to define basic RBAC, hierarchical RBAC which also supports inheritance between role, and RBAC-Level-2, or also called “constrained” RBAC, which supports the definition of separation of duties as well. Further, using formal descriptions of security properties, model checking techniques may also be applied, such that BoD and SoD can be verified by specifying appropriate LTL formulas as patterns as well. These specifications can be used with the help of reference monitors that are able to enforce both authorization and SoD-related policies.

Usage Control: Usage Control is primarily associated with the functional and behavioral process aspect and expresses conditions that must hold after the access to a resource [179]. It is often used to capture regulatory compliance and especially privacy and data protection requirements. More specifically, it requires the specification of pre- and post-conditions for the execution of process activities or the access to resources or data. Conditions not only describe the maximal number of access but also temporal relationships of activities independent of specific execution paths. They may also constrain the data retention, for example, requiring to delete local copies of a data item after its access and usage. However, depending on their observability, usage control requirements are not enforceable by a reference monitor and they encode liveness properties. However, it is possible to reformulate certain usage control policies in a way that they become controllable. Such enforceable usage control policies consider the interplay of different activities and are encoded in the control flow, typically in the course of the specification of con-

ditions and obligations. These control flow constructs can be captured as patterns as well. Usage Control, analogously to authorization, can capture the informational process aspect as well and it can consider data elements used in the process and their influence on the control flow of the process. For example, the annotation of branching conditions with data-related constraints determines the choice of subsequent paths. The consideration of given usage control policies in the control flow is also achieved by means of process rewriting, which is challenged to keep functional and behavioral aspects of the initial control-flow. By this, it is possible to enrich a process model with activities derived from usage control policies, for example policies in which the condition determines the obligation, such as “whoever uses a service has to pay for it” or “when the data is accessed, the user must be informed”. Thus, usage control policies can use the same model specification language as the process model. Hence, a big part of usage control can be taken into account in the process model. Such model-based policies, however, do not lie in the regarded area of conflict between the workflow and the execution monitor. However, there is a minor portion of usage control policies that are as formalizable as enforceable safety properties, for instance, the maximum number of access to resources.

Isolation: Isolation policies stem from the informational process aspect. Isolation generally says that confidentiality and also integrity of information must remain during the execution of a process. Isolation policies can be subsumed to the class of information flow policies, which define the way information moves throughout a system. Any confidentiality and integrity policy embodies an information flow policy. They either preserve confidentiality of data, to prevent information from flowing to an unauthorized user to receive it, or the integrity of data, such that information may flow only to processes that are not more trustworthy than the data [33]. In this respect, authorization and SoD represent information flow policies as well and access control can be seen as a component of information flow, in which the accessed object is the information. The informational aspect constitutes a generalization of the organizational aspect. However, its focus is on information and its distribution, not on the organization of the company in the sense of connecting resources and users to processes to eventually run a business. Isolation policies, on the one hand, address potential conflicts of interest. On the other hand, isolation policies restrict the direct and indirect flow of information in a process and the PAIS:

- As a rather organizational policy, isolation can be used to avoid conflicts of interests. Here, the aim is to prevent the flow of sensitive information between competing companies or departments involved in the execution of a process. The history of activities that have already been executed is key for the decision

as to whether a person is authorized to carry out certain activities. For this, the Chinese Wall security model is able to describe business policies to separate conflicting domains in a history-dependent dynamic way. It is a model of a security policy that equally refers to confidentiality and integrity. In particular, by defining so-called conflict classes, it can be avoided that business areas that are in conflict with each other interfere and conflicts of interest arise (e.g., if competing enterprises work with the same external consultant).

- Isolation is further necessary for business processes of different clients such that information must not flow between the different client domains (for instance in cloud technologies), but also within business processes when sensitive information is only allowed to flow to certain subjects involved in the process execution and not to unauthorized people. Here, information flow policies can specify security levels modeling clearance levels and allowed or unwanted flows between parts of the process, for example users or activities of different security levels. Further, they describe that subjects involved in a process are fully isolated from each other or that they can only communicate over a so called declassification channel. For example, the Bell-La-Padula security model can be used to preserve the confidentiality of data elements. It associates users and workflow tasks with a security label [28]. Thereby, it can be stated that, for instance, two tasks can only be performed by users within the same security clearance. In analogy to usage control, there are patterns that are able to directly encode the isolation policies into the control flow model (e.g., the so-called “no-read-up”, “no-write-down”) and that can be used to check if the policies are fulfilled in a given process model.
- Besides explicit information flows due to direct access operations, **covert channels** (or implicit flows) may allow an indirect information transfer and inferences about secret information by analyzing the process behavior (structure-based implicit flows and timing channels) or its data handling (storage channels), such that executions of the process do not interfere with one another. The consideration of covert channels mainly originates from contexts with high security standards, such as military. Such information flow policies restrict what information subjects can infer about objects from observing system behavior [182]. Associated security requirements are characterized by the term non-interference. A program (or process) is typically said to be non-interfering if the values of its public outputs do not depend on the values of its secret inputs [97]. In fact, what is often meant by the informational aspect is to consider the implicit flows, most prominently non-interference properties. Non-interference is a very restrictive security notation that can reflect not only the confidentiality of data elements but also the confidentiality (in a sense of non-observability) of process activities.

Regarding the enforceability of isolation policies, for instance, the Chinese Wall security model can be enforced, because it may limit the number of tasks that any single user can perform. This also applies for Bell-La-Padula, for example when users involved in two tasks have the same security clearance. Both examples represent enforceable safety properties whose violation can be checked on the trace level. However, it is important to note that not all isolation related properties can be characterized as trace properties such that they are enforceable by a monitor. Such monitors which see executions as a sequence of performed actions are not sufficient to enforce strong information flow properties. Whereas trace properties hold for sets of traces, these isolation requirements are only specifiable as properties over sets of sets of traces, so-called hyperproperties [46]. They can be used to define strong information flow policies which specify to what extent information can be learned by users of a system, or respectively, the actors that participate in a process. Hence, strong indirect information flow policies do not define sets of properties in the sense of a trace property, so they do not define safety properties. This particularity is highlighted in the works of McLean and Schneider. McLean [148] proved that non-interference information flow policies are not trace properties. Because they are no safety properties, there are no enforcement mechanisms to enforce them during execution [182]. Hence, they cannot cause an obstruction in the sense of this work.

This difference can be illustrated with a short excursion into communication theory. If one understands the obstructive situation communication-theoretically, the trace focuses on what is said, that is, which activities are actually carried out. Based only on what is said, on the given trace prefix, an enforcement monitor can decide whether it is permissible or not. In contrast, strong information flow policies do not only depend on what is said (in terms of a trace prefix) but also on what is not said. In a sense, this can be related to Watzlawiks first axiom “One cannot not communicate” [211]: What is actually said does not only relate to itself but can be seen in relation to what else could have been said, based on a given set of possible pieces of information or messages. If this is transferred to a process, it must not only be considered how a process is executed, but also what information can be inferred from what could have been executed on the basis of the model (which is what was not executed). Nevertheless, although an obstruction results from a focus on “what is said”, the question “what else could have been said” may be relevant for the resolution of an obstructive situation.

The focus of this thesis is on controllable safety trace properties, whose enforcement causes an execution monitor to obstruct the process execution. Hence, in conclusion, only authorization and SoD policies in the sense of execution monitoring are

fully enforceable as safety properties. Only few usage control and isolation policies are enforceable in such a way that they can be of interest in causing obstructions. Hence, with regard to the different process aspects, an obstruction primarily shows the conflict between the functional and behavioral aspect and the organizational aspect. Besides, the informational aspect is not enforceable in its essential characteristics as far as trace properties are concerned. Strong informational flow policies are therefore not considered any further. However, some of the enforceable isolation policies also relate to users, so that, similarly to the organizational aspect, the assignment of users to activities is controlled, for example, Chinese Wall and Bell-La-Padula. Analogously to the isolation policies, only a part of the usage control policies is enforceable, for example the maximal number of resources a user can access (cardinality). Equally, for usage control, the enforceable part can be related to the organizational aspect. As a further observation, policies can also be mapped into the control flow, for instance usage control policies. Thereby, they eventually also form execution sequences or traces, which can be grasped as properties. However, because they are not encodeable as an enforceable policy, they cannot cause an obstruction and are therefore neglected.

Based on these findings, the considered policy shall typically involve the authorization policy that defines which users are authorized to perform which workflow tasks. On top of that, further polices can be involved that stem from SoD, but also Usage Control or Isolation requirements, and further restrict which of the authorized subjects can perform specific tasks. These are often called “authorization constraints”, which means they are constraints on top of the basic authorization but are crucial in finally determining the user-task assignment during execution. Hence, it is possible that a user, while authorized by the authorization policy to perform a particular task, is prevented (by one or more constraints) from executing this task in a specific workflow instance because particular users have performed other tasks in the workflow before. As an example of such policy, an execution of the workflow of determining the market value shall now be constrained by the authorizations illustrated as assignments from subjects to tasks in Figure 2.6a. For example, Alice can control the computation (t2) whereas Bob is not authorized to do so. Moreover, the workflow system that executes the payment workflow has the SoD constraint that t1 must be executed by a different subject than the one performing t2. SecureBPMN [36], SecBPMN [177] or further BPMN extensions [39]³ are suggestions for modeling security policies, which have, however, not been consid-

³ Basin et al. provide a concept [23] and tool support [39] by extending the modeling tool Oryx with a BPMN extension for authorizations to enforce abstract SoD constraints.

ered in the standard so far. For this work, a variation of these approaches is used to model authorization constraints. The affected tasks are connected with dashed lines, whose label represents the type of the individual constraints, e.g., “ \neq ” for SoD in Figure 2.6b and “ $=$ ” for BoD.

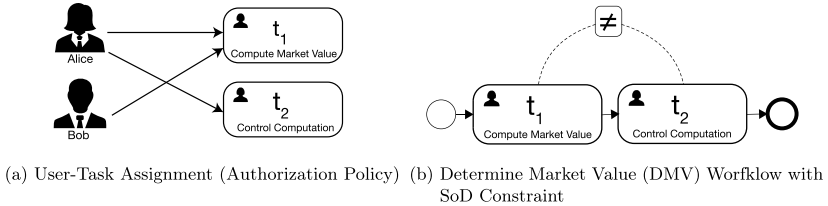


Figure 2.6 DMV Model and policy based on the example in Burri [39]

In conclusion, all of the identified enforceable policies can have a negative impact on the process execution, because, depending on the contextual situation and the existing execution, they ultimately may prevent available users from being assigned to the execution of a task, thus causing a workflow to become obstructed. Such a scenario may not only depend on the constraints, but may also happen when authorized users are unavailable. Existing research relates this to the general notion of workflow satisfiability and workflow resilience.

2.1.2 Satisfiability

The notion of obstruction is related to the so-called satisfiability problem of workflows. The basic version of the Workflow Satisfiability Problem (WSP) assumes the existence of a process model specification, an authorization policy, and a number of authorization constraints. Given a set of users U , a set T of tasks, and a policy, which consists of an authorization list for each user $u \in U$, which determines the tasks for which u is authorized, and a set of constraints on T . Then, the WSP asks, if it is possible to find a valid plan $\pi : T \rightarrow U$, which assigns a user to every task, such that the policy, namely both authorizations and constraints, is satisfied. If a valid plan exists for an instance of the WSP, then the instance is called satisfiable. The authorization list itself can be seen as a set of specific, rather simple constraints that encode the authorization policy. However, it makes sense to assume that for every task, there is some user who is authorized to perform it. Otherwise, it is trivial that

the workflow is unsatisfiable. Because this builds the basis for further constraints, the authorization list, or the authorization policy, respectively, is handled separately.

Given the introduced policies of the market value computation, the workflow is in general satisfiable because Bob can execute the first task and Alice is authorized to perform the second, which is illustrated in Figure 2.7. In other words, there exists a valid plan to execute the workflow.

The practical motivation for the WSP is based on three main reasons, which are of particular interest to policy-designers as well: The first one is that it can be used in a form of static analysis before deployment to ensure that the workflow specification is useful in the sense that there is at least one possible “execution path” throughout the workflow. Secondly, the WSP can be used to synthesize plans for workflow instances, which assign the tasks to the users for each instance of the workflow so that they can be used when instantiating the specification. Thirdly, the WSP can also be used in a more dynamic way if tasks in a workflow instance are not assigned to users in advance, which is related to obstruction-free enforcement. This section will consider the satisfiability aspects at design time, such that satisfiability can be analyzed before the execution of a process. These aspects are also connected to workflow resilience, which will be considered afterwards on the basis of this section.

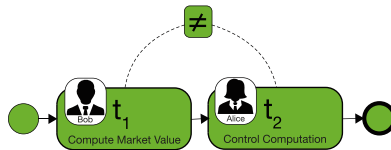


Figure 2.7 A satisfiable assignment of users to tasks

2.1.2.1 Initial Publications

Workflow satisfiability and the associated problems are an active area of research. Its beginnings date back to the turn of the millennium. The work of Bertino et al. [30] from 1999 considers the completion of security-aware workflow instances. Their exponential algorithm assigns users and roles to tasks in sequential workflows by keeping an authorized user from performing a task when this implies that subsequent tasks would not have authorized users who satisfied the workflow constraints. Thereby, the algorithm generates an actual valid user task assignment rather than deciding whether a workflow is satisfiable or not. Afterwards, “unfortunately, solutions to [the satisfiability] problem have largely been overlooked in

the literature” [194]. The problem was tackled from another direction by Wainer et al. [204] in 2003. Their idea is to assign priority levels to each constraint and different override levels to users. In case a workflow is not satisfiable, some constraints may usually be overridden by unauthorized users who, however, have an override level that is higher or equal to the priority level of the affected constraints. Inconsistencies within the specification of constraints were analyzed by Tan et al. [194] in 2004. Such inconsistencies can cause an unsuccessful workflow execution. They define a model for constrained workflow systems, which includes constraints such as cardinality, SoD and BoD. The authors define a workflow as a partially ordered set of tasks and explicitly specify workflow and task instances. Authorization constraints are given for pairs of tasks in terms of relations over users that must be satisfied when executed. Eventually, in 2005, Crampton was the first to establish the term “workflow satisfiability”. He refined the existing ideas by looking at workflows as partial orders, defining simple SoD and BoD constraints, and developing an algorithm to determine if there is a mapping of users to tasks that meet the constraints. This represents the basic problem formulation of the WSP. In 2006, Solworth used the notion of “approvability” and allowed SoD constraints in the presence of loops if the first allocating task is always executed by the same person. For this, an approvability graph is designed to describe sequences of actions defining the termination of workflows with an RBAC policy, sequential or conditional executions. The approaches have so far all been based on the so-called ordered version of the WSP, which means they always refer to the task order check, task by task, whether there is still a valid assignment. In 2007 Wang and Li [208] were the first to not consider the order and introduced the unordered version of the WSP. In their work, they introduce an extension to the common role-based access control (RBAC [178]) to be able to define authorization constraints and capture common workflow system security requirements (e.g., SoD). They use plans to study the time complexity of the WSP and prove that the WSP is NP-complete in their access control setting and reduced the problem to Boolean satisfiability problem (SAT), which allows the use of off-the-shelf SAT-solvers. Moreover, they prove that a workflow system supporting subject-task authorization and either so-called subject-inequality (SoD) or existence-equality (BoD) constraints (or both) to be NP-hard, however, efficiently solvable if the parameter is the number of tasks of a workflow, which is usually smaller than the number of involved users. Wang and Li’s FPT proof motivated many later works by Crampton et al., all then considering the unordered version of the WSP for workflows specified as partial orders. Wang and Li further were the first to use the term of resilience in this context. With these works, foremost the works of Bertino et. al., Crampton et al., and Wang and Li, the basis has been laid for the subsequently strongly growing research interest in the field.

Since 2007, research related to the WSP has branched out in different directions. In the observation of the initial publications, essential criteria already emerge, which have then been deepened in the course of further research. Besides, it can already be observed that it is a common practice in the analysis of workflow satisfiability and also resilience to abstract from parts of a workflow specification. For example, it is common practice to limit the allowed control flow constructs or supported authorization constraints and to neglect data-oriented elements. For instance, there is only one work from 2011 that actually at least considers the data flow. However, it does not model data-flow but overapproximates data-oriented gateway decisions with an internal choice operator [24]. Further, despite its strong relations and origins to access control, it is hardly spoken of a “subject” (which accesses an object). The subjects, which can represent human users but also IT-agents or systems, are often simplified to “users”, which may better reflect the business process context and the direct connection to the problem of potentially missing process participants. In this thesis, both terms are used interchangeably. Today, more than 100 publications can be counted and, to date, about 10 to 15 new papers on the subject have been published every year in renowned conferences and journals. Covering every publication in detail therefore would be too exhaustive and would diffuse the focus. Therefore, the spotlight will be on the key findings and deficits that are of relevance when considering obstructability. For this, first important aspects of satisfiability, afterwards resilience and then its runtime versions (in the section on execution monitoring) will be considered.

2.1.2.2 Process Structure

The structure of the process models upon which the different instances of the WSP in the literature are based differs. As the initial publications already indicated, differentiation can mainly be made between workflows that only allow a sequential or linear execution of tasks, and partial orders that also enable concurrent executions. There are only few other approaches that allow for choice branches and looping tasks (e.g., using Hoare’s process algebra CSP [24]). In the latter case, the tasks executed in a workflow vary from one instance to another, which further implies that there may be constraints that only apply for certain sequences of tasks. In this regard, the example process in Figure 2.5 requires full support of all of the mentioned structural possibilities.

2.1.2.3 Order of Assignment

Solutions to the WSP also differ in how the order of the tasks is considered when the users are assigned. As one possible solution, the unordered WSP offers a plan that assigns users to tasks in such a way that all tasks have an assigned user and

all constraints are satisfied. In contrast, the ordered version offers a plan with an execution sequence, such that the assignment must respect the ordering of tasks defined by the control flow. The ordered and unordered versions of the WSP are only equivalent for workflows with tasks that can be executed in any order [61]. For the other cases, assuming either the ordered or the unordered version can make a big difference. This can be imagined by a slight modification of the DMV authorization policy in Figure 2.6a, such that Alice and Bob would now both only be authorized for the second activity, and only Alice would be authorized for the first one. In the unordered case, assigning Alice to the second task before assigning her to the first would not allow the assignment of the first task anymore due to the SoD constraint. The ordered version would, however, first consider the assignment of Alice to the first task such that for the second task, Bob would still be assignable. In literature, the consideration or non-consideration of the order is roughly in balance while there is the tendency that the more complex the policy under consideration, the less the order is taken into account.

2.1.2.4 Constraint Types

The subsequent analysis of the constraints considered in WSP research will show that the WSP considers constraints that relate to the types of enforceable and controllable policies from process security that were determined in Section 2.1.1.2. The terminology for these policies in the WSP context differs, however, from the terminology of process security policies. The latter rather stems from business or regulatory rules. For instance, the SoD constraint against fraud can be related to the requirements of an Internal Control System (cf. Chapter 1). In contrast, the terminology of the constraint types in WSP research will manifest further references, such as the connection to constraint satisfaction problems (CSP) and its complexity-theoretical considerations, as well as the entwinements to access control research, combined with the ambition to generalize findings on the instances of the WSP and the constraints considered therein.

Therefore, in the course of the different publications on the WSP, a growing number, differentiation and abstraction of constraints types—which are in fact described very differently in the individual publications albeit they often mean the very same thing—have developed. Although some initial works followed other attempts to specify constraints, for example, Bertino et al.’s constraint specification language [30] and Li and Wang’s Separation of Duties Algebra [208], the following

classes of authorization constraints for workflows have meanwhile crystallized [63, 181]⁴.

- *Counting (or also named cardinality) constraints* specify that a user is allowed to perform either zero tasks or a specific number of tasks within a certain range (e.g., to count the maximal number of accesses). One example of counting constraint is $(1, 2, \{t_1, t_2, t_3\})$, which means that a user can execute 0, 1 or 2 tasks among those tasks in $\{t_1, t_2, t_3\}$.
- *Entailment constraints* describe properties for the assignment of users to two disjoint sets of activities that entail each other, typically SoD or BoD constraints. They differ in the allowed cardinalities of the sets, such that either both sets are singletons, at least one set must be a singleton, or there are no restrictions on the cardinality of the sets. Respective examples are, according to the enumeration, $(\{t_1\}, \{t_2\}, \neq), (\{t_1, t_2\}, \{t_3\}, \neq)$ and $(\{t_1, t_2\}, \{t_3, t_4\}, \neq)$. The first constraint is satisfied if a user u_1 executes t_1 and u_2 executes t_2 (because $u_1 \neq u_2$). The second and third constraint are satisfied if u_1 executes t_1 and u_2 executes t_3 . Those are examples of SoD. BoD constraints can be defined similarly by using $=$ instead of \neq . A special class of singleton entailment constraint, which is of interest for different security models, considers *equivalence-based constraints*. For example a constraint (t_1, t_2, \sim) , where \sim is an equivalence relation on the set of users, means that the user who executes t_1 and the user who executes t_2 belong to the same equivalence class, for example the same role (cf. RBAC) or the same security clearance (Bell-La-Padula) (or (t_1, t_2, \approx) for different classes).
- There are two generalizations of these classes, namely user-independent constraints and class-independent constraints. *User-independent constraints* are those whose satisfaction does not depend on the individual identities of users. *Class-independent constraints* are those whose satisfaction depends only on the equivalence classes that users belong to. Given a class-independent constraint, it does not matter to which specific classes the assigned users belong, only that the classes are different (\approx) or the same (\sim). Every equivalence constraint (t_1, t_2, \sim) or (t_1, t_2, \approx) is class-independent. Every user-independent constraint is class-independent, meaning that, if a plan satisfies a user-independent constraint, any user of that plan can be replaced by an arbitrary user, such that the replacement users are all distinct. Many constraints of practical interest, such

⁴ Although, subsequently, these different kinds of constraints are given by the help of some formal examples, they only intend to give a better understanding of the intuition behind the provided concepts. More profound formalizations will be given in the subsequent chapters.

as separation-of-duty, binding-of-duty, and cardinality or counting constraints, are user-independent [60].

2.1.2.5 Complexity and Fixed-Parameter Tractability

At the core of parameterized complexity, there is the idea of finding an aspect of the problem that makes it intractable, namely NP-hard. Then, depending on the application under consideration, a parameter k is introduced, which measures this aspect in such a way that k would be relatively small for those problem instances that arise in practice. The aim is to design efficient algorithms called fixed-parameter tractable (FPT), which run in time $O(f(k) * n^c)$, where f is an arbitrary computable function in k only, n the size of the problem, and c an absolute constant [71]. Being polynomial for any fixed value of k , such algorithms are extensions of polynomial algorithms for NP-hard problems. A given formula of a satisfiability problem is parameterized by the number of variables, such that the problem size n with k variables can be checked by brute force in time $O(2^k n)$. In this respect, to make the WSP fixed-parameter-tractable, Wang and Li [206] introduce the number of tasks $|T|$ as the parameter k , arguing that in practice it is often much smaller than the number of users $|U|$.

At the beginning of the WSP research, it was well-known that the WSP is NP-complete. Wang and Li [206] prove that the WSP is also W[1]-hard, which first of all means that it is highly unlikely that there is an FPT algorithm for the problem. However, in their work from 2007, they also show that WSP is FPT if the constraint set is limited to certain simple constraints. In fact, this assumption appears very natural in practice. Crampton et al. [65] extend the classes of workflow specifications from Wang and Li [207] for which the satisfiability problem is known to be FPT. Those classes include counting constraints, entailment constraints and constraints based on equivalence classes. In this way, organizational hierarchies or constraints, which for instance implement the Bell-LaPadula security model or other business rules, can be defined. The authors establish the circumstances under which an instance of the WSP has a polynomial kernel, they are able to solve the original problem more quickly by applying kernelization and show that it remains FPT for counting and equivalence constraints [64]. Crampton et al. [61] pick up the concept of constraint expressions [128] (logical combinations of constraints) to translate an instance of WSP for entailment constraints with unrestricted cardinality into multiple instances for singleton entailment constraints. The underlying idea is that an instance of the WSP for a conditional workflow can be solved as many instances of parallel workflows. By this, it is uniformly possible to support conditional workflows and entailment constraints with no limit in cardinality and thereby keeping fixed-parameter-tractability. They further develop a first solution [59] to the WSP

regarding seniority constraints and specify special cases that are fixed-parameter-tractable in which they are still able to represent common subject hierarchies from practice. Cohen et al. [49] solve the WSP using techniques for the Constraint Satisfaction Problem, which allow the authors to present a generic algorithm to solve the WSP using the general constraint types: cardinality, so-called regular (e.g., SoD) and user-independent constraints. Their solution builds executions incrementally, discarding partial executions that can never satisfy the constraints. The authors show that their algorithm is optimal for user-independent constraints. As an important landmark, Cohen et al. further introduce the aforementioned notion of user-independent constraints [49], which constitute a natural generalization of the simple constraints Wang and Li considered. Crampton et al. [60] were the ones that did extend the notion of user-independent constraints to that of class-independent constraints. They show that such WSP remains FPT and propose an algorithm to solve it. It is shown that user-independent constraints provide a good trade-off between expressive power and tractability. On the one hand, user-independent constraints include many workflow constraints of practical interest. On the other hand, the WSP remains FPT if user-independent constraints are considered. This focus on user-independent constraints enabled the development of highly efficient algorithms and tool-support [47, 63, 125]. For example, Crampton et al. [60] and Cohen et al. [47] show the superiority of their FPT algorithm in comparison with the classical SAT reduction of the problem.

2.1.3 Resilience

Resilience asks to what extent a workflow is satisfiable against the absence of users, for example due to vacation or illness. Thus, it is a question in the context of the WSP, which represents a way of conditioning the WSP on the basis of constraints that are independent of the workflow. As a preliminary study, Li et al. [135] introduce the concept of resilience policies for access control systems. These policies specify a minimum number of users who must have certain privileges. Thereby, an appropriate level of redundancy is ensured in the system so that the system is resilient to the absence of users. The resilience checking problem (RCP) related to this, draws on the ability to check whether an access control state satisfies a particular resilience policy. Wang and Li [206] then examine resilience in the workflow system context and its relationship to the WSP. Based on the observation that there are different types of workflows whose execution time frame varies between hours, days and weeks, Wang and Li propose a three-layered view on resilience in workflow systems [207]: (1) static resilience —some subjects are absent before the execution while remaining

subjects will not become absent during the execution; (2) decremental resilience—subjects are absent before or during the execution and absent subjects will not become available again; (3) dynamic resilience—subjects may be absent before or during the execution and absent subjects may become available again. A workflow is said to be (statically) k -resilient and remains satisfiable even after any k users are removed from the current state.

Regarding the market value example, suppose that Alice becomes ill and is not available to participate in the execution of the workflow: Although, based on the authorization policy, Bob is authorized to execute all tasks of the workflow, he will not be allowed to perform t_2 after executing t_1 due to the SoD constraint. Similar problems would arise if Bob was not available. Hence, the absence of either Alice or Bob would result in an unsatisfiable workflow, which is why the example workflow specification represented in Figure 2.6 is not resilient for $k > 0$ absent users.

Solving resilience questions can clearly be motivated by similar aspects for policy designers as done for the WSP. Moreover, it is of particular interest for emergency planning, which is important for many companies today. As identified in Chapter 1, a policy should follow the principle of least privilege such that access is only granted if it is absolutely necessary. Resilience requirements have a completely contrary conception. For example, the mentioned resilience policies show the attempt to capture these opposing goals by means of a policy as well. Looking for ways to reason on and resolve the contradictions between conventional access control policies and resilience requirements is an active research area, and a challenging task for policy designers.

In essence, resilience is associated with ensuring the achievement of business goals even if some users are not available for the tasks that contribute to the achievement of those goals. Although this has similarities to the general aim of this thesis to ensure the achievement of business goals, an obstruction does not necessarily happen due to an unplanned absence of users, but is rather caused by the policy. Nevertheless, based on these similarities, resilience approaches offer interesting concepts, so that the following aspects will also be of interest for considerations to detect and handle obstructions.

2.1.3.1 Synthesizing Execution Plans

The generation of execution plans is an important component in determining resilience because the existence of multiple valid plans makes it possible for a workflow to be completed even if a number of users are absent. Literature about the synthesis and verification of such plans for security aware workflows addresses different levels of resilience. Crampton et al. [57, 66] use bounded model checking to generate authorized plans and derives measures to evaluate the resilience of

solutions by obtaining the sizes of minimal subjects bases [68]. Thus, before the execution of a workflow, it can be analyzed what the minimal subject base is to complete a given workflow during execution (static resilience). As Lowalekar et al. [138] point out, several assignments with the same degree of k -resilience may exist such that it may be necessary to pick the most favorable one. As an approach related to decremental resilience, Paci et al. [158] generate a set of valid plans that are stored to decide which user to assign to a task in case the previously assigned user is unavailable at runtime. For this, they introduce resilience constraints that state the minimum number of users for a satisfiable execution. They describe a process to be user-failure-resilient if a user-task assignment that meets the resilience as well as the security constraints can be found. Massaci et al. [145] propose an approach to analyze a priori execution if a given subject assignment is resilient against the dynamical absence of subjects.

2.1.3.2 Quantification

Instead of investigating the maximum number of absent users (k -resilience) or the minimum number of available users (minimal subject base), the so-called quantitative resilience, proposed by Mace et al. [141], examines the probability of the availability of the users. Based on this, the path that offers the highest degree of resilience with regards to the maximum probability of a valid process completion is determined. Therefore, quantitative resilience seeks to measure the extent to which a workflow is resilient, rather than simply deciding whether it is resilient or not. Thus, with the introduction of quantitative resilience, the risk of user absence, or its cost respectively, is quantified for the first time.

2.1.3.3 Feasibility and Change of Policies

As for the risk, one is willing to take, to achieve resilience, literature is going further. Thus, it can be considered which policy changes are needed in order to make a non-satisfiable process specification satisfiable. These changes can be connected to risks or quantified as costs. The idea to edit the policy was initially associated with the concept of feasibility [128]. The authors describe workflow feasibility as the dual of workflow resilience. Hence, their intention is to not “boil down” a given workflow specification to a critical state to assess its resilience, thereby determining the minimal subject base or maximal number of absent users. In contrast, they consider a workflow that is not satisfiable, such that the question is, if it is feasible at all to somehow complete the workflow. They use relationship-based access control, which allows them to repair the policy by edge addition or removal in a social network. With regards to the constraints needed to address the workflow satisfiability problem, relationship-based access control models, however, only offer a limited way of

specifying authorization policies. Nevertheless, feasibility represents the basic idea in repairing and changing the policy to allow for satisfiability, which is then explored in further works. For example, the idea to change the policy is used by Basin et al. [25] which use a cost-based approach to increase the resilience of a workflow. Given an unsatisfiable workflow, a new user-role assignment is determined, whereby the costs of change regarding the risk of a role substitution, maintenance and administration are minimized. Further, Mace et al. [142] allow policy designers to automatically evaluate workflow resilience and compute optimal security constraint changes to ensure a certain resilience threshold.

2.1.3.4 Policy Violation

A further, in a sense more drastic, step towards reaching resilience of an unsatisfiable workflow is the violation of the policy. Here, the risks associated with a policy violation are considered. With the so-called Valued WSP [62] and its refinement, the Bi-Objective WSP [63], Crampton et al. define approaches to optimize the user-task assignment for a workflow that is unsatisfiable, for instance due to user unavailability, such that the “least risky” assignment is found in order to achieve resilience. They distinguish between the authorization constraints and the authorization policy, which can be violated in order to complete a workflow. The risk associated with the violation is expressed as a cost. The algorithms that find a user-task assignment which completes the workflow with minimum cost, is shown to be fixed-parameter tractable with user-independent constraints. The bi-objective WSP aims to minimize two weight functions associated to a valid plan, one representing the violation of constraints and one representing the violation of the authorization policy, which results in a set of incomparable solutions (a Pareto front), allowing the user to choose the most suitable. This may help policy designers to find a plan that, for example, ensures that the cost of constraint violations is zero and that the cost of policy violations is minimized. If the overall cost is zero, the workflow is definitely satisfiable. In relation to Mace et al. [128], Crampton et al. [63] further show how their approach can be used to consider user availability as well. On the one hand, there are the costs of assigning an unauthorized or unavailable user. On the other hand, there are the costs of assigning an authorized user, which correspond to the probability that the user is not available. The work by Crampton et al. also touches on works related to risk-aware access control [43, 44, 80, 156], which aims to quantify the risk of letting a user execute an action instead of just making a decision to approve or deny such action. It also ensures that the accumulated risk remains within certain thresholds. However, unlike other work, their focus is on calculating user-task assignments at minimal cost instead of access control decisions [62].

It can be concluded that, if a workflow is found to be unsatisfiable, it can never be completed without changing or violating the security policy. The areas of costs, policy changes and violations, and user availability can also be considered together. For example, the approach of Crampton et al. on the one hand allows a consideration of whether the risk of a violation is greater than the risk that the actually authorized user is not available [63]. All these metrics and costs ultimately represent indicators, which, as shown above, can be used to minimize violations, to not exceed thresholds, or to indicate options for action, among which the user has to choose.

2.1.4 Obstructability: The New Factor

As a general observation, checking for resilience of a satisfiable workflow is comparable to putting a healthy patient through his paces at the doctor's in order to compare his vital values with the critical threshold values that indicate illness. In contrast, the feasibility, or more generally the change or violation of the policy, assumes a sick patient, who should become healthy again with targeted measures, or in other words, it checks whether the restoration of health is feasible at all. These two poles of approaching resilience and related problems are reflected in the identified different aspects of existing research.

The same metaphor can be used to illustrate the difference between obstructability and obstruction. Checking obstructability assumes an actually "healthy" process specification, comparable to checking satisfiability or resilience. In contrast, the assumption that an obstruction is given, can be compared to the case of having a sick patient, so to say, a "sick process instance", which must be treated in such a way that it can recover, i.e., that the process can be completed. Because the latter happens during the execution time of the process, it will be covered separately in the next section. For the preventive analysis, in analogy to satisfiability and resilience, the notion of obstructability of a security-aware workflow is introduced. Given a security-aware workflow, obstructability asks: Is there a partial assignment of users to tasks of the workflow (or a partial plan) that obstructs the execution of the complete workflow, such that no other authorized user can be assigned to the remaining tasks? In terms of its effects, obstructability, as opposed to satisfiability or resilience, describes a danger rather than a desirable ability. Nevertheless, it appears reasonable to examine security-aware workflows from this point of view as it allows to reveal whether and to what extent obstructions are present therein. The mere analysis of the satisfiability or resilience tends to neglect or overlook the cases of possibly still existing obstructions, which will be examined in the following.

Regarding satisfiability, if a workflow is unsatisfiable (and not trivially unsatisfiable), this definitely means that there are only permutations of user-task assignments possible that are not able to fully execute the workflow, which in an ordered assignment represent obstructions. However, a satisfiable workflow may still be obstructable. Given the DMV example in Figure 2.6, for which satisfiability and 0-resilience was attested, and that all users are available, if Alice executes the first task, the execution of the second task will be obstructed by the policy. This obstruction is depicted in Figure 2.8. On the one hand, this is because Bob is not authorized to execute the second task. On the other hand, Alice would in fact be authorized to do so based on the authorization policy. However, due to her execution of the first task, it would conflict with the SoD constraint. This means, the satisfiable DMV workflow contains an obstruction. It is this simple example that clarifies that satisfiability does not mean obstructability, and that obstructability is not the same as non-satisfiability. Obstructability, however, looks at satisfiability from the opposite angle: Every unsatisfiable workflow is obstructable, but obstructable workflows are often satisfiable. In this way, an obstruction-free process specification is not the same as a satisfiable one either.

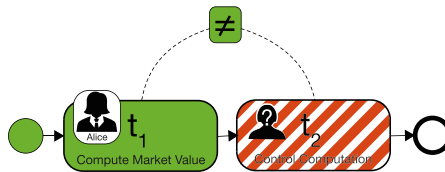


Figure 2.8 An obstructed execution after Alice is assigned to the first task

Regarding resilience, a resilient workflow is equally not necessarily obstruction-free. Suppose that, in the example, there was Claire and Dave as the third and the fourth user. Claire has the same access rights as Bob, Dave has the same rights as Alice. Now, if either of the four was absent, the workflow would still be satisfiable, such that it would be 1-resilient. However, if either Alice or Dave was absent, the aforementioned obstruction, as given in the example above, would still occur, if one of them executed the first task. Hence, resilient executions can still obstruct. However, obstructability can be compared to resilience to a certain extent because it is the organizational aspect that blocks further execution as well. The relationship between obstructability and resilience can therefore be stated this way: The foremost reason for obstruction is a policy problem because it is the policy that blocks further

process execution. This policy problem can however, be amplified from a lack in resource availability.

In conclusion, it can be observed that satisfiability and resilience only indicate the fact that there is the possibility of the workflow being able to be completed, potentially despite absent users. They concern the whole process specification, that is the workflow and its policy. In contrast, obstructability means that the process specification inherits the danger to obstruct but can in general be satisfiable or resilient. As introduced at the beginning of this chapter, an obstruction happens in a single case, which resulted in the chosen sequence, case and trace perspective to describe security properties. Hence, the difference in these concepts lies in the model vs. the case-based perspective, which clarifies why a satisfiable or resilient workflow may still contain obstructions.

Obstructability analysis before execution can clearly not handle an occurring obstruction. However, it is able to identify weak spots of the policy and build the basis for considering how to improve the policy or to handle such situations at runtime. Similarly to satisfiability research in its first years, it seems that obstructability has been overlooked as well, although capturing and handling obstructability is gradually becoming more relevant in research and industry, as Chapter 1 has shown. To the best of knowledge, this is the first work to explicitly introduce the “obstructability” of security-aware workflows as a notion.

2.1.4.1 Requirements for Obstructability Analysis (ROA)

For the analysis of obstructability, executions that represent a blockage between the policy and the workflow must be found. Analogously to satisfiability, the trivial case of such an obstruction would be that the policy simply does not provide authorizations for every task of the workflow. Only considering that this trivial case is not given, allows to speak of an obstruction in the sense of blocking actually authorized users, which, however, are not authorized in the obstructive situation. Hence, the interesting elements of the policy that actually cause an obstruction, are the constraints put on the basic authorization policy whose access control decisions depend on the execution history of the workflow instance. As shown before, obstructability contains elements of both satisfiability and resilience. Therefore, the findings and deficits of the different aspects of existing works on the preventive analysis of resilience and satisfiability are illustrated in order to formulate requirements that enable an adequate analysis of obstructability as well.

Ordered Assignment and Pre-Assignments (ROA-1): Constraint satisfaction in related work is often assumed to be independent of the ordering of tasks because the policy is defined in terms of sets and functions, and plans are also defined as

functions. In fact, the ordering of tasks is only relevant to the sequence in which the tasks of a workflow instance are executed, not in determining whether there is a valid plan, which is the basic application of WSP and resilience approaches. From the perspective of obstructability, however, it is exactly this sequence that encodes the execution history that is of interest. For example, the obstruction described in Figure 2.8, has no connection to reality if first the controlling of the computation is performed by a user before it is clear who actually does compute the market value. Hence, because obstructions consider the individual process case, it is natural that the assignment of users to tasks must take the ordering of tasks into account, which is defined by the control flow. Because an obstruction can only occur with respect to this task execution order, the ordering of tasks is actually implicitly assumed. However, based on this, a further distinction can be introduced, namely between the actual task execution, and the bare assignments of users, which will be termed as pre-assignments. In fact, situations may arise, in which a specific task at hand cannot be executed because succeeding “later” tasks have already been pre-assigned to users, tasks that should not have been executed yet. It may be the case that such pre-assignments must be preserved, and may not be canceled, for example, because it must be ensured that a task is only performed by a specially qualified user. Indeed, it is not untypical in the conduct of a PAIS to first reserve users for specific tasks before they are actually ready to be executed, for example to ensure that the process execution runs through without user bottlenecks. Based on this observation, a differentiation between the ordered version and the unordered version of an obstruction is introduced. While the ordered version of an obstruction assumes the tasks to be assigned and executed at the same time, the unordered version differentiates between these time points, such that it may be the case that tasks that are to be executed later, have already been assigned to users. This can be compared to the computation of a plan, which is used to prepare its actual execution. Such an “unordered” version of an obstruction means, an obstructive situation may be caused by such pre-assignments as well (or at least takes pre-assignments into account). It, however, still relates to the task execution order. More specifically, it encompasses an “unordered” (in the sense of not sequential) pre-assignment of users to tasks, and an ordered assignment with respect to the users that actually execute the tasks.

Comprehensive Structure (ROA-2): As indicated in the CEW example, the structure of a business process or workflow does not only need to be sequential or concurrent. Currently, more comprehensive models hardly exist in WSP literature at all. Few approaches consider control-flows that are more complex than partial orders. However, because they do not focus on obstructability, they rather implicitly also

analyze obstructions but do not explicitly find them, nor are they able to adequately capture and represent them. Because this dissertation aims for a practical solution, it also tries to capture the most realistic and comprehensive workflow representation for the definition of the problem of obstruction. This means sequential, concurrent, parallel, conflicting and looping activities shall be allowed. In this way, it is also possible to further consider our example, which represents a process template from practice as well.

Allow for Efficient Techniques (ROA-3): Literature discussing WSP and resilience in workflows mainly offers theoretical approaches that focus on understanding the complexity and finding efficient solutions to the problem. In particular, research related to the WSP shows that (under the assumption that $P \neq NP$) the NP-complete WSP is efficiently solvable for a growing number of constraint types. The relatively efficient algorithms developed for this purpose assume that the number of tasks is significantly smaller than the number of users who are authorized to perform tasks in the workflow, as well as that all constraints are user-independent. The efforts to stay in the class of FPT problems to solve the problems efficiently should also be considered for the analysis of obstructability. Hence, the envisaged representation to reason on obstructions should allow for an efficient light computation as well. This also means that preferably only constraints that support this goal should be used.

Common Constraints (ROA-4): Because of the aim of this work is to consider obstructions which result from a conflict of the enforcement of security properties and the workflow, the enforceable policies, namely the authorization and further constraints, are considered. Although there are different representations of how authorizations are encoded (e.g., RBAC), they ultimately encode the possible assignments of users to tasks. Therefore, the basic authorization, that is, the basic user-task assignment as it would be represented in an ACL, is sufficient to consider. Against the identified manifold facets of constraints “on top” of the authorization policy, it is not surprising that rules, such as SoD or BoD, which initially appear intuitively plausible, seem rather complicated in the course of the development of the constraint terminologies in WSP research. Here, SoD and BoD constraints will be considered because they are typical user-independent entailment constraints in related work and in practice. Due to the practically oriented focus of this work, the terms “SoD” and “BoD” will continue to be used. However, thereby, the existence of further constraints and constraint classifications needs to be taken into account such that it is possible to integrate respective extensions.

Synthesizing Partial Plans (ROA-5): The approaches to synthesize execution plans seem promising to be adapted to synthesize partial plans, which are needed to indicate obstructive sequences. For instance, the number of such obstructed executions may allow policy designer to assess the given extent of obstructability, when they are related to the number of satisfiable ones. The synthesizing of partial plans may also inspire solutions to find the partial plan to complete an obstructed state (in the sense of liveness).

Considering Costs (ROA-6): To address the identified need to take indicators into account (cf. Chapter 1), obstructability analysis should allow to consider costs. A key question in analyzing and especially handling obstructions is, what needs to be changed or which parts of the policy need to be violated in order to allow for process completion. The investigation on satisfiability and resilience revealed comparable questions, in which an unsatisfiable workflow is assumed, whose policy needs to be changed or violated in order to reach satisfiability or resilience. Because the basic idea of this thesis is to also take a certain degree of violation into account, while still being policy-compliant to allow for more (compliant) behavior, especially the assessment of the violation with a cost is adjustable for obstructability analysis. In literature, the possibility to assign costs to violations is rather coarse-grained, for instance, it is not possible to weigh individual user-task assignments with different costs. More fine-grained approaches would allow for more security-sensitivity, which could be realized by not differentiating between constraints and policy violation, but rather allowing to assess each violation separately. Clearly, this also stresses the aforementioned requirement to consider the order because violation can be history-dependent and order-dependent as well. What the literature on satisfiability and resilience has in common with the handling of obstructability is that it does not consider changing the order of tasks, namely the control flow, because this would change the business goals encoded therein.

Capture the State of Obstruction (ROA-7): For an adequate analysis of obstructability, not only the requirements to perform an comprehensive analysis, but also how the analysis results can be captured and presented, should be taken into account. What all approaches on satisfiability and resilience have in common is that they only sparsely allow to actually specify an obstruction in relation to the information provided from the process specification. Basin et al., for instance, basically regard an unsatisfiable sequence as a partial execution sequence. As a more promising, yet still deficient example, the Bi-Objective WSP models such a partially completed workflow instance by adjusting the policy such that the set of authorized users for each previously executed task is reduced to the user who executed the task. This

is clearly problematic when loops are supposed to be supported as a structural element because a recurring activity could then only be performed by the ever-same person. Further, the fact that an obstruction foremost represents a conflict between the organization and the functional and behavioral aspect, is what also manifests the usual and often reasonable practice of separating the process specification into its different aspects. Hence, related work does not allow to get a big picture to comprehensively represent the overall state of the system regarding an obstructed process. Therefore, obstructability analysis should be able to depict its analysis result, namely identified obstructions, in a comprehensive representation. For this, not only workflows and the policy need to be specified, but also the obstructed state, such that the execution (and (pre-)assignment) history can be encoded, and all existing information is provided. Thereby, it should be allowed to capture the overall PAIS system state regarding the considered workflow and its policies in one comprehensive representation. Because an obstruction identifies a possible weak spot of a policy and its workflow, such a representation can also help the policy designer to better visualize and comprehend, and finally improve the policies.

In conclusion, an approach for the analysis of obstructability is supposed to consider the order in the explained sense, a comprehensive process structure and costs, allow for efficient computation, and to capture and visualize the overall state of obstruction. A representation that meets with these requirements could then also be extendable to investigate and capture satisfiability or resilience as well.

2.2 Process Execution Monitoring: The Case of Obstruction at Runtime

The general notion of execution monitoring “includes security kernels, reference monitors, firewalls, and most other operating system and hardware-based enforcement mechanisms that have appeared in the literature. The targets may be objects, modules, processes, subsystems, or entire systems; the execution steps monitored may range from fine-grained actions (such as memory accesses) to higher-level operations (such as method calls) to operations that change the security-configuration and thus restrict subsequent execution” [182]. This thesis considers execution monitors in the sense of reference monitors that are implemented in a PAIS, allowing or denying access from users to the process tasks (as depicted in the AAA-Figure 1.6 in Chapter 1). Such execution monitoring enforces security requirements during process enactment (cf. BPM lifecycle) and encompasses passive observation and active interception [33] (which relates to the only-observable and the controllable security properties [26]). On the one hand, preventive monitors actively control the

access of users to tasks during execution, thereby enforcing safety properties. In contrast, detective monitors observe the execution and can detect a violating behavior and possibly trigger some sort of mitigation action. The latter, which considers safety properties as well, but may also be able to assess liveness, will be considered in the section on detective analysis (Section 2.3). The core assumption in execution monitoring (in the preventive sense) is that if something unwanted happens, it shall be stopped. While this may be true in a classic access control context, based on the observations in Chapter 1 on the conflicting interests between security and business goals, this may not be desirable on the business layer with respect to business processes in a PAIS. The question of obstructability of security-aware workflows becomes a real danger if a PAIS actually blocks at runtime. It represents the decisive phase to identify, avoid or handle obstructions. Hence, during execution, it is not about the analysis of obstructability but the question is rather how to deal with actually occurring obstructions at runtime.

This chapter discusses which existing approaches there are to actually handle runtime obstructions or related problems. After a closer look at the different process elements and notions during runtime, it will be investigated how PAISs actually enforce the execution. Afterwards, the ways how preventive monitors are able to handle obstructions will be considered in more detail. Thereby, analogously to the previous section on preventive analysis, deficits will be identified such that requirements can be deduced how to allow for a more adequate handling of obstructions. Because an obstruction at runtime results from the process specification, this section strongly builds upon the findings on preventive analysis in the previous section. The selected literature in this section is therefore examined with the focus on the avoidance and handling of obstructions. It does not go again into details on the different aspects found in section 2.1 and the already deduced requirements, for example regarding structural process components or computational complexity.

2.2.1 Process Enactment

The focus of this subchapter is on the execution phase of the process and related entities, such that also the PAIS, which in fact steers the process execution, will be regarded in more detail. Examining the information system in which an obstruction of the process execution happens at runtime, then allows to better relate existing approaches to the overall setting given by a PAIS.

2.2.1.1 Process Execution

The central part of Figure 2.4 depicts the terminology in the phase of enactment (as related in Table 2.1), that is, the point of time of the actual execution of the process. A process is enacted by instantiating activities and executing them in a coordinated manner. This coordination of the activity executions takes place within a certain scope, which is called *case*. A case represents an instance of the process. It encompasses all activity executions that refer to a particular trigger, such as a so-called start event (in BPMN), or a particular input to the system for which the behavior is described by the process [40].

2.2.1.2 PAIS Steering Execution

Based on the configuration phase of the BPM lifecycle, different types of information systems can come into use, which then support the execution of a process. An important criterion here is whether the system is process-aware. Even if this is not the case and the execution of a process is completely manual, information that is created or consumed during the execution of the process (e.g., valuation of collateral and corresponding values) is often stored in a database or a document management system. If one remembers the Anglo-investigation from Chapter 1, for example the email material that revealed some of the fraudulent behavior, there was probably a database system, an e-mail program, a spreadsheet program, or a text editor. Even when such systems do not support an automation or a coordination of activities, the execution of a process can manifest itself in such information systems either way. For instance, the Anglo-emails may reflect the triggering of certain business activities. In this sense, such systems or tools may be used to execute tasks in some business process. However, these tools are not “aware” of the processes they are used in. Therefore, they cannot be actively involved in the management and orchestration of the processes they are used for [3].

Apart from this indirect support of a process, the already introduced PAIS represents a specialized type of information system to support process automation. There are many manifestations of such systems, for instance BPMS (Business Process Management Systems), WFMS (Workflow Management Systems), ERP (Enterprise Resource Planning) systems, CRM (Customer Relationship Management) systems, rule-based systems, call center software or high-end middleware, such as WebSphere. These systems all have in common that there is a process notion present, that they are aware of the processes they support, and that they can be configured in some way (through an explicit process specification, via predefined settings, or using customization) [3]. As briefly explained in Chapter 1, a specific class of PAISs is formed by generic systems that are driven by explicit process models. Examples are BPMS and WFMS. WFMS primarily focus on the automation of

business processes [119]. A WfMS directly implements behavioral and functional aspects defined by the model, creating cases according to the provided blueprint. The basic problems of satisfiability and resilience assume such systems as well, which in practice usually provide basic authorization enforcement, whereas support for authorization constraints in such systems is rare. BPMS has a broader scope: from process automation and process analysis to process management and the organization of work [81]. What BPMS and WFMS have in common is that they both support the coordination of activity executions based on the process specification and allow for process automation.

This thesis assumes a PAIS in the sense of a BPMS because it provides holistic support for the specification, execution, monitoring and auditing of intra- as well as cross-organizational workflows, which also entails the consideration of the different process phases of the BPM lifecycle. The use of such a PAIS does not necessarily mean that all activities are automated. They can still be performed manually. However, the PAIS supports the coordination of the execution. On the one hand, activities to be executed can be selected and assigned to possible users based on the policy. However, outstanding activities can also be made available to users for selection. Thereby, and as a contrast to traditional WfMS, the users are commonly in control of which activity to execute. As a consequence, they may also allow the users to deviate from the process specification given by the underlying process model, thereby providing a degree of flexibility, which is crucial in many application domains to keep a certain room for maneuver [40]. This could, for example, mean that in the collateral evaluation process, depicted in Figure 2.5, an employee could deny or accept the acquisition before the respective collateral market value has been controlled. Although this would not be in line with the process as defined in its model, such a deviation can make sense based on contextual factors that are not captured in the process model. However, in the Anglo case, it would indicate suspicious behavior because the control activity was skipped. Such flexibility, however, does not exclude the case of obstructions because, despite all flexibility, process tasks in the normal case still need to be executed to reach the business goal of the process. Hence, processes have to be completed and the provided flexibility needs to support this. Such process-aware information systems with suitable steering mechanisms can be used, for example, to conduct certain paths defined in the specification and thus meet requirements with regard to the control flow, that is, the interplay of different activities in the process. Hence, a PAIS is able to enforce a plan (a user task assignment), such that, in case it obstructs, it can also enforce a plan to resolve the obstruction.

2.2.2 Avoiding Obstructions

If an obstructable process specification is enacted with such a PAIS, there is the danger of an obstruction factually occurring at runtime. To address this, literature, on the one hand, does not focus on the obstruction but how to avoid it. This has resulted in the development of avoidance strategies, namely preventive monitors that avoid an execution to become obstructed by enforcing only execution plans that are obstruction-free. In particular, this has a strong relation to the synthesization of execution plans as elaborated in Section 2.1 because these plans can be seen as a guide to enforce an assignment of users to tasks that allow for a satisfiable or a likely satisfiable (in case of quantitative resilience) execution. Put differently, obstruction-free enforcement aims to suppress the potential obstructability of a process specification.

2.2.2.1 Enforcing Obstruction-Free Workflows

In analogy to Schneider's enforcement classes, Bertino et al. [30] provide a categorization of authorization constraints in workflow systems into static constraints (enforced at design time), dynamic constraints (enforced at runtime), and hybrid constraints (enforced at design and runtime) [134], which is also useful for obstruction-free policy enforcement. Although obstruction-free policy enforcement may also be enforced by static constraints, the subsequently observed literature is mostly concerned with obstruction-free enforcement during runtime. Basin et al. [24] introduce an algorithm realizing this goal. Thereby, they provide a mechanism to enforce only processes that are obstruction-free (based on a trace-based notation). Crampton et al. [67] present two mechanisms to analyze the realizability of a workflow instance under given access control constraints, which can support authorization enforcement before the execution (static) or during the execution (dynamic) of a workflow. Bertolissi et al. [31] and dos Santos et al. [52], similarly to Basin et al. [24], provide approaches for the automated synthesis of run-time monitors to enforce authorization policies in business processes. In particular, they develop enforcement mechanisms that try to prevent the reaching of an obstructed state. As elaborated before, the field of obstruction-free enforcement has strong interrelations with the aspect of synthesizing execution plans. In essence, it means that the plans that were computed before process execution find their application to enact the process during execution. Therefore, the regarded literature can also be seen as an extension to the approaches presented in Section 2.1.

2.2.3 Handling Obstructions

Even if the satisfiability of a workflow has been analyzed before, if the minimal number of subjects for a resilient execution is known or if the workflow is analyzed to be obstruction-free, and even if, based on all these findings, preventive monitors aim for an obstruction-free execution, exceptional situations, in which the execution of a workflow becomes obstructed anyways, can still suddenly occur. It is not enough to try to avoid obstruction but they need to be handled. Therefore, another direction in literature lies in actually handling obstructions that occur during runtime. In Section 2.1, the analysis of literature already identified approaches that seem useful to actually complete obstructed process executions at runtime as well. For example, the approaches of changing or violating the policy in the preventive analysis can serve as a basis for finding partial execution plans that take a certain degree of risk into account and that can complete runtime obstructions. Such a handling of obstruction has its similarity to the aforementioned obstruction-free monitoring, because it provides execution plans too. These approaches however, require, for example, the change of policies or imply violation, which means that obstruction-freedom has a certain cost, a certain “price to pay” or risk. Further, while obstruction-free enforcement mechanisms focus on valid plans, handling obstructions focus on the executed obstructive partial plan. In order to complete the workflow execution, its aim is to eventually find and append a partial plan to the obstructive partial plan. Beyond that, there are further approaches to handle obstructive situations that originate from the area of access control. There are mainly two approaches to access a certain object for the case when no subject is available [70]: the concept of “Break-Glass”, which often relates to the clinical context in case of emergency, and delegation. Thus, a look at these concepts is first taken, in order to set these in relation to business processes and the associated approaches.

2.2.3.1 “Breaking” the Policy

In a Break-Glass scenario, if the execution blocks an attempted access to an object, it explicitly asks whether it should be resumed. For example, the user that is blocked by the execution monitor must confirm that she or he has exceptional privileges and can be held responsible for access misuse. It is the user, who must balance potential use against harm. If the user “breaks the glass”, the policy violations are recorded along with the further process execution such that after the execution, a so-called post-access evaluation can take place, and traceability and accountability are given. The costs of Break Glass are therefore strongly influenced by (manual) audit costs, which is why there are automation approaches as well [37, 163]. However, this does not significantly address the problem that Break Glass approaches override

initial constraints, disregarding their security related intention or responsibilities of subjects to certain tasks. In a way, Break-Glass defines alternative constraints that take action in the case of an obstruction. The assessment of the cost of overwriting is exclusively dependent on the user who is in full control over the access. The system is not in charge, which involves the risk of user abuse.

Regarding business processes, overriding security constraints to enable a workflow to complete was considered in other works that appear in the literature as well. The approach of Wainer et al. [204], introduced in the context of the WSP, allows to override policies in exceptional situations as well. This is, however, only possible if the overriding users possess the necessary predefined override level, and the affected task does not surpass this, which allows to contain the possible transgressions to a certain extent. Brunel et al. [38] directly incorporate the possibility of violations into the policy. Their approach introduces a violation management, such that a security constraint is allowed to be overridden. However, its overriding implies pre- or post-conditions that need to be fulfilled such that the security policy is eventually met. In a sense, it represents a combination of Wainer's pre-assigned override levels (as a pre-condition) and break-glass post-access evaluation (as a post-condition).

2.2.3.2 Delegation

The other concept of access control to handle missing users is delegation, such that another subject is empowered to access the object (cf. the delegation required by the BaFin in Chapter 1). With regards to the override of initial constraints disregarding their security-related intention or responsibilities of subjects to certain tasks, delegation seems less harmful because the initial constraints of a workflow are mostly retained, except for the right that is delegated to another subject to be able to execute a task. Nevertheless, delegation involves the danger of collusion of subjects and misuse [209] (as seen for example in the Anglo-case). It further requires the delegator to be available to perform the delegation. This raises the question what happens if a user who is able to delegate her or his right unexpectedly becomes unavailable herself or himself so that she or he is unable to delegate her or his right regarding a critical task. The approach of Crampton et al. [70] considers these deficits and suggests the concept of auto-delegation, in which qualifications that indicate a potential delegatee are introduced. Examples on how the so-called qualification hierarchy may be computed based on an RBAC-model are given. In this way, a mechanism automatically resolves user unavailability by delegating a task to the most qualified available user. Because this work is located in the context of access control, the satisfiability and completion of processes is not taken into account. However, the basic idea of Auto-Delegation-Mechanism seems promising for the use in the process context, more specifically, in a PAIS.

With regards to business processes, Crampton et al. were the first to examine the satisfiability of workflow systems while delegating tasks [69]. Bakkali et al. [21] present an approach to bypass situations where obstructions occur by applying a specific delegation process, requiring a manual definition of potential delegates for the respected tasks. These delegates are selected according to their suitability, but may not have the necessary competence or expertise. The focus of the delegation from Bakkali et al., however, is on the task level. This leaves open to what extent a (secure) completion of the entire process is still possible.

2.2.3.3 Changing the Policy

The approach of Basin et al. [25], which was only briefly mentioned before, is now examined in more detail as an example for the type of approaches pursuing the change of policy with an unsatisfiable case at hand. Based on the distinction between administrable (e.g., RBAC), and non-administrable (e.g., SoD/BoD) authorization policies, it tries to solve an unsatisfiable workflow by changing only the administrable policies. The Enforcement Process Existence Problem asks, whether there is an obstruction-free enforcement mechanism that overcomes unsatisfiable workflows by reallocating roles to users at runtime such that the non-administrable policies are satisfied. Based on manually predefined costs of the risk of assigning a new user to a role, maintenance or administration costs, it is possible to determine the cheapest change of the authorization policy. Comparing this approach with the requirements from Section 2.1, the representation of the obstructed state is given only by the obstructed execution trace. It is the basis for finding the alternative policy. This alternative policy however only regards the authorization policy, not the constraints. Thereby, it might happen that an actually well-qualified user is not even considered to execute a pending task because she or he may be blocked by a “non-administrable” SoD constraint, and that a rather unqualified user is added to the role that allows to execute the task, no matter how “costly” this may eventually be. This is comparable to the risk in the approach of Bakkalili et al., where suitable manually predefined delegates that however may not represent the best options regarding competence or expertise are selected.

2.2.3.4 Violating the Policy

Crampton et al. [63] draw the same line of separation between the different policy types as Basin et al., namely authorization policies and constraints. So far, it has been the most sophisticated example of the approaches that allow for policy violation. As elaborated in Section 2.1, they aim to find the “least risky” user-task assignment for a workflow that is unsatisfiable. They assume that security constraints and user-task permissions can be violated, or overridden in order to complete a workflow.

However, they allow to violate both, which results in a bi-objective optimization. In a further approach, there is also the idea of having a certain kind of budget what violations may cost, whereas the costs are still rather coarse-grained. However, with respect to the requirement of a comprehensive workflow structure, neither of the approaches considers loops or conditional branches. In contrast to Basin et al., they do not consider only the obstructed execution trace as input, but to some extent actually allow to model the obstructed state by changing the user-task authorization to singletons that contain the user who executed them, as depicted in Section 2.1. Despite limiting the representation of obstructability analysis result by not capturing all provided information in the obstructed state, encoding the obstruction with a simplification of the authorization policy also limits the set of possible solutions to resolve the obstruction as well, especially when loops are considered.

2.2.4 Completeness

In summary, if it is not possible to avoid an obstruction, and given that an early process termination is not an option, ways how to still complete an obstructed execution must be provided. To give this idea a name, the term of completeness is introduced. Completeness can be seen as the consequence of obstructability. While obstructability is concerned with the detection of obstructions, completeness focuses on the handling of obstructions in order to find solutions to complete obstructed execution. To formulate it as a question, completeness asks: Given an obstructed workflow execution that resulted from the enforcement of a partial plan, is there a partial plan to complete the workflow that meets certain (security) requirements? In order to not allow such a solution to become arbitrary—one could trivially allow for completeness by neglecting all security requirements—it is important that the question of completeness depends on the requirements that such a solution should consider. The basic input for such a solution is given by the process specification that is used by the PAIS.

2.2.4.1 Requirements for Specification-Based Completeness (RSC)

Based on the identified aspects and deficits, requirements for a specification-based approach to resolve and complete obstructions are identified. These requirements build upon the requirements stipulated in Section 2.1. In particular, a comprehensive process model structure should be considered for realistic obstructions at runtime. Further, the fact that runtime obstructions usually need to be handled immediately underlines the necessity to allow for efficient techniques. In the following

requirements for the completion of obstructions, further references are made to the requirements of preventive analysis.

Obstruction-resolving enforcement (RSC-1): An important research direction is the avoidance of obstructions, which strongly bases on the findings from preventive analysis. There are several approaches that aim to provide obstruction-free enforcement mechanisms for which satisfiable plans can be used to ensure an obstruction-free enforcement, which is why the synthesization of runtime monitors is proposed. Despite their focus on prevention, in case of an obstruction, the monitors for obstruction-free enforcement are helpful as well to enforce the resolution of an obstruction. In case an obstruction needs to be fixed, obstruction-free enforcement mechanisms could focus on partial plans that complete the process, starting from the obstruction. In this respect, also plans that involve the least risky assignment regarding violations can be considered. The capability to enact such plans is also reflected in the aforementioned possibilities given by a PAIS.

Security-sensitive Overrides (RSC-2): Despite the identified deficits of Break-Glass approaches, foremost in not considering the existing policy, the general idea to allow to break out of an obstructed state and thereby taking violations into account is also helpful for the handling of obstructions. Further, the idea of Break-Glass to require subsequent inspection of the affected case can still be implemented in more security-sensitive approaches as well, as a means to further improve security. A PAIS can provide such additional mitigating techniques to prioritize audit of the affected case. Hence, there is a need for security-sensitive overrides that take violations of the policy into account. Based on the findings from the requirements from Section 2.1, costs can be used for such an assessment of security-sensitivity.

Automatable Delegation (RSC-3): Although classic delegation is more security-sensitive than Break-Glass approaches because it at least considers a responsible delegator with enough expertise to choose and an appropriate delegatee, it requires a high administrative effort, for example, the availability of the delegator, and involves fraud risks as well. Therefore, there is a need for an automated delegation. Technically, if a security-sensitive alternative assignment is computed, for instance with cost-based approaches, its enforcement can be regarded as the enforcement of an exceptional temporal security policy, which is comparable to a temporal delegation of rights as well.

Obstruction-Aware Completability (RSC-4): The question of completability of an obstructed workflow depends on the information provided to describe the

obstructed situation at hand. In this respect, the existing cost-related approaches that allow for completion of an unsatisfiable case are only able to take the obstructive situation into account to a limited extent. For example, Basin et al. consider only the obstructed execution sequence, Crampton et al. in parts shrink their policy to capture the obstructed state, whereby information for computing a solution is lost. Because the existing approaches do not allow taking the “full picture” of how the obstructive situation arose and only assume a limited representation of an obstruction, it is in turn not possible either for them to offer a solution that takes full account of the course of process execution until the obstruction occurs. The identified requirements from Section 2.1 already attested that the overall state of obstruction in a PAIS must be captured in a better way, including all available information. In turn, this builds the foundation to handle and complete an obstruction adequately because the more information is provided, also the “better” and more security-sensitive solutions can be. Hence, to resolve an obstruction, there is the necessity for an approach that is fully aware of the obstruction.

In conclusion, obstructions at runtime need to be handled. The conflict between the workflow and policy enforcement is to be captured and resolved. Based on the enumerated requirements, to address the deficits of overriding, delegating or aborting the process, this thesis caters to find optimal partial plans, such that eventually, the obstructed case can be completed in a security-sensitive way. To allow for completeability, a PAIS can then enforce such an obstruction-resolving plan to steer the process towards its completion. It therefore requires an efficient automatable approach that allows a violation of the policy while taking the requirements from the preventive analysis, in particular, an extensive representation of the state of obstruction and a comprehensive process structure, into account.

2.3 Detective Process Analysis: The Case of Obstructability by Incompleteness

The detective analysis is related to the “evaluation” phase of the BPM lifecycle and focuses on the recorded process executions. As examined in Chapter 1, process automation goes along with the magnitude of data that is generated in the course of digitization. More specifically, the enterprise information systems (regardless of whether process-aware or not) generate data while the processes are executed, such that in some form, the execution of the process is recorded. According to Bishop [33], logging is “the recording of events or statistics to provide information about system use and performance”. Given a PAIS, process executions can be recorded and captured in a process log. Each activity that executed a task of a process

is recorded as an event that can be assigned to an instance of a process or a case. Such recorded cases, namely the recorded sequences of events, are represented as traces, which constitute the overall process log. Analogously to the process model, a log captures different dimensions as well, for example the control flow or the organizational perspective. The research discipline that builds upon this process entity is captured under the notion of process mining [3]. Detective analysis subsumes not only detective monitoring but, more generally, auditing, which, from a general computer security perspective, is “the analysis of log records to present information about the system in a clear and understandable manner” [33]. Hence, it can be used as a posteriori technique for determining security violations. Whereas detective monitoring implies that a potential action is taken timely, the reaction time for auditing can be significantly longer. However, the execution traces of the process are regarded in both cases. Based on the given trace properties, such process executions can be analyzed as to whether (and how) the designated business goals are achieved (liveness) and whether policies were adhered to (safety). Hence, whereas the so far regarded literature mainly covers safety properties and the implications of their enforcement, logs additionally provide the possibility of checking for liveness properties. In particular, they allow to consider completed “closed” cases, which allow to assess if liveness properties were indeed “eventually” fulfilled. Regarding the observation of obstructions, this is mainly interesting for property of completion, for example if an end activity could be reached. If this is not the case, the obstructability of the process involved is indicated by incompleteness, i.e., traces that do not represent completed executions due to obstructions.

In general, the main difference to the modeling and specification of a process is that the log actually represents how the process was actually executed, or “how it was lived”. In contrast, a process model rather aims at a comprehensive view of a process and generalizes individual cases of the real world process [40]. The basis for the WSP builds the process model and policy specification. Nevertheless, the use of logs for satisfiability, resilience and obstructability research can be well-motivated because there are manifold reasons to use their so far untapped potential, for example: Regarding the WSP, the log can help to assess to what extent specific satisfiability problems are actually relevant, such that it is able to investigate the importance of individual paths of a workflow. Regarding resilience, besides assessing the importance of specific paths and its relevance, the log can reveal the probability of a user to be available. The log further offers the possibility to simplify the computational complexity the computation because it represents a finite set of events and traces. Regarding the computational complexity, because a log represents a finite set of events and traces, if it is replayable on a model, to identify if, for this finite set of traces, it is still satisfiable. Thereby, loops would be restricted

to the finite set of realistic events given by the log. This may, for instance, give insights on how changes to policy design impact the conduct of the lived process. Regarding obstructability, at first sight, logs in fact “limp a step behind” in handling obstruction during runtime. However, they provide a basis to learn from completing obstructions. On the one hand, a log contains obstructed executions which manifest themselves in an incomplete or aborted trace that can result from an obstructive policy design or exceptional user absence situations. On the other hand, the log can contain complete and successful traces, or is even able to document how an obstructive situation has been resolved. Regarding the latter, depending on the flexibility a PAIS allows, or, in other words, how strictly it insists on adherence to the control flow, the log reveals important and practically relevant insights, which may differ from the control flow of the model, for example because further contextual factors beyond the model and specification were taken into account. That way, the log may encompass completed traces that represent compliant behavior that deviates from given security policies and properties. Further, a trace of a completed execution that involves violations may also result from a Break-Glass scenario, which, however, may have been checked by audit and was subsequently assessed and marked for being without concern. In this case, also the log would capture behavior of completed, compliant executions that deviate from the initial process specification. The log may further be useful if an information system does not support the modeling and enforcement of authorization constraints. Indeed, the lack of controls during runtime, as show in Chapter 1 (cf. ACFE results), is often reflected in the fact that preventive controls are not always used due to the associated costs and the sometimes negative effects on process execution. The risk of deviating execution paths is accepted and the compliance check is shifted to audit analyses. Although there are attempts to integrate monitor synthesis techniques, such constraints are often specified separately and handled by auditing software (cf. CSI tools), whose main goal is to detect problems a posteriori. However, even in case of no control and authorization enforcement during the execution, logs are of beneficial use. An actual obstruction during process execution would then not block the process. The involved users may, however, be aware of an actually obstructive situation based on contextual information (e.g., known regulatory rules). After extracting the data of such a system into a log-file, a subsequent analysis of the respective log is able to reveal if such a situation was at hand and how it was handled, namely if the process was blocked, aborted or if there was some way to overcome an obstruction. The log would then give insights to indicate weak spots of the process, for example risky or failing workarounds. On the other hand, this could also reveal successful workarounds, which may even help to resolve and guide other obstructive situations. Thereby, a user who is aware of an obstructive situation could, for example,

be assisted by a system that recommends how to proceed, based on the insights provided by the log.

These manifold reasons further reveal that it seems natural and beneficial to differentiate between successful and obstructed log executions, which can be done, for example by analyzing the liveness property of process completion. This allows to reason on the causes of successful executions and even to guide a log-based handling of obstructions. Moreover, the log allows to derive further indicators, for example, Resource Behavior Indicators (RBI) [164] or indicators that are used in predictive monitoring [143], which helps in assessing the risk of violation in case of an obstruction and can enrich the process model. Such information improves the overall level of information how to handle and complete obstructions.

While most of the so far regarded related work is aimed at theoretical preventive observations, the use of logs for a practical application to approach (un)satisfiability is hardly ever considered. As far as known, there is only one further approach that relates logs and process mining to the extended context of the workflow satisfiability problem [52]. It is used to preprocess and reconstruct a control flow model such that a user subsequently defines the policy on top, which, however, represents a different focus than this work does. This is the first work that aims to use logs for the analysis and handling of obstructions. Taking “real” runtime obstructions and “real” solutions into account stresses its practically oriented focus. For this, this chapter will systematically draw potentials from the use of logs regarding obstructability. Based on the possibilities of process mining, general ways of how logs can be used beneficially to analyze and resolve obstruction will subsequently be identified. After a short look on event logs, these possibilities will be identified and elaborated further along the threefold disciplines of process mining, namely process discovery, conformance checking and enhancement. Finally, the potentials and requirements for a log-based approach will be deduced.

2.3.1 Process Logs

When a process is supported by information systems, details of the execution of the process are generally available in the form of *event data*. Although PAISs directly provide event logs, as mentioned before, there are many information systems that store such information in unstructured form (for example databases), such that event data can be distributed over many tables or need to be retrieved from subsystems that exchange messages. In such cases, event data exist, but some effort is required to extract them, such that data extraction is an essential part of any process mining endeavor [3]. After getting and extracting the data, possibly from different sources,

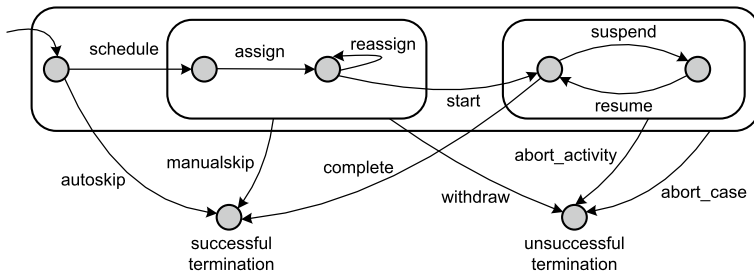
Table 2.2 Example DMV trace

Case	Activity	User
21	Compute Market Value	Bob
21	Control Computation	Alice

these data take the form of an *event log*. Event logs represent the footprints left by process executions that were stored by an information system [3] (e.g., a PAIS). As shown in Figure 2.4, they consist of a collection of *events* that record which activity for which case was executed. Thus, an event log depicts the recorded behavior of a process. Events can be distinguished by the cases in which the respective activities were executed. This results in event sequences, designated as *traces*. They represent the recorded behavior for the individual cases of the process. Table 2.2 displays an example of such a trace. Accordingly, a trace is a recorded representation of a case of the process, analogously to an execution sequence of a process model, which is a modeled representation of a case [40]. This section takes a closer look on process logs, in particular on what they are able to capture, and which ways are offered to detect obstructions.

2.3.1.1 Formats

Event logs are the core ingredient for process mining algorithms. They exist in different formats. After the Mining eXtensible Markup Language (MXML), its successor, the XES format, was established. XES is an XML-based format for the interchange of event log data between tools and application domains, which is approved by the IEEE as the Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams (1849-2016) [115].

**Figure 2.9** Standard transactional life-cycle model [3]

Analogously to the described basic structure, an XES document represents an XML file and contains a log consisting of any number of traces. Each trace describes a sequential list of events that are assigned to a particular case. The log, its traces, and its events can have any number of attributes that can be nested in each other. No fixed set of mandatory attributes is required for each element (log, trace, and event). However, to provide semantics for such attributes, the log refers to so-called extensions. Each extension can define attributes that are considered standard when the extension is used. XES can declare certain attributes as mandatory fields. For example, it can be specified that each trace should have a name. Thus, not every possible attribute must be contained in a log [3]. In logs of higher granularity, different information on the state of an activity execution can be captured as well. This transactional information on activity instances can have different state attributes according to the standard transactional model, which is displayed in the state machine in Figure 2.9. These are of considerable interest when filtering the log for potentially obstructed or completed traces in the sense of unsuccessful or successful traces respectively.

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <log xes.version="1849.2016" xes.features="">
3 <extension name="Concept" prefix="concept" uri="http://.../
  concept.xesext"/>
4 <extension name="Organizational" prefix="org" uri="http://.../org.xesext"/>
5 <extension name="Lifecycle" prefix="lifecycle" uri="http://.../lifecycle.xesext"/>
6 <global scope="trace">
7 <string key="concept:name" value=""/>
8 </global>
9 <global scope="event">
10 <string key="concept:name" value=""/>
11 <string key="org:resource" value=""/>
12 <string key="lifecycle:transition" value=""/>
13 </global>
14 <classifier name="Resource classifier" keys="org:resource"/>
15 <classifier name="Activity classifier" keys="concept:name
  lifecycle:transition" />
16 <string key="concept:name" value="DMV Log"/>
17 <trace>
18 <string key="concept:name" value="22"/>
19 <event>
20 <string key="org:resource" value="System"/>
21 <string key="lifecycle:transition" value="schedule"/>
22 <string key="concept:name" value="Compute Market Value"/>
23 </event>
24 <event>
25 <string key="org:resource" value="Alice"/>
26 <string key="lifecycle:transition" value="start"/>
27 <string key="concept:name" value="Compute Market Value"/>
28 </event>
29 <event>
30 <string key="org:resource" value="Alice"/>
31 <string key="lifecycle:transition" value="complete"/>

```

```

32 <string key="concept:name" value="Compute Market Value"/>
33 </event>
34 <event>
35 <string key="org:resource" value="System"/>
36 <string key="lifecycle:transition" value="schedule"/>
37 <string key="concept:name" value="Control Computation"/>
38 </event>
39 <event>
40 <string key="org:resource" value="System"/>
41 <string key="lifecycle:transition" value="pi_abort"/>
42 <string key="concept:name" value="Control Computation"/>
43 </event>
44 </trace>
45 <trace>
46 <string key="concept:name" value="23"/>
47 ...
48 </trace>
49 ...
50 </log>

```

Listing 2.1 XES example of DMV trace

Listing 2.1 shows an XES version of a further example case of the DMV process. Here, the organizational extension defines a resource attribute of type `xs:string` (e.g., for “Alice”). Further, this log also provides the lifecycle extension that allows to represent the status of each activity execution. In this way, the process abortion that occurred in case 22 can be documented (see line 41). Hence, depending on the information recorded during process enactment, a log file is able to capture obstructed and successful traces in different levels of detail.

2.3.1.2 Filtering

It is common practice to filter the logs before applying process mining techniques. Filtering is basically first used to clean up the log so that it does not contain any erroneous traces, for instance, by minimizing the noise [40]. There are systematic approaches to identify noise patterns [192] as well as to filter and clean event data [53, 205]. In this respect, the term of “incomplete logging” is also related to errors that happen during the recording of process data, for example missing activities distributed along the entire process execution. It is important to note that the aim to filter traces that are incomplete—in the sense of not completely executed—represents a notion of incompleteness that should not be confused with the term “incomplete logging”. In this context “fragmentary” or “gappy logging” would be a more differentiated description of what is often meant by “incomplete logging”. In contrast, in the sense of an obstructed execution, incomplete means “not completed”, with respect to the flow of activities, and is an error-free recording of a partial trace.

Endpoints Filter: A log usually represents an excerpt from the system log over a certain period of time. Due to this selected time period, there may be incomplete traces whose start or end activity lie outside the selected scope. It must therefore be ensured that traces whose beginning or end are “cut off” are filtered out as well. In this respect, for example, the so-called “endpoints filter” allows to determine what should be the first and the last event in a process case. With regard to the identification of obstructions, given a basically filtered trace that does not contain any traces that were cut off due to the selection of the log interval anymore, the endpoint filter is still of interest. It can also be used to filter traces that fulfill the liveness property of process completion scanning the traces with regards to the occurrence of an end event. That way, if the end activity that distinctively characterizes a completed process is known, complete and incomplete traces can be separated. However, this does not necessarily mean that incomplete traces are also obstructed traces. Therefore, in order to increase the likelihood of actually filtering obstructed traces if a log provides more detailed information, further and more fine-grained filtering can be conducted.

Attribute-Based Filtering: Based on the attributes given by the XES standard, further fine-grained filtering can be done in order to split the log into aborted or completed cases. As indicated in Listing 2.1, the transactional model provides state attributes that allow to indicate that a case or process instance was aborted (cf. `abort_case` or in XES: `pi_abort` (see line 41)). Further, an activity that was only “scheduled” (see line 36 in Listing 2.1), but has not been assigned yet can indicate an obstruction as well, when the assignment was not allowed due to the policy or when it was not possible due to unavailable resources. Filtering traces in this way, is a first indicator that incomplete traces were actually aborted due to an obstruction. Conformance checking, which will be highlighted as a process mining-method in the next section, is able to go one step further. For example, based on the given policy, it is able to indicate if SoD conflicts were involved within an aborted or only scheduled case. This would exclude cases that were aborted for other reasons, for example system failure, and it would substantiate the suspicion that the regarded traces were indeed obstructed due to the enforcement of safety properties.

To conclude, filtering in its common application is often used, for example, to select only the most frequent event logs in order to simplify the application of subsequent process mining methods and to lighten their computation and to strengthen the significance of the results in the light of a specific question. This whole process has an iterative nature because process mining results most likely trigger new questions and these questions may lead to the exploration of new data sources or more detailed data extractions. Typically, several iterations of the extraction,

filtering, and mining phases are needed. In this context, the underlying questions of Process Mining can also be seen from a business intelligence perspective [3]. A concrete question is addressed to the logs, for example, how the process performs, whereupon the log is filtered against this question and analyzed. The same applies to the use of logs in the security context, or in the sense of this thesis in identifying and dealing with obstructions. The following methods of process mining will be introduced briefly, but will then be specifically considered with regards to their possible advantages for the detection and handling of obstructions in security-aware workflows.

2.3.2 Process Mining

Process Mining addresses both processes and data, which are fundamental elements of digitization. It is therefore a fairly young research discipline, and can be located between machine learning and data mining on one side and process modeling and analysis on the other. The basic idea of Process Mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event records that are easily accessible in today's systems [3]. It can essentially be subdivided into three disciplines, as shown in Figure 2.10. This section presents a brief overview of the process mining methods and relates it to process security and obstructability.

2.3.2.1 Process Discovery

Process Discovery has the goal of discovering a process model based on logs without the use of any a-priori information. If the event log provides further information, for instance about resources, it is possible to discover resource-related models, such as a social network that reveals how people work together in an organization [3]. Discovery approaches that are related to security aim to reconstruct models that are as precise as possible, to not rule out possibly rare but important deviations that involve suspicious facts and indicate violations [189]. Challenges here are clearly the aforementioned incompleteness or noise, which affects the log but represents behavior that did not actually happen. Regarding obstructions, this challenge also applies for the distinction between erroneous/incomplete and uncompleted (in the sense of unsuccessful) executions.

Hence, given that noise and error are eliminated, analyzing a discovered model allows to reveal unsuccessful executions, for instance, remarkable paths in the model that noticeably skip the usual activities and come to an abrupt end. On the other hand, based on previous filtering, the discovery technique can focus on successful

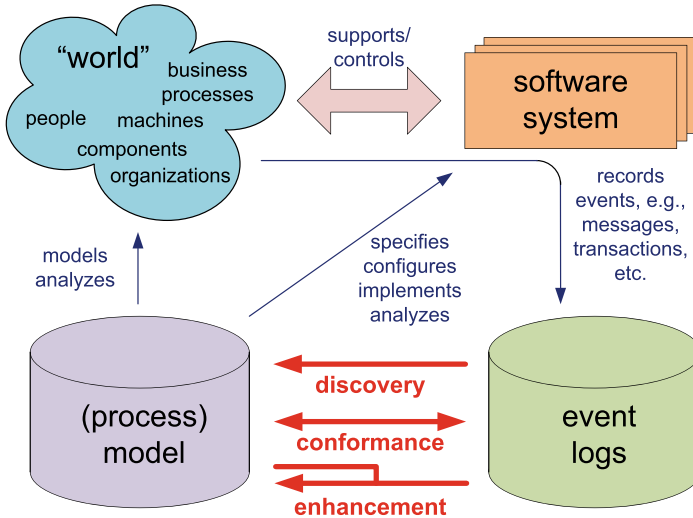


Figure 2.10 Positioning of the three main types of process mining: discovery, conformance, and enhancement [3]

executions, such that the discovered model of successful traces is used to identify obstructions or to show which paths there are to avoid or even handle them. Such comparisons of log and model already belong to the subsequent type of process mining.

2.3.2.2 Conformance Checking

Conformance checking bases on a discovered or a manually defined model. It compares a process model with an event log of the same process and can be used to confirm that the reality recorded in the log is consistent with the model, and vice versa. For instance, the SoD constraint states that the computation of the market value and its control need to be done by two different people. By scanning the event log using a model that defines this requirement, a violation, and thereby, potential fraud of the actors involved in the found traces, can be detected. Therefore, conformance checking can be performed to identify, detect and refine anomalies and measure their severity [3]. Checking for security properties may therefore be performed by means of conformance checking. Clearly, as a basis for a precise analysis of security requirements with conformance checking, process discovery also need to be precise to capture important security aspects, for example deviations, as well,

and to not rule out possibly rare but suspicious paths that indicate violations. More specifically, based on the comparison of model and log, there are three general so-called conformance checking artifacts, which indicate consistent and deviating parts: rule checking, token replay, or alignments. The indicated example on SoD represents such behavioral rules, which are defined by the model and violated by some traces of the event log. As to the replay, events of traces can either be replayed by task executions in the process model, or the replay fails. An alignment tries to “align” events from a trace of the event log with the task executions of an execution sequence from the model [40].

Identifying Obstructability ex-post: Because the traces in the log represent the actual behavior how the process was lived, the process log can be depicted similarly to the overall behavioral scope in Figure 2.14. Conformance checking can then be illustrated as drawing the lines in an unclassified behavioral scope given by the log according to given requirements such that, for example, the secure, compliant and non-compliant areas are identified. The separation of consistent and deviating parts with regards to certain requirements can be used to separate successful and obstructed traces as well, which allows to reason on the obstructability of the process and how it is handled. For example, by checking if there are traces in which a liveness property, for instance, a finalizing activity, is missing (rule checking), by replaying traces on a model to indicate non-replayable and potentially obstructed traces, or, in a more fine-grained way, also alignments can be used to indicate successful and obstructed traces. Alignments can be assessed against different metrics, most prominently fitness and precision. Fitness investigates how much behavior of the log is captured by the model, precision asks how accurate the model describes the log. Depending on the granularity of the log, successful traces can also be traces that only come close to the model, but still represent a complete execution. Here, execution sequences based on a model that addresses the requirements stipulated for the preventive analysis in Section 2.1 may increase the significance of the alignments computed with a given obstructed trace.

The identified categorization and separations in the Process Mining context can build the basis for further analysis. For example, by identifying traces that do not reach the end of an execution, indicators on possible related problems can be investigated in comparing them to successful ones. Clearly, based on such a separation, discovery techniques can identify a successful and an obstructed model to identify problems in the process or policy specification as well.

2.3.2.3 Enhancement

Enhancement represents the third type of process mining. Its overall idea is to extend or improve a process by using information about the actual process recorded in some event log. On the one hand, the model or the log can be repaired (i.e., improved), based on the findings of discovery or conformance checking. On the other hand, the model or the log can be extended or enriched with further information based on such and further findings. Repairs or extensions can also be intertwined and work in both directions, the log improves and repairs the model, and vice versa. Log enhancement can therefore enrich the events of a log with additional information, which can then be used for further techniques for a log-driven analysis of the regarded process. Such information originates from a process model or some analysis conducted based on a process model, for example, using the labeling of activities with responsible roles or the probability to complete the process. The enhancement of the model enriches the model based on the log, which enables further types of model-driven analysis. A typical example is, if two activities are modeled sequentially but can happen in any order in reality, the model is corrected to reflect this [40].

Extension can mean adding a new perspective related to the log. A prominent example is the extension of a model with performance data. For instance, by using timestamps in the event log, one can extend the model to show bottlenecks, service levels, throughput times, and frequencies. In particular, the model can be enriched with the duration of the activity executions, such that a distribution is fitted to the execution times recorded in the log per activity. This information enhances the process model and, for instance, enables performance simulation and prediction. In this way, logs can be used to tackle the gap of not knowing what is going to happen next in the process execution and they can help in better defining probabilities that certain events occur. Further, a model can be extended with information about resources, decision rules, quality metrics, etc. [5].

Hence, regarding this thesis, on the one hand, “extensions” seem useful to enrich models or logs, thus enabling the required indicators to be taken into account. On the other hand, “repairing” also offers interesting approaches, which are beneficial for the completion (or “repairing”) of obstructed executions.

Extensions to Capture Indicators: Extension, in particular the extension of the model, are of interest for the consideration of an indicator-based security. Here, it is of interest to mainly consider security relevant indicators. There is a wide range of indicators that can be derived from logs [17, 164, 165]. Besides, the prominent Key Performance Indicators (KPIs), mining resource profiles and related indicators have raised a lot of interest in recent years. Regarding this work, the Resource Behavior Indicators (RBIs) are subsequently sketched as a particularly suitable

example because resource behavior is also relevant for security and completability. Different taxonomies cover the different behavioral aspects of resources, which are exemplified in Figure 2.11. When it comes to resilience, satisfiability and obstruction resolution, user reliability can, for example, be considered as an important factor to assign the most reliable users to an already “ailing” execution in order to ensure its completion. Such a reliability indicator lessens the risk that an employee is likely to be involved in security violations. That way, the process model can be enriched with indicators that preferably assess policy violations and the associated risks.

Ultimately, the computation and the weighting of different indicators can result in a final number expressing the overall risk. This can be done not only with regard to the users but also to the tasks to be performed. A model which is able to capture these indicators would create a framework for an indicator-based security and would enable a security-sensitive and differentiated view on violations.

Repair: Fixing the process specification: In general, repairing can be understood in the sense of repairing the model such that it better reflects reality. On the one hand, repairing can be useful for policy designers who want to fix obstructions on the basis of the insights from process mining. Based on the separation into successful and obstructed logs and further conformance checking, weaknesses in the specification, for example, in policy design, can be uncovered. Further, if certain risky paths that allow obstructions do not occur in the log, the process designer may consider

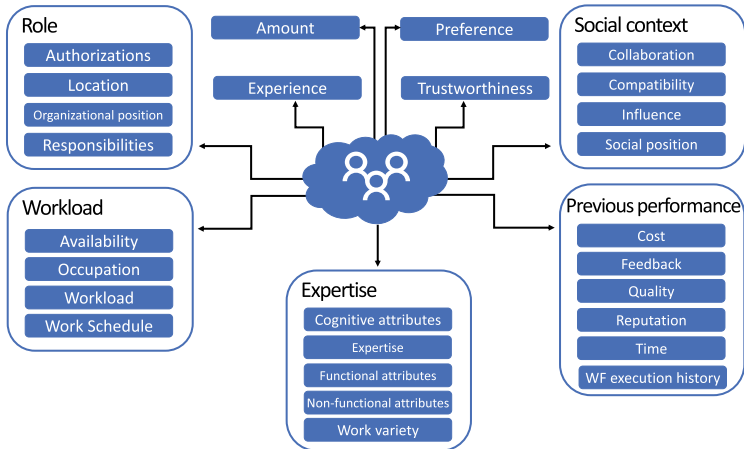


Figure 2.11 Using logs to deduce resource indicators [17]

changing or adapting the model. For example, it is possible to repair an SoD policy in a way that makes the overall policy more restrictive and thus prevents obstructions during the execution by design. To do this, two activities for which an SoD conflict exists can be assigned to only different users, so that no user is authorized for both activities at all (which would, however, negatively impede the flexibility of the process execution). In a broader sense, repairing may also be understood as the fixing of an obstructed path in the model, such that based on the log, either the path is changed, or it is extended such that it can still be completed.

Based on the log that contains successful executions and an obstructed trace, the question is how this trace can be repaired such that it is completed with minimal violation. Repairing can also be understood as a repair at runtime, where the question would be how an obstructed execution can be completed. It is then not a question of repairing the whole model, but only an execution sequence or a case. In this respect, a further distinction of process mining is introduced, regarding the point of time of its application, namely online and offline process mining.

2.3.2.4 Offline and Online Process Mining

The traditional way of using process mining is offline. This means that only closed cases are taken into account, that is, the event log contains complete traces corresponding to the cases completely processed in the past. However, for operational support, for example, the handling of obstructions during execution, it is necessary to consider “live” event data and to provide an online response to these data. The basic idea here is to only consider cases that are currently still running, because these can also be influenced and still generate events. They are described by partial traces [3].

The basic setting of these online process mining approaches is that there is some partial trace of a running execution. Based on that partial trace, an operational support system considers insights from the log to find violations or make predictions and recommendations (cf. Figure 2.12). In particular, the log is used to learn a normative, predictive or recommendation model, which then builds the basis for the operational support system to put the given partial trace into the perspective of past executions. Interestingly, this basic setting is comparable to the basic situation of this thesis in handling obstructions. On the one hand, there is a partial trace that is obstructed, and on the other hand, the approach is supposed to provide, in a sense “predict” or “recommend” a security-sensitive partial trace to still complete the execution. Various techniques can be used to generate predictions, for example, techniques from supervised learning. On the basis of the information contained in the partial trace and a prediction model, predictions can be derived, for instance, regarding a KPI such as the remaining flow time or the expected total costs. The

predictive model is based on historic event data, but can be used to make predictions for the cases that are still running. Recommendations base on predictions as well [3].

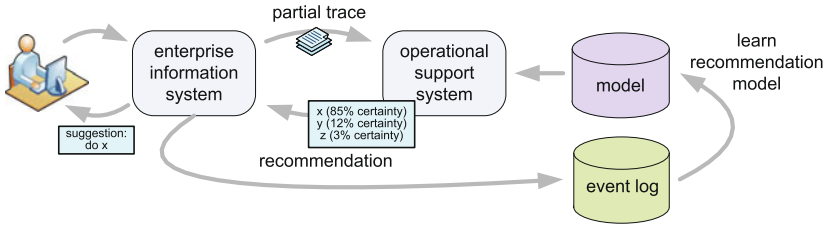


Figure 2.12 Recommendation [3]

Instead of focusing on the model and the policies for the execution monitoring as done before, logs can also be used to check executions in terms of a monitor. Thus, not only the preventive analysis, but also the detective view can be used as a basis for monitoring. Here, the previously identified detective monitoring comes into play. It is related to process enhancement in a sense that the process is extended with additional information, but discovery and conformance checking can also be involved. Therefore, predictive monitoring will be introduced in the following.

2.3.2.5 Predictive Monitoring

A sub-discipline of online process mining that is particularly noteworthy against the background of this work is predictive monitoring. Predictive business process monitoring techniques go beyond traditional ones by predicting quantifiable metrics about the future state of the running instance of a business process (i.e., the cases) [143]. There are different questions that want to be predicted: What will probably be the next activity that will be executed? Will there be violations in the execution? How long will the overall process take or will the remaining time stay below a certain bound (e.g., for a loan application)? Or, what will be the results of the process (e.g., will a client purchase an item or not, or more generally, the achievement of the business goal)?

More specifically, as depicted in Figure 2.13, predictive monitoring assumes a partial trace for which predictions about the future can be made on the basis of the process log. The event log is the input of these methods and provides the necessary characteristics that define the process for the prediction. The predicted value represents the output of these methods and applies either to the current process instance or to a collection of instances. Depending on the target of the prediction, this

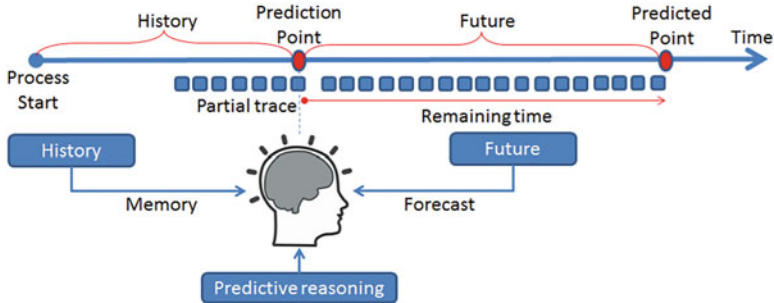


Figure 2.13 Predictive Process Monitoring [203]

value belongs to a particular domain and can be numeric (e.g., the remaining time of a process), Boolean (i.e., regarding an outcome, e.g., the fulfillment of a particular goal), or categorical (e.g., regarding a user) [144]. Related literature objects to predict a wide range of values, among which are time, foremost remaining execution time, the prediction of the next event in the given case [91, 166, 195], an estimation of the value of a single indicator or an aggregate attribute, LTL formulas, which determine the occurrence of a certain situation in the process, or the risk probability (e.g., the violation of a constraint or abnormal termination) or in general, the final outcome of a case with respect to a possible set of business outcomes [81, 84, 85, 143, 149, 150]. Regarding the latter, each running case of a process can be classified according to a given set of possible categorical outcomes [202]. For instance, a possible outcome of a case may be that a collateral evaluation is finally completed (e.g., the acquisition was finally approved). On the other hand, it could also be closed unsatisfactorily (e.g., the evaluation was aborted and the process goal was not reached). Another further outcome that could be considered would be if the collateral evaluation was performed in a specific time (with respect to a maximum acceptable waiting time for the overall process). Predictions can be used to alert process users to problematic cases or to support the assignment of resources, for example, assigning additional resources to risky instances [144, 201]. Hence, such an outcome may be the fulfillment of a compliance rule, a performance objective (e.g., maximum allowed cycle time) or business goal, or any other characteristic of a case that can be determined upon its completion.

Predicting Obstructions: The questions of predictive monitoring can be related to the question of obstructability: Will the process be successfully executed or will it get obstructed? Will there be enough users to execute the process? Will the current par-

tial trace be obstructed in the course of its continuation due to missing authorizations or can the case even come to an end? Or, will a positive outcome of an obstructed execution be achieved? The similarities of questions make predictive monitoring particularly interesting, especially the outcome-oriented version. Particularly, predictions on the outcome in terms of fulfillment or violation of security properties can be performed. Further, the prediction of the process termination can determine the probability of the fulfillment of a liveness property. In the sense of operational supports, such an online on-the-fly conformance checking can not enforce liveness or safety properties, but allows to check them very promptly. Thus, even if, as previously explained, liveness can not be enforced in the classical sense, one can at least increase the probability of the fulfillment of liveness properties with corresponding prediction or recommendation methods. In this respect, the log may give insights on which sequences involving which actors and which activities are most likely to succeed. Therefore, an estimation can be made, which allocation of users to tasks will probably also “enforce” process termination. In turn, also problematic execution paths, for example due to unsatisfiable policies, could be identified, for example by relating the possible paths to their rate of completion.

Recommending Obstruction Resolution: From the view of the handling of obstructions, similarly to previous observations in this chapter, the fundamental deficit of the current approaches to predictive monitoring is the tendency that they are basically “avoidance approaches” as well. Prediction techniques are used to avoid undesired states of the process, e.g., security violations, or also obstructed process executions. For example, there are techniques to maximize the probability that a process execution satisfies all constraints, or, to identify if it is likely that a possible execution path takes too long, such that another path with a better prediction can be chosen. Hence, this can be compared to the goal of preventive analysis, which tries to avoid unsatisfiable workflows, or the enforcement of obstruction-free authorizations during runtime. Indeed, to some extent, predictive monitoring can be compared to the approach during runtime to use synthesized plans, to guide and predict the execution of a process in terms of following an obstruction-free path. The question is to what extent a predictive monitor really enforces its predictions, which is comparable to the obstruction-free enforcement mechanisms presented before, or if obstruction-free predictions are only recommendations such that the user may eventually choose which path to follow. Because predictions also allow for proactive and corrective actions to improve process performance and mitigate risks, the so-called “prescriptive process monitoring” goes one step further. Prescriptive process monitoring not only wants to predict that process executions may result in an undesirable process state such as a security violation but also seeks

to prevent this. It extends the scope of purely predictive systems by not only generating predictions but also advising users in case of an instance being likely to lead to an undesired outcome if and how they should intervene in an ongoing case. Such an undesired outcome can be prevented or mitigated, for example, by optimizing a given utility function [201]. However, prescriptive process monitoring still represents a rather observational procedure. It is still an avoidance strategy, albeit a more proactive one. The ideas of correction and intervention are rather meant to influence the development of process execution on the basis of predictions in such a way that the defined goals can be achieved as far as possible, which can only happen within the scope of action set by the process model. Further, as the name suggests, prescriptive monitoring additionally “prescribes” in advance how to react to possible undesirable developments and defines key indicators or thresholds, which should alert the responsible users in case they are exceeded, so that they can then take corrective action according to the correction prescribed in advance. It is not considered to “touch” (or even violate) the process specification. The questions of what to do in the event of an actual process obstruction, and how obstructed traces are fixed, which, as explained, can occur despite all predicted probabilities, are not dealt with. Nevertheless, due to the comparability of the questions posed regarding obstructability, the methods of predictive monitoring seem promising to resolve obstructions as well. Due to the abundance of such methods, which is underlined in two recent literature reviews [144, 202], it is foremost a matter of showing the fundamental feasibility of the adaptation of the methods to the aims of this work. Predictive monitoring is therefore of considerable interest for the handling of obstructions; on the one hand regarding the indicators that are considered, on the other hand, regarding the approaches that are used to generate predictions such that they may inspire solutions how to complete an obstructed trace based on the log.

2.3.3 Log-Based Completeness Requirements (RLC)

In conclusion, it has been identified that the log offers the potential to create a different approach to satisfiability, resilience and obstructability. Due to the lack of log-based techniques in the given context, this section on detective analysis has not identified deficits of existing techniques, but revealed potentials for the application of logs. Based on these observations and the observed potentials, log-based approaches should consider the following possibilities and requirements for the analysis and handling of obstructions:

Detect and Separate Obstructed and Successful Traces (RLC-1): In order to be able to use logs in the context of this thesis, it is necessary to have methods to separate obstructed traces from successful ones. This can be done by attribute-based trace filtering, with process discovery, in which an obstructed trace is captured in a path that bypasses other essential activities, or in a more precise way with the help of conformance checking. Thereby, logs can be separated in obstructed and successful executions. Based on this separation, an obstruction can be put into perspective. It is possible to assess the obstructability of the process, even if the process was not controlled by a PAIS during execution.

Obstructed traces can be used to identify deficits in the process specification. One can think of the reasons why the obstructed traces were obstructed to improve the process. Further, they can be used to assess the probability of a preventively detected obstruction to occur in reality. Successful logs can be used to guide how the policy is to be improved or for the completion of a running instance, for example, with success rates regarding an outcome in predictive monitoring, e.g., process completion, or by reasoning on how they might be used to complete partial traces. Further, a discovered model based on the successful logs can guide the completion in case of an obstruction as well.

Identifying and quantifying indicators: Assign Costs Based on Log (RLC-2): As identified, the log can be used to identify manifold indicators. Methods of filtering and conformance checking but also process enhancement and predictive monitoring can be used for this. These indicators can then be considered and used to determine an execution sequence that can complete an obstructed execution as security-sensitive as possible. This, in turn, underlines that the model must be able to consider costs, such that quantifiable indicators can actually enhance it. In the light of the requirements for a representation, as specified in Sections 2.1 and 2.2, the extension of the model with further information has similarities to the previously identified requirement to consider costs, for example in finding executions scenarios for satisfiable processes. What would be new here, in contrast to the cost-based approaches shown in the previous sections, is that the model would directly be extended with the costs. Hence, the need to capture costs, was now determined before, during and after execution as a requirement for all approaches.

Proposing measures: Finding paths to complete obstructions (RLC-3): How does a log-based approach need to be designed to not only detect but to provide measures to complete obstructed executions? As identified, logs can also be used to recommend actions based on the behavior they reveal. This generally meets the approach of this thesis on how to deal with obstructions, so that the basis for the

required rational decision is extended with data. Here, although online-process mining has a similarity to the handling of obstructions during execution monitoring (cf. Section 2.2), the log-based approach has not yet been considered for the handling and completion of obstructions. Therefore, methods already used in predictive monitoring are meant to inspire solutions to resolve obstructions. In particular, among other goals, predictive monitoring uses logs to complete processes (as a positive outcome). This and involved log-based techniques represent a starting point to resolve obstructions as well. Here, first and foremost, the basic practicability of using logs to resolve obstructions needs to be shown.

To conclude, this work is meant to also consider logs to unleash their potential in detecting and handling obstructions. There are manifold ways to separate completed traces from obstructed ones, for example by analyzing safety and liveness properties. Based on this, it is possible to derive meaningful indicators, which can be considered as costs when solutions are calculated for an indicator-based security. These costs can then be incorporated in the representation required in Section 2.1 and 2.2. Here, completing obstructed traces, in particular the adaption of methods provided from predictive monitoring seems promising to actually tackle obstructions based on logs. Such a completed trace would possibly violate a safety property such as an SoD requirement, but would eventually allow the liveness to be “enforced”. Depending on the information system that is used to execute a process, the policy or process model is not necessarily enforced and it may happen that only logs are available. Therefore, the log-based approach should stand for its own, but at the same time it should be able to be used to complement the specification-based approach.

2.4 Security-Sensitive Detection and Handling of Obstructions

The previous sections of this chapter identified that engineering methods that can handle unexpected process “obstructions” is a hardly touched research field. However, it is very relevant in practice, because it contributes to the construction of reliable, secure enterprise information systems [8].

There are different research directions ultimately trying to improve and finally avoid an obstructive state. Besides the extensive research, which has been conducted with respect to the analysis of satisfiability and resilience before process execution, mainly research topics concerning the enforcement of obstruction-free executions or other forms of an avoidance of obstructions were identified. Interestingly the previously identified nature of classic IT security comes to light in this, i.e., an

obstructive state is rather avoided than handled. This can be illustrated with the help of an exemplary allocation of existing approaches to the behavioral space. Obstructive situations mainly originate from applying classic IT security to workflows. Figure 2.14 indicates the restricted behavioral space of satisfiable workflows, and the even more restricted scope of resilience (for example for $k=1$). Obstruction-free behavior corresponds to the secure behavior or is even restricted further. Hence, similarly to these indicated two areas, a big part of existing research mainly has the tendency to even further restrict the behavioral scope. Research on related notions focus on keeping the secure state, across all process entities. As for the use of logs,

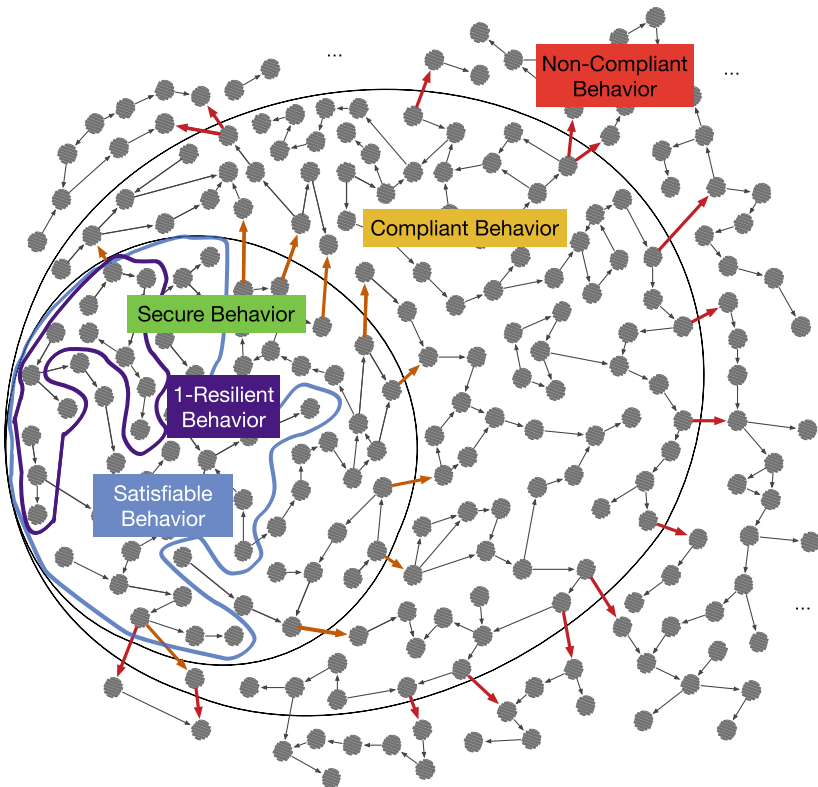


Figure 2.14 Process behavior sketching completed satisfiable executions reaching end state (no outgoing arc) and k -resilience (with $k=1$, i.e., one absent user)

predictive monitoring equally aims at avoiding unwanted outcome (e.g., avoidance of non-completing paths). All such approaches act in the frame set by classic security. Before the execution these avoidance-techniques definitely make sense because the obstructive situation is not there, and they can be used to improve process or policy design and detect design flaws. Taken together, classic WSP research operates in the frame set by IT security as well and only a few other approaches consider a security violation. Interestingly those who do so actually assign costs to policy changes or violations, which is somewhat in line with the required indicator-based approach to process security.

2.4.1 Main Deficits in Obstructability Research

Figure 2.15 shows the different entrepreneurial possibilities in avoiding or confronting and handling of obstructions. To actually handle obstructions, there is a need to extend the behavioral scope in the sense of the paradigm shift that is meant to work with an indicator-based security, i.e., to allow the handling of obstructions in the frame set by compliance (see cross-hatched area in Figure 2.15). After each sub-chapter identified requirements for preventive, runtime and detective approaches, the main deficits regarding the detection and handling of obstruction will be highlighted and summarized. This allows to put the solutions of this thesis into a more comprehensive framework.

Despite the need to automate regulation, as identified in Chapter 1, this chapter identified that there is no automated or semi-automated security-sensitive solution to obstructions. Although there are approaches with this intention, they often only exist as first concepts, and aim to change policies, or even override them without taking them into account (e.g., Break-Glass).

Although delegation does not totally override policies, it involves the risk of collusion and requires the availability of a delegator. Here, automating delegation seems promising to provide a higher degree of process automation. There are only a few approaches that allow to work with indicators. They actually consider a rather coarse grained policy violation. However, these promising cost-based approaches, which allow for policy violation, are not able to adequately capture the state of obstruction and do not consider a comprehensive process structure. Regarded approaches do not start from a somehow specified obstruction, from which it would be reconstructable how the obstruction occurred in the first place. They rather assume that there is such a situation and then search for solutions without actually fully taking the obstructive situation into account. In particular, there is no representation of the problem of obstruction that captures all inputs involved. This, however, would allow to set

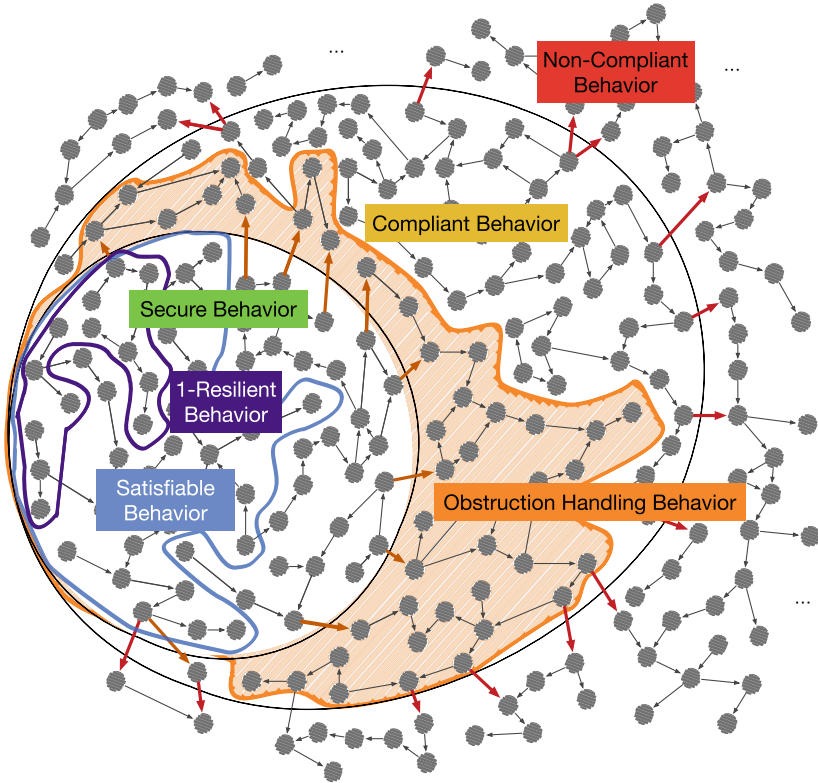


Figure 2.15 Existing WSP and resilience approaches and the potential behavioral gain in handling obstructions

the frame that allows for further steps to handle the obstructive situation. Although logs constitute a further input, literature has so far not considered their use to handle an obstructive situation. In essence, the following main deficits in the detection and handling of obstructions can be observed, namely there is

- no comprehensive representation of the problem of obstruction,
- no security-sensitive solution, and
- no use of logs.

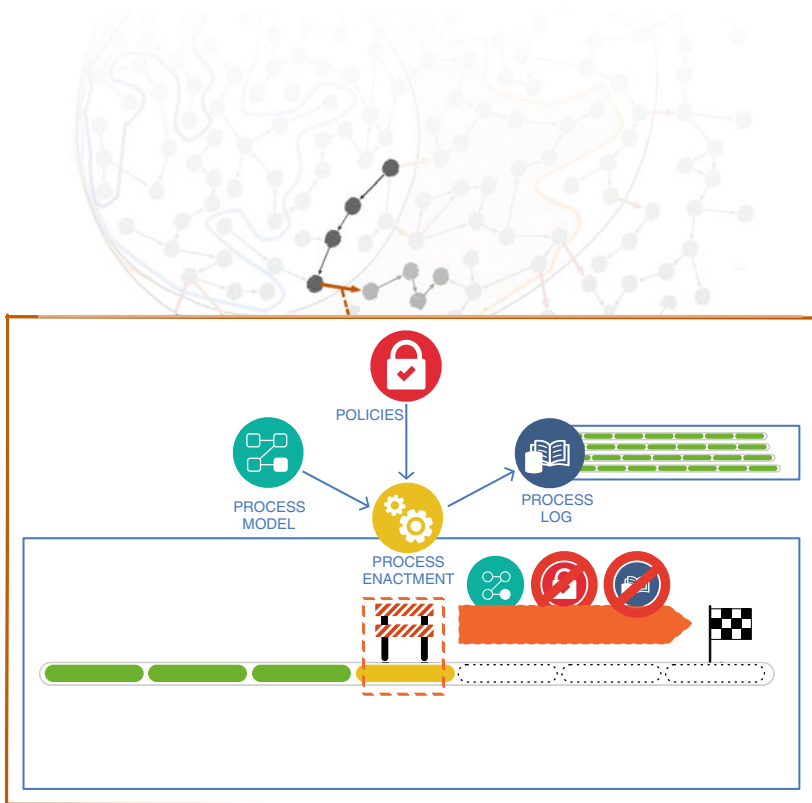


Figure 2.16 Main deficits regarding completability

2.4.2 New Spheres of Action: Expanding the Behavioral Space

How can the scope of action be widened to allow for more obstruction handling behaviour that is still compliant (see cross-hatched area)? In order to understand the general approach to obstruction handling and its requirements, the big picture, which illustrates the general effects of obstruction, is broken down to the consideration of the individual case of an obstruction again. The arrow towards the finish flag in Figure 2.16 indicates an obstruction arc. Then, the aim of the approaches of this thesis is to find a partial trace or activity sequence to complete the obstructed process execution (i.e., find the missing path to reach the finish flag). This again corresponds

to what the literal meaning of the obstruction expresses. In contrast to a deadlock, which in its literal sense has no possibility of still finding a good end, an obstruction means that something is blocked, but the goal is still in sight. Therefore the goal is to clear away or bypass what is causing the obstruction to still reach the goal that has so far been blocked. Clearly, if a process obstructs in practice due to the lack of practical solutions, process participants often choose pragmatic ways beyond the identified approaches. They figure out how to work around an obstruction, for example, by making phone calls to find out why a process is blocked, or they get their business done in a way completely unrelated to the process. This is hardly observable nor even controllable from a PAIS and involves many security risks. Therefore, a security-sensitive approach to tackle obstructions needs to set the frame that allows a PAIS to be in control and aware of the obstructive situation and address the deficits of existing approaches.

2.4.2.1 General Framework for Requirements

Based on the identified deficits and against the background of Figure 2.16, three main requirements are identified for the approaches to detect and handle obstructions:

- *Capturing the state of obstruction (GR-1)*: Obstructions can currently only tediously and not comprehensively be represented, based on separate specifications of the process model, the policy, and further constraint specifications. Hence, there is no intuitive representation that can capture the obstructed state. Given that such a representation would be graphical, it would even allow to better visualize obstructions, for example, if an SoD rule conflicts with the progression of the process. Further, it allows to better handle obstructions because it forms a more comprehensive basis to apply analysis techniques, which make it possible to find a solution.
- *Detecting and solving obstructions in a security-sensitive way based on indicators (GR-2)*: Based on the identified deficits of existing approaches, the “tackling” of an obstructed state has to be done in a more security-sensitive way. Along all three phases of execution, the need to foster an indicator-based security can be underlined. Here, a cost-based approach constitutes a frame how to consider and capture indicators. Given that the least cost represents the highest degree of security-sensitivity, the minimization of these costs then represents an optimization problem.
- *Considering all inputs (model, policy, log) (throughout all approaches) (GR-3)*: Based on the identified phases of process execution, this thesis strives for a holistic approach, and is meant to tackle model- as well as log-based situations. The problem shall be solved by taking all relevant inputs into account and deduct

an optimal solution based on indicators, thereby taking the least risk or violation into account. Logs are not used for obstruction analysis and handling, and are useful for the indicator based view on security. Such a solution helps to better implement regulation because it allows to better consider risks as well. Here, one can imagine the help of ex ante as well as ex post approaches to steer the execution towards completion. Hence, the notion of obstructability of this work aims to handle obstructions at runtime in a way that allows to benefit from the approaches of the other phases as well. By addressing these requirements, the risk of damage by blocked process executions as well as violation of security policy shall be minimized.

In conclusion, as depicted in Figure 2.15, the goal of this thesis is to develop an approach that can detect obstructions and resolve them policy-wisely. The general approach is located between security and business goals, compliance and indicators. In the following chapters, the identified deficits will be addressed with a holistic approach that takes the design, execution and audit phase into account. Figure 2.17 illustrates problem setting and places the contributions of this work into the identified gaps. The SecANet approach (Chapter 3) addresses the lack of capturing the state of obstruction and lays the foundation for an indicator-based solution. In order to handle obstructed executions, a hybrid approach will be presented, depending on

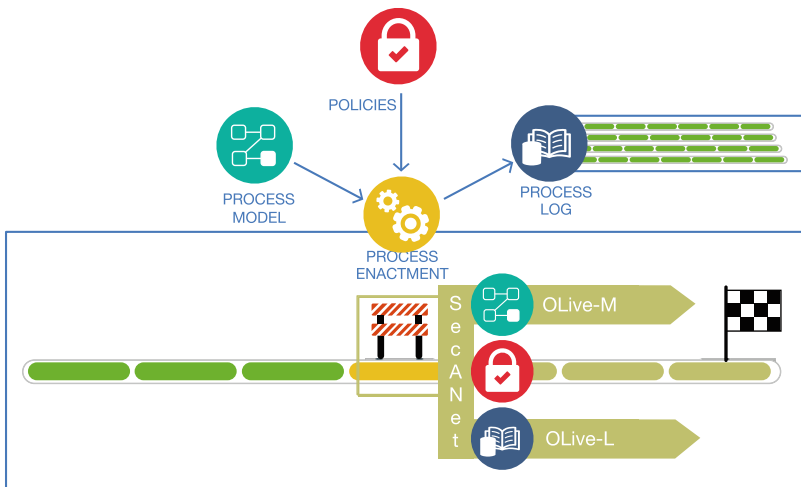


Figure 2.17 Contribution of work to analyze, detect and resolve obstructions

the existence of historical information. Both will be able to consider indicators such that obstructive situations will become resolvable in a security-sensitive way. The OLive-M approach (Chapter 4) will show how, based on the SecANet, model-based solutions to obstructions can be computed by using light analysis methods. The OLive-L approach (Chapter 5) foremost uses the log to find solutions to handle an obstruction. That way, this work represents a holistic approach that takes models and logs into account and is applicable at design-, run- and audit-time. Beyond the general requirements (GR) identified in this section, these approaches will address the specific requirements that apply to their execution phase (i.e., ROA, RSC, RLC).

Altogether, by analysis, detecting and handling obstructions, the goal is to improve security in business processes. Conflicts caused by security policies are supposed to be captured and resolved in a security-sensitive way and the processes are allowed to complete policy-wise, and still meet compliance. The overreaching goal is to relax the tension between too strict security controls and business goals of processes in the practical setting. By providing mechanisms that predict (or at least detect) obstructions and, based on the existing process controls and data, propose workarounds that, albeit not fully policy-compliant, allow enactment with the nearest security-sensitive match feasible, it will become possible to engineer enterprise systems with a large extent of flexibility and security.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





Obstruction Modeling

3

The previous chapters identified the need to capture the state of obstruction as well as the fact that the inputs whose interplay may finally result in such a state are the workflow and its related policies. Although the approaches observed in Chapter 2 propose different ways to connect such authorizations and constraints with the workflow, they lack in making the obstruction state tangible in an intuitive way. Currently, it can only tediously be represented based on different inhomogeneous and limited definitions and formalizations. These approaches neither form a basis to conduct obstructability analysis nor find an adequate solution to escape an obstruction.

Based on the general framework of requirements in Chapter 2, this chapter proposes an approach that aims to address the deficits and requirements for obstructability analysis (ROA) stipulated in the previous chapter (cf. Section 2.1). For this, *one* representation that allows grasping the problem of obstruction (cf. GR-1) and facilitating its analysis (cf. GR-2) is supposed to encode the business goals and compliance requirements. In particular, such a representation should allow an adequate capturing of all inputs (cf. GR-3) of a security-aware process specification, namely the process model and the policy. On the one hand, for an obstructed case, the actual state of obstruction is supposed to be capturable in such a representation. On the other hand, it is meant to provide obstructability analysis, namely to detect obstructions in a security-aware process specification and to capture the results as well. The state of obstruction captured in the representation will then provide to search for ways how to escape from that problem during process enactment but also to be used to identify weak-spots of the policy design. A central question is how to capture the different inputs of a security-aware process expressively and, at the same time, preserve their behavior.

Because obstructions occur during the process execution, the guiding principle of this approach is to represent the obstruction state in a process-oriented way as well. Chapter 2 already introduced the manifold ways to model processes, which

usually focus on the control flow as well as policy models (e.g., BPMN, EPC, Petri nets, RBAC, ACL). The process is, therefore, primarily anchored in the control flow. It builds the point of reference for process-oriented modeling. It can further be observed that the authorization and SoD or BoD rules manifest themselves in the activity sequences (or traces) of the process in a PAIS as well, denoting, for example, who executed which task. In this view, the overall policy can be understood in a process-oriented way too. More specifically, authorization can be regarded as a process in itself, i.e., the process of granting user access to a specific task. Further, also SoD or BoD constraints can be expressed in a process-oriented way. To follow the binding or separation of granted accesses, they reduce the options to choose who is supposed to execute which task. More precisely, these possibilities are reduced depending on the tasks already performed. Hence, in order to realize a process-oriented representation, a formalism will first be identified that is able to adequately capture the control flow with respect to the identified requirements. In the next step, the policy will be “flattened” (i.e., incorporated) into the process model by using the same way of modeling as identified for the control flow. It will then be necessary to analyze if the behavior of the inputs as well as the overall system behavior is not changed.

This chapter will first explore the possibilities for the modeling of the process, which can primarily be used for the modeling of the control flow, and relate to comparable approaches beyond the WSP and resilience context. In particular, based on the identified requirements, the modeling method should allow both a comprehensive process structure (ROA-2) and an efficient analysis (ROA-3). Based on the process model representation that is considered reasonable, the policy will also be understood as process-oriented, such that it is integrable into the existing workflow. The policy will consider basic constraints (ROA-4) as well as the option for pre-assignments (ROA-1). That way, a security-aware representation that not only represents the functional and behavioral but also the organizational aspect of a security-aware process specification will emerge, named SecANet. A SecANet will also allow considering costs (ROA-6), which can be manually defined or which consider insights from the process specification or the log as well. Finally, the approach will support capturing obstructed states (ROA-7) as well as synthesizing partial plans (ROA-5) that cause obstructions. In all of this, the general intuition and feasibility behind the flattening in the SecANet approach will crystallize. The important system properties of the resulting representation will be highlighted. A SecANet decomposition will then allow reasoning to what extent the approach maintains the integrity of inputs as well as the integrity of the overall process. By introducing the so-called re-enactment, cyclic security-aware workflows can be considered by the SecANet encoding as well. Section 3.2.6 will subsume SecANet-based

satisfiability and obstructability checks as SecANet soundness and show further extensions to facilitate analysis. An experimental evaluation will compare soundness checking runtimes with typical WSP-related approaches. In the course of the discussion, the computational complexity of the SecANet encoding will be examined. With the introduction of SecANet+, the SecANet approach will be generalized further. It will lay the basis to integrate additional inputs that model further behavior that affects the process. To show the extensibility of the SecANet+ approach, the integration of additional inputs that further constrain the execution of tasks, e.g., counting constraints but also resilience, will be indicated. Finally, the development of the SecANet formalism and its use as a target metamodel of security-aware process specifications creates a basis for further analysis. It will build the foundation for the model- and the log-based approach presented in Chapters 4 and 5, respectively.

3.1 Ways to Model Processes

Process models usually contain at least the control flow of a process (i.e., the entirety of valid activity sequences). Depending on the metamodel used, they may also include other aspects (data use and flow, actors, etc.). Although each metamodel has various abilities and characteristics, whose suitability depends on its application scenario, it is often possible that a formalism easily translates into other notations and formalisms [6]. UML [116, 117], BPEL [123], BPMN [118], EPCs, YAWL [105], Petri nets and Transition Systems are the most well-known and studied metamodels. Although the question about the most commonly used metamodel is hard to answer, a recent BPM survey at least states that “the OMG’s modeling notation, BPMN, has become a popular, worldwide standard” [104]. To some extent, the effort of this work in capturing the policy into a single process-aware representation is comparable to approaches that provide modeling support for process-related access control models (Strembeck et. al [191], for example, list further literature for related UML, RBAC and BPMN extensions). However, the focus of this chapter is not to support such a modeling in an established metamodel. Based on a specification given in some metamodel, this chapter aims to deduce a representation that focuses on the analysis and handling of obstructability and that meets the stipulated requirements.

Notwithstanding the ideas or methods pursued or developed by the numerous approaches mentioned in Chapter 2, they all show that the prerequisite for their application is the availability of a formal model. For example, this is the case for the satisfiability analysis in which often partial orders describe the flow of process tasks, but also conformance checking. Regarding the latter, for instance, the differentiation between conforming and non-conforming process behavior demands a formal model

of the control flow of the process. Up to now, the use of a formal model for the desired representation has been a rather implicit requirement. Against the background of the various metamodels already mentioned, which are often motivated by practice, it is necessary to make this requirement explicit. Such a formal model is supposed to allow the modeling of realistic processes. As identified in Chapter 2, only a few of the WSP-related approaches support comprehensive structural possibilities of processes. Because process mining aims to look at realistic (lived) processes as well, it is worthwhile to observe the metamodels used in process discovery. While, in the beginning, literature mainly based on directed graphs and state machines, there is a distinct tendency towards Petri nets. Publications from other areas of Process Mining support this trend as well [189]. Petri nets, in comparison to state-based metamodels (e.g., transition systems), are more suitable to depict concurrent behavior and causal relations between single activities within a process [55]. They are not purely textual and formal representations but also allow a graph-wise illustration. Their graphical readability is comparable to BPMN models. Petri nets provide a formal semantic and unique definitions for their structure and behavior. Different Petri net dialects allow various possibilities for analysis [2, 4, 86]. In particular, research findings on Petri net properties can be applied to business processes, such that boundedness, liveness, or reachability, for example, determine process completion. To assess the behavior of business processes, Petri net behavior can be examined by observing the language of the net, which encompasses all possible execution sequences. Because a crucial requirement of the approach of this chapter is to not only preserve the behavior of the process specification but also to consider its applicability for further analysis, it is essential to consider the influence of the chosen Petri net dialect on the existence of (efficient) analysis techniques and observability of behavior-related (in the sense of language-related) properties. Regarding the analysis of Petri nets, especially for the class of ordinary so-called Place-Transition nets (P/T nets), there is a plethora of efficient techniques [186]. In contrast, the analysis of the class of Colored Petri nets, as a typical example of high-level Petri nets, is more complex [77]. There are high-level Petri net representations that, in some way, already allow modeling security requirements. For example, there are works on the definition of security properties on Petri nets [12, 111–113] as well as approaches for the verification of access and usage control guidelines [19, 121, 127, 139], integrity conditions [218] and requirements of information flow control [10, 11], or conflict of interest detection [217]. In contrast to ordinary nets, apart from the higher computational complexity, high-level nets do not directly reveal their behavior in their language, which limits the possibilities to reason on the required language preservation.

In essence, Petri nets are graphical, well-known, and widely used. They are the oldest and best-investigated process modeling language allowing for the modeling

of concurrency in a mathematical representation. More specifically, it is the graphical nature of Petri nets, their explicit representation of the notion of state, and the existence of analysis techniques that made them eminently suitable for workflow specification [105]. In particular, their distinction between states and events (which manifest in so-called firing sequences) seems promising to depict the state of obstruction. Moreover, their graphical representation also offers the potential to visualize the security-aware process specification more intuitively. Because Petri nets appear to be suitable in many respects and meet the set requirements, they are the method of choice to model the envisaged representation. The rationale in all of this is that, on the basis of such Petri net-based representation and the plethora of analysis techniques, a way to resolve obstructions can be found as well. As far as known, the identified potential of Petri net techniques has not been applied for obstructability analysis so far. Choosing the Petri net formalism does not mean a loss of generality because it translates to other formalisms. For example, regarding formalisms often used in WSP-related approaches, it is known that a family of partial orders is needed to characterize a single Petri net [126]. Thereby, modeling the control flow with Petri nets has the advantage of compactly representing a workflow specified as potentially many partial orders (cf. Section 2.1). Further, as a rare example of WSP approaches that allow for comprehensive process structure, Basin et al. use Hoarse Process Algebra CSP. In this regard, it is always possible to transform a CSP process [213] into a Petri net (more specifically, a so-called safe Petri net). Moreover, while a transformation of informal models into formalized models is not trivial, at least for the regarded basic modeling constructs, there are transformation techniques, e.g., to transform BPMN models [79] into Petri nets¹. Although Petri nets provide an intuitive and simple graphical representation, they determine a precise execution semantic.

Thus, the approach of this chapter will use Petri nets as its formal model in such a way that it is able to integrate a security-aware process specification into a single net that allows analyzing the obstructability and capturing obstructions, a so-called SecANet. Because the aim of this work is to allow for efficient computation as well as straightforward encodings and behavioral observation, the idea is to stay with ordinary nets because, as explained above, for other Petri net classes, e.g., colored Petri net, the reasoning gets harder. Therefore, all inputs will be flattened into a single P/T net. Thus, it is possible to keep the net constructions as simple as possible. Because such a P/T net can directly encode the control flow of the

¹ For more complex BPMN elements, there are some issues in their mapping towards a formal execution semantics such as Petri nets or other modeling languages such as the Business Process Execution Language (BPEL) [170].

process, the essential question will be how to map security policies into the net. In this regard, Chapter 2 only formalizes authorization as a mapping of the set of users to the respective set of workflow tasks so far. Moreover, it captures SoD and BoD constraints with the help of different binary relations from the set of users on the respective tasks, for example, entailment constraints. Thus, also straightforward user-task assignments that can be represented by ACLs will be chosen, instead of the more complex representation of RBAC (which, in turn, may also be decomposed to a direct user-task assignment).

The following section is going to formally describe Petri nets, in particular, ordinary Petri nets (or P/T nets), and related concepts. P/T nets build the foundation for the definition of different net properties and net subclasses such as workflow nets (WF-nets), which allow modeling workflows with a precise task execution semantic. The properties of Petri nets such as boundedness, safeness, liveness or deadlock-freeness, will be of interest because they affect, for example, the computational complexity, the interpretability of the state of the net as well as language-related reasoning. After the definitions related to Petri nets, users, user-task authorization as well as SoD and BoD constraints will be defined. That way, the basis will be laid to formally grasp obstructability in workflows.

3.1.1 Petri Nets: The Method

Petri nets represent a graphical and mathematical modeling tool that can be applied to many systems. As described above, they are very well suited for the description and investigation of information processing systems that are characterized as concurrent, asynchronous, parallel, or non-deterministic. Their graphical nature allows for visual communication similar to flowcharts, block diagrams, and networks. Additionally, within these nets, tokens are used to simulate the dynamic and simultaneous activities of systems. That way, a Petri net can encode certain properties of the system it models. There are structural and behavioral properties that relate to the net structure, or the token-related dynamic behavior, respectively. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models for system behavior. Petri nets are therefore suitable for both practical and theoretical use [155]. The following definitions related to Petri nets and to properties that will be of relevance in this thesis will mainly base on the works of Murata et al. [155], Desel, Esparza (e.g., free-choice net [73]), and van der Aalst et al. (e.g., WF-nets [3]).

3.1.1.1 Petri Net

A Petri net is a bipartite graph consisting of places graphically represented as a circle and transitions depicted usually as squares, rectangles, or vertical lines. Only two distinct nodes can be related to each other via directed arcs or edges (i.e., an arc connects a place with a transition or vice versa). The resulting net structure is static, but tokens controlled by the firing rule can flow through the network. The distribution of the tokens over the places determine the state of a Petri net and represents its marking [3].

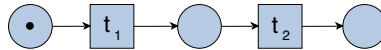


Figure 3.1 Petri net of the Determine Market Value (DMV) process

Figure 3.1 shows an example of a Petri net that models the process of two sequentially ordered tasks t_1 and t_2 wherein the leftmost place contains one token². The written formal notation of this example will be given after the definition for Petri nets.

Definition 3.1 (Petri Net and Marking). A Petri net is a 4-tuple $N = \langle P, T, \mathcal{F}, m_0 \rangle$, where P is the set of places, T is the set of transitions, satisfying $P \cap T = \emptyset$ and $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation, and m_0 is the initial marking.

A marking (state) is an assignment of a non-negative integer to each place. If a non-negative integer k is assigned to place p by a marking, it can be said that p is marked with k tokens. For this, a marking is denoted as a $|P|$ -vector $\mathbf{m} \in \mathbb{N}^{|P|}$, where the component p of the vector is a natural number. Then, the p -th component of \mathbf{m} , denoted by $m(p)$, is the number of tokens in place p . To allow for better readability, markings can also be denoted as sets or multi-sets. Because a linear order is assumed for the set of places $P = \{p_1, \dots, p_{|P|}\}$, the characteristic vector (or indicator/incidence vector) can be used to link the vector notation with the (multi-)set notation. For a set $m \subseteq P$ (or a multi-set or bag $m \in \mathcal{B}(P)$ over P), the characteristic vector of m with respect to P can be denoted as $\chi(m) = \langle x_1, \dots, x_{|P|} \rangle$, where $x_i = 1$

² For the sake of clarity, the added tokens, the three places p_1 , p_2 and p_3 (from left to right), are not labeled with their names here.

if and only if (shortened as iff) $p_i \in m$ (or x_i is the multiplicity of p_i in the multi-set m) and 0 otherwise, for $1 \leq i \leq |P|$. In this work, markings will explicitly be written as (place) vectors \mathbf{m} if required. Normally, however, the notation used is evident from the given specific context.

Based on Definition 3.1, the net P/T net N depicted in Figure 3.1 can be denoted as $P = \{p_1, p_2, p_3\}$, $T = \{t_1, t_2\}$, $F = \{\langle p_1, t_1 \rangle, \langle t_1, p_2 \rangle, \langle p_2, t_2 \rangle, \langle t_2, p_3 \rangle\}$ and the marking $m_0 = \langle 1, 0, 0 \rangle$. The visualized marking defines that a single token (small solid black circle) is assigned to place p_0 . This is the standard visualization of markings that is equivalent to their formal definition. To illustrate the different marking notations, suppose p_1 is marked with 1, p_3 with 3 and all other places with 0 tokens. This marking m can be denoted as $m(p_1) = 1$, $m(p_2) = 0$ and $m(p_3) = 3$, or as the multi-set $\{p_1, p_3^3\}$, or as the characteristic vector $\chi(\{p_1, p_3^3\})$ that, in turn, encodes the vector notation of the marking $\langle 1, 0, 3 \rangle$.

The set of places and transitions build the nodes of a net. The definition of flow relations above does not allow so-called arc weights > 1 . A Petri net is said to be ordinary (or plain) if all of its arc weights are 1's, i.e., an arc only allows to produce or consume exactly 1 token [155]. As mentioned before, all Petri nets considered in this thesis are ordinary P/T nets. The set-based definition of the flow-relation further implies that there may not be multiple arcs between the same pair of nodes (which prevents the possibility to model arc weights > 1 with multiple arcs). Based on the given net structure, paths between the nodes are connected by a sequence of arcs.

Definition 3.2 (Paths and Cycles). *A path is a sequence of nodes u_1, \dots, u_r such that $\forall i, 1 \leq i < r : F(u_i, u_{i+1}) > 0$, i.e., u_{i+1} is an output of node u_i . A path is called simple if no node appears more than once on it. A simple path is called a cycle if all nodes along the path differ, except for the initial and the final node.*

Definition 3.3 (Strong Connectedness). *A Petri net is strongly connected iff, for every pair of nodes (i.e., places and transitions) x and y , there is a path leading from x to y .*

A further structural characterization related to paths is given by the notion of so-called PT- or TP-handles, whose existence indicates a rather “bad” structure.

Definition 3.4 (PT-handle and TP-handle [90]). *A place-transition pair $(p, t) \in P \times T$ is called a PT-handle iff there are two simple paths from p to t sharing only the two nodes p and t ; a transition-place pair $(t, p) \in T \times P$ is called a TP-handle iff there are two simple paths from t to p sharing only nodes p and t .*

Hence, Figure 3.1 is not strongly connected because there is for example no directed path for the pair (p_3, p_1) that leads from p_3 to p_1 . However, it does not contain PT- nor TP-handles. The notion of pre- and post-set is essential to describe net properties, in particular, regarding the net structure.

Definition 3.5 (Pre-set, Post-set). *Given a node $x \in P \cup T$, its pre-set $\{y | \langle y, x \rangle \in \mathcal{F}\}$ and post-set $\{y | \langle x, y \rangle \in \mathcal{F}\}$ are denoted by $\bullet x$ and x^\bullet , respectively.*

For instance, in Figure 3.1 the pre- and post-sets of $t - 1$ and p_1 can be denoted as $\bullet t_1 = \{p_1\}, t_1^\bullet = \{p_2\}$, $\bullet p_1 = \emptyset$ and $p_1^\bullet = \{t_1\}$.

So far, the definitions have mainly focused on the properties given by the basic net structure. As indicated before, besides the net structure, a Petri net model of a dynamic system also consists of a marking. The system dynamics or behavior is given by the evolution rules for the marking [186].

The following definitions are token-related and therefore allow the consideration of properties related to the behavior of the net. The distribution of tokens results from a marking (cf. Definition 3.1). A transition can “fire” when all its input places are marked. This Petri net terminology of “firing” a transition represents the execution of a task (which, in turn, stands for the execution of an activity). Firing a transition generates a new marking. Thereby, each input place of the transition loses one token, and each output place of the transition receives one token. Thus, the execution semantics of a Petri net can be grasped as a token flow game that follows this firing rule. That way, the token flow determines the actual system behavior, which is why, as soon as Petri nets are marked, literature often uses the term “system” in different variations.

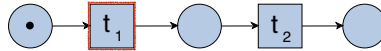


Figure 3.2 DMV Petri net with initial marking

Definition 3.6 (Enabledness and Firing Rule). A transition t is enabled in a marking m when all places in $\bullet t$ are marked, which can be denoted by $(N, m)[t]$ iff $\mathbf{m} \geq \chi(\bullet t)$ (vector notation) or $m \geq \bullet t$ (multi-set notation), respectively³. When a transition t is enabled, it can fire by removing (or consuming) a token from each place in $\bullet t$ and putting (or producing) a token to each place in $t \bullet$.

For instance, based on the given marking in Figure 3.2, transition t_1 is enabled (highlighted with red) because all places in $\bullet t_1$ are marked. Firing t_1 then consumes a token from each place in $\bullet t_1$ and produces a token in each place $t \bullet$, which results in the marking $m_1 = \langle 0, 1, 0 \rangle$ depicted in Figure 3.3.

A transition without any input place is called a “source transition”, and one without any output place is called a “sink transition”. Based on Definition 3.6, a source transition is unconditionally enabled, and the firing of a sink transition consumes tokens, but does not produce any. A pair of a place p and a transition t is called a self-loop if p is both an input and output place of t (e.g, to model a “do-while-loop”). A Petri net is said to be pure if it has no self-loops. Based on the firing rule the set of reachable markings of the Petri net, also called its state space, can be determined.

³ Because multi-sets are essentially vectors with natural numbers as components (cf. Definition 3.1), they can be used for calculation component by component and to compare them in the same way as it is usual for Petri net vector markings.

Definition 3.7 (Reachability and Feasible Sequences). Given a Petri net N , a marking m' is reachable from m if there is a sequence of firings $t_1 t_2 \dots t_n$ that transforms m into m' , denoted by $m[t_1 t_2 \dots t_n]m'$ or $m' \in R(N, m)$. A sequence of transitions $t_1 t_2 \dots t_n$ is a feasible sequence if it is fireable from m_0 . The set of reachable markings from m_0 is denoted by $[m_0]$.

For instance, based on the marking $m_0 = \langle 1, 0, 0 \rangle$ depicted in the Petri net of Figure 3.1, the marking $m_1 = \langle 0, 1, 0 \rangle$ shown in Figure 3.3 is reachable through the feasible sequence t_1 , while, for example, the marking $m' = \langle 0, 1, 1 \rangle$ is not reachable. Hence, the reachability of the marking in Figure 3.3 can be denoted as $m_0[t_1]m_1$.

The set of reachable markings represents the set of states of the system. The transitions between the markings represent transitions between the states. Hence, to capture the behavior of a Petri net, a transition system can be obtained. It contains a set of possible states and a set of transitions that stand for the potential changes of the systems state. That way, transition systems represent automaton that describe the behavior of a system of processes [18]. After the succeeding definition of transition systems, the reachability graph (or marking graph) can be defined.

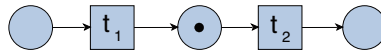


Figure 3.3 DMV Petri net with new marking after firing t_1

Definition 3.8 (Transition System). A Transition System (TS) is a 4-tuple (S, E, A, s_{in}) where S is a set of states, E is a set of events where $S \cap E = \emptyset$, $A \subseteq S \times E \times S$ is the set of arcs (or transitions), which connect states, and $s_{in} \in S$ is the initial state. The transitions are denoted by (s, e, s') or $s \xrightarrow{e} s'$. $S_0 := \{s \in S \mid \nexists (s, e, s') \in A\}$ denotes sinks within the TS, i.e., states without outgoing arcs.

A TS is called finite when S and E are finite. This thesis only deals with finite transition systems. A state s' is reachable from a state s if there is a sequence of

events $\sigma = \langle (s, e_1, s_1) \dots (s_k, e_k, s') \rangle$. Reachability can be denoted with $s \xrightarrow{*} s'$ or, specifically, $s \xrightarrow{\sigma} s'$. Based on this, the reachability graph can be defined as follows.

Definition 3.9 (Reachability Graph). *For a Petri net $N = (P, T, F, m_0)$, the reachability graph $RG(N)$ represents a TS whose states correspond to the set of reachable markings $[m_0)$ and whose events correspond to transitions. There is an arc (m, t, m') iff $m \xrightarrow{t} m'$. m_0 is the initial state.*

The boundedness of a Petri net means that, within all reachable markings, there is an upper bound for the number of tokens contained in places. Thus, it is a consequence of the finiteness of its reachability graph. A 1-bounded Petri net is called “safe”.

Definition 3.10 (Boundedness and Safeness). *A Petri net is bounded iff for every reachable state and every place p the number of tokens in p is bounded, i.e., there is an integer k such that the number of tokens in any place cannot exceed k . More specifically, a Petri net is k -bounded if no marking in $[m_0)$ assigns more than k tokens to any place of the net. A net is bounded if it is k -bounded for some k such that no place holds more than k -tokens, i.e., $\forall p \in P$ and $\forall m \in [N, m_0) : m[p] < k$. A Petri net is safe if the numbers of tokens in each place cannot exceed one, i.e., it is 1-bounded.*

Although the terms of liveness and safety that are introduced in this section are reminiscent of the two classes of security properties considered in Chapter 2, it is important to note that safety and liveness security properties are not equivalent to their counterparts in Petri net theory. However, they can be used to describe somewhat similar properties [132].

Definition 3.11 (Liveness). *A PN is live iff every transition can be infinitely enabled through some feasible sequence of firings from any marking in $[m_0)$. Hence, for no matter which marking has been reached from m_0 , then for each transition t of the model, there must be a sequence of firing such that the resulted marking enables t , i.e., $\forall m \in [N, m_0), \exists m' \in [N, m)$ such that $(N, m)[t]$. Further, a transition is “simply live” if it can fire at least once. A Petri net is “simply live” if all of its transitions are simply live.*

Historically, the concept of a safe or n-safe (or n-bound) Petri net as well as liveness and related variants have been in use since the 1970s. At the beginning of the 1980s, temporal logic then distinguished between safety properties and liveness properties (i.e., “something bad” or “something good” happens, respectively). As elaborated in Chapter 2, a sensible description of the requirements of a system always includes properties of both types [172]. This analogy also applies to Petri nets because the properties of liveness and n-safeness can be subsumed under the notion of well-formedness.

Definition 3.12 (Well-Formedness). *A Petri net N is well-formed iff there is a state m for the net such that (N, m) is live and bounded.*

Definition 3.13 (Deadlock-Freeness). *A Petri net N is said to be deadlock-free if at least one transition is enabled at each marking m derived from the initial marking m_0 .*

Definition 3.14 (Reversibility). *A net is reversible if the initial marking m_0 is reachable from every marking of $[m_0)$. This means that the system can get initial settings after it is executed.*

It should be noted that liveness, boundedness, and reversibility are independent of each other. Because the notion of obstruction can be compared to a deadlock situation, the notion of siphons and traps will be introduced.

Definition 3.15 (Siphon). *A siphon (or also named deadlock) is a set of places such that every transition that outputs to one of the places in the siphon also inputs from one of these places. Formally, a non-empty subset of places D of a net is a siphon if ${}^{\bullet}D \subseteq D^{\bullet}$. This means that if a siphon is blank (i.e., does not contain any tokens), it will remain blank for every possible firing sequence. Hence, if a marking $m \in [N, m_0)$ is in a deadlock state, then $\forall d \in D : m[d] = 0$, i.e., D is an unmarked set of places. In this case, no*

transition can place a token in the siphon because there is no token in the siphon to enable a transition that outputs to a place in the siphon.

Because every transition that has an input place in a “blank” siphon is in a deadlock and will have no chance of firing, a siphon is “bad” for liveness. Such a deadlock, in the sense of a Petri Net, is a deadlock in the usual sense only if it is blank. Therefore, literature also refers to the wording “potential deadlock” for the siphons defined above [161].

Definition 3.16 (Trap). *A trap is a set of places such that every transition that inputs from one of these places also outputs to one of these places. Formally, a non-empty subset of places D of a net is a trap if $D^\bullet \subseteq \bullet D$. Hence, once a trap is marked (i.e., does contain at least one token), it will always be marked, no matter what firing sequences take place. Once a token is in any of the places of a trap, there will always be a token in one of the places of the trap. Hence, if marking $m \in [N, m_0]$ is in a trap state then $\exists d \in D : m[d] = 1$, i.e., D contains at least one marked place. In other words, the firing of transitions may move the token between places but cannot remove all tokens from a trap.*

For instance, the net in Figure 3.1 contains the siphon $D = \{P_0\}$ because $\bullet D = \emptyset$ and $D^\bullet = \{t_1\}$, i.e., $\bullet D \subseteq D^\bullet$ and the trap $D = \{P_2\}$ because $D^\bullet = \emptyset$ and $\bullet D = \{t_2\}$, i.e., $D^\bullet \subseteq \bullet D$. A more detailed explanation will be given in the course of the example SecANet in Section 3.2.

Traps and deadlocks are not exclusive. For instance, every strongly connected Petri Net is both a trap and a siphon. An essential connection between traps and siphons is that if a deadlock contains a marked trap, it will never become blank, such that this siphon is no threat to liveness anymore. This property is also known as the siphon/trap- or trap/co-trap-property [51].

3.1.1.2 Basic Modeling Examples

Several fundamental situations may occur in the dynamic behavior of Petri net systems. In particular, a strong feature of Petri net models is that notions concerning concurrent systems can be formulated in a very natural way. Based on the previous definitions, basic modeling possibilities that illustrate how a Petri net can be used

and composed to model the required sequential, parallel, conditional (or exclusive) and looping behavior (cf. ROA-2) will be provided. For this, three fundamental relationships that may hold between the occurrence of two events, i.e., the firing of transitions t_1 and t_2 (which may model tasks, or activities), will be regarded: causality, concurrency, and choice [155].

Causality: Causality means that the occurrence of an event conditions the occurrence of another one. This results in a sequential order. Figure 3.1 illustrates an example of such a causal relationship. Here, t_1 needs to occur to enable t_2 , or, in other words, firing t_1 is the cause that enables t_2 .

Besides modeling causal relationships, the possibilities of a Petri net modeling concurrent systems are distinctively interesting for parallelism and non-determinism. The non-deterministic and nonsimultaneous firing of transitions is reflected in the following two ways [161]:

Concurrency: When two enabled transitions do not affect one another in any way (i.e., they are causally independent), and a transition may fire before or after or in parallel with the other, it is called concurrency. There is no need to synchronize the events unless it is required by the underlying system that is being modeled. When synchronization is needed, it is easy to model this as well. The Petri net shown in Figure 3.4 [155] illustrates how this can be expressed. The parallel or concurrent activities are represented by transitions t_1 and t_2 . They are both enabled after the firing of the transition *Parallel Begin*. After the firing of both parallel transitions, the transition *Parallel End* can fire. Moreover, the path from *Parallel End* to *Parallel Begin* models a loop.

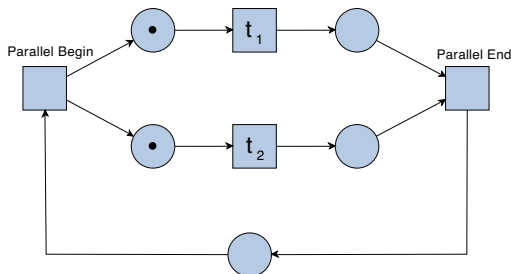


Figure 3.4 Parallel activities in a Petri net

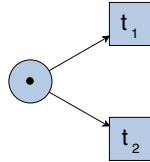


Figure 3.5 Conflict, choice, or decision Petri net structure

Choice: The other situation, in which simultaneousness is more difficult to tackle, can be handled by defining events that occur non-simultaneously. This structure of the place p , having two (or more) output transitions t_1 and t_2 as shown in Figure 3.5, is referred to as a conflict, decision, or choice, depending on its applications. Both activities are enabled but only one of them can be executed. In other words, only one transition can fire because, by firing, it removes the token in the shared input and disables the other transition. Hence, a choice relationship can be used to model decision nodes for exclusive paths or conditional gateways in a process model. It is a structure exhibiting non-determinism [155], in a sense that the selection which single transition to fire out of a set of enabled transitions may be determined in the modeled system, but not in the model simply because the model does not contain the complete information about the system. For instance, in the DMV example, the decision if the computation of the market value is correct cannot be solely assessed by the information provided by the model. In practice, at least the expertise of the case officer and further context information on the collateral under consideration and its computed market value would be required.

In summary, two events are causal if t_1 causes t_2 , they are in conflict if either t_1 or t_2 can occur but not both, and they are concurrent if both events can occur in any order without conflicts. These examples illustrate that Petri nets can comprehensively model the required structures and that they constitute a basic construction kit for Petri net modeling. In the combination of these elements, further, but still rather basic, situations can be observed. When conflict and concurrency are mixed, it is called confusion. There are two types of confusion. Figure 3.6 shows a symmetric confusion because two events t_1 and t_2 are concurrent while each of t_1 and t_2 is in conflict with event t_3 . Figure 3.7 shows an asymmetric confusion, where t_1 is concurrent with t_2 but will be in conflict with t_3 if t_2 fires first. These elementary relationships between activities will be important in the subsequent identification of Petri net subclasses as well because they provide a means to assess their modeling capabilities.

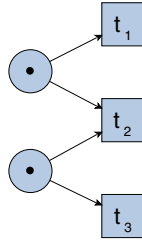


Figure 3.6 Symmetric confusion

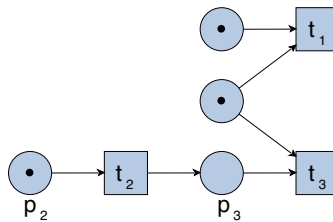


Figure 3.7 Asymmetric confusion

3.1.2 Petri Net Subclasses

The introduction above on ordinary Petri nets (i.e., P/T nets) allows a more thorough elaboration of the mentioned aspects regarding the suitability of high-level Petri nets. The underlying question is whether ordinary nets are sufficient to cover all relevant inputs or whether extended nets may address the requirements of the envisaged representation in a better way.

To start with, the simpler the structure of a Petri net is, the simpler the formal description of the behavior of individual transitions and also the corresponding analysis procedures become. On the other hand, simplifying the net also lowers the level of detail of the model and counteracts the modeling efficiency and the convenience of modifications and the extension of Petri nets. Regarding the latter, besides timed Petri nets [168] or stochastic nets [151], the introduction of individual objects as tokens particularly increases the descriptive power of nets and allows for small but efficient as well as practical modeling of systems. Such aforementioned high-level nets subsume predicate/transition nets [95], colored Petri nets (CPNs)[120], or nets with individual tokens [171] (e.g., relation nets) [155]. That way, Petri nets can be extended, for example, with resources, data, users, or also time-related information,

which results in different Petri nets dialects. Besides the fact that different token colors can be used to represent data-aspects, e.g., modeling data items or the information flow [190], they also allow modeling access control policies by modeling users [139, 191], which is particularly interesting for the intended modeling. However, such data-oriented nets not only require more complex analysis techniques [77] but also complicate behavioral observation. When, for example, token colors represent users, the transition execution sequences may not directly reveal decisions related to user accesses. In this case, the behavior of the system is not explicitly expressed in the language of the net. Although methods to transform CPNs to ordinary nets can overcome these issues, such unfoldings represent a separate step that would have to precede the desired representation [77]. Folding into (or unfolding from) a CPN, however, requires knowledge of the still unknown desired representation. Instead, the language should reflect user accesses for now, which means that transitions need to be used not only for task or activity modeling but for policy modeling as well.

Hence, in contrast to extended Petri nets, using P/T nets for the intended modeling implies that both the data reasoning stays gentle and policy-related behavioral properties are reflected in the language of the net as well. Nevertheless, in the wake of the requirement to allow for efficient solutions, it is not necessarily sufficient to only consider the class of P/T nets for this but it is necessary to also examine the implications of specific subclasses and properties on later analyses. Therefore, the subsequent section will not look at high-level abstractions and extensions beyond P/T nets. Instead, it will scrutinize the class of ordinary Petri nets more thoroughly. By imposing restrictions on these Petri nets, in particular on the net structure, the following different Petri net subclasses can be obtained [32, 51, 101].

3.1.2.1 Place- and Transition-Related Systems

Definition 3.17 (State Machine or S-System⁴). *A State Machine (SM) is a Petri net such that each transition has exactly one input place and one output place.*

⁴ S stands for the word “Stellen” (“places” in English) whose use goes back to Petri’s dissertation, written in German.

Petri net state machines can describe all finite automaton. The characteristic nodes in SMs are the places. Each transition allows the tokens to flow from one place to another, but a token in a particular place may enable multiple transitions, which represents a conflict (as depicted in Figure 3.5).

Definition 3.18 (Marked Graph or T-System). *A Marked Graph (MG) is a Petri net such that each place has exactly one input transition and one output transition.*

The characteristic nodes in a marked graph are the transitions. Each place receives tokens from one transition and loses tokens to another, but a single transition may have multiple input and output places. Marked graphs allow synchronization, as depicted in Figure 3.4. That way, tokens in multiple places are simultaneously lost due to the firing of a single output transition and gained by firing a single input transition.

Some nets, for example, a closed-loop of transitions and places (which does not have any conflict or synchronization), are both an MG and an SM. However, in general, the Petri net subclass of state machines allows the representation of decisions but does not allow for synchronization (concurrency is only possible if more than one token is distributed over the net). On the other hand, marked graphs allow the representation of concurrency but not of decisions or conflicts. Hence, both do not allow the comprehensive modeling capabilities that are required for the approach.

3.1.2.2 Choice-Related Nets

A restricted Petri net subclass that allows for non-trivial behaviors, in particular conflicts and synchronizations, builds the class of free-choice Petri nets, which can be seen as a State Machine enriched with a Marked Graph. Free-choice nets have the behavioral feature that if two transitions share an input place, and if that place is marked, both transitions are enabled, or there is no marking that enables one transition and disables the other one. For instance, the choice construct depicted in Figure 3.5 is an example of free-choice.

Definition 3.19 (Free-Choice Nets). A *Free-Choice (FC) net* is a Petri net such that every arc $\langle p, t \rangle$ from a place p to a transition t is either a unique outgoing arc (t is the only output transition of p (no choice)) or a unique incoming arc to a transition (p is the only input place of t (no synchronization)). Hence, for all pairs of places $p_1, p_2 \in P$, if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$ then $|p_1^\bullet| = |p_2^\bullet| = 1$.

In other words, if any output transition of a place p is enabled, then all output transitions of that place p are enabled. Hence, in this sense, it is possible to freely choose which of the enabled transitions is supposed to fire. Each of the three nets in Figure 3.8 represents a self-contained sufficient description of free-choice nets. To extend the class of nets while retaining this basic property of a free-choice between conflicting transitions, the majority of papers on FC nets use a slightly weaker definition. A system of such a so-called extended free-choice net can be simulated by an FC system.

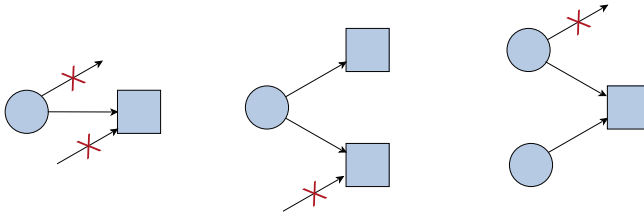


Figure 3.8 Three manifestation of the free-choice property

Definition 3.20 (Extended Free-Choice Net). An *Extended Free-Choice (EFC) net*, is a Petri net such that for all pairs of places $p_1, p_2 \in P$, if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$, then $p_1^\bullet = p_2^\bullet$.

For example, the EFC net in Figure 3.9(a) can be simulated by the FC net in Figure 3.9(b). (Extended) Free-choice nets are essential in the theory of net systems, in particular in the structural theory of Petri nets. Their structural and behavioral prop-

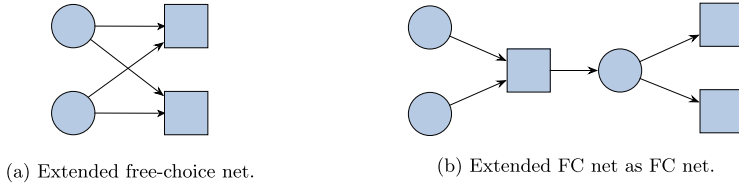


Figure 3.9 (Extended) free-choice

erties are strongly interrelated, i.e., the behavior of a marked net can be connected to the structure of the underlying unmarked net [32]. For example, in the light of the state space explosion problem in behavioral Petri net analysis, such structural methods are more appropriate to deduct if a specific marking can be reached. Even in bounded nets, the state space can be exponential on the size of the net. In FC nets, liveness and well-formedness, which subsumes boundedness, can be linked to siphons and marked traps (Commoner's Theorem [51]) as well as to the rank of the incidence matrix of the net (Rank Theorem [73]). In the course of this work, FC nets are particularly interesting because, firstly, they are able to model the stipulated non-trivial control flow requirements (ROA-2) and, secondly, there are still powerful methods for their analysis and synthesis (ROA-3). While, for example, reachability is EXPSPACE hard for arbitrary Petri nets [146] and still NP-complete [88] for live and safe FC nets, it is efficiently solvable for the class of FC live, safe and reversible Petri nets [75]. Furthermore, the problem of deciding whether an FC Petri net is live and bounded is solvable in $O(n * m)$ time, whereby n is the number of places and m the number of transitions. Many of the analysis problems of live and bounded FC Petri nets have, in turn, been shown to have polynomial time complexity [88].

When decisions are not made “freely” but are influenced by previously executed activities, so-called asymmetric choice (AC) nets may describe this. The combination of choice and synchronization (which has been identified before as “confusion”) is also termed Non-Free-Choice (NFC).

Definition 3.21 (An Asymmetric Choice Net). *An Asymmetric Choice (AC) net (also known as a simple net) is an ordinary Petri net such that for all pairs of places $p_1, p_2 \in P$, if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$, then $p_1^\bullet \subseteq p_2^\bullet$ or $p_2^\bullet \subseteq p_1^\bullet$*

ACs [32] allow asymmetric confusion (see Figure 3.6 for an asymmetric combination of choice and concurrency) but disallow symmetric confusion (cf. Figure 3.7). The post-sets of any two places may be disjoint or identical (as in free-choice), or one post-set may be contained in the other. Hence, asymmetric choice nets represent an extension of free-choice nets where the theorems that are important in FC only hold partially [73, 122]. The study of the property analysis complexity of ACs found, for example, that it is co-NP-hard for the analysis of liveness and boundedness [136].

3.1.2.3 Workflow Net

As a further subclass within the class of ordinary nets, the so-called workflow nets (WF-nets [4]) can be used to represent processes in process-aware information systems with Petri nets. Therefore, the techniques and methods of this thesis assume that the control flow of the process is specified with such a type of net. A WF-net satisfies two requirements: Firstly, a WF-net is a Petri net that has one input place i (denoting the initial state of the system) with no incoming arcs and an output place o (denoting the final state of the system) with no outgoing arcs. A token in i corresponds to a case that needs to be handled and a token in o corresponds to a handled or completed case. The transitions in a WF-net represent tasks and the places represent conditions. Both should contribute to the processing of a case, such that in a WF-net, there are no dangling tasks or conditions. Therefore, every transition t , or place p , respectively, should be located on a path from place i to place o . The later requirement corresponds to strong connectedness (cf. Definition 3.3). Its examination requires that o is connected to i via an additional transition t^* [4]. These two structural properties are termed “WF-structuredness”.

Definition 3.22 (WF-Net). A Petri net $N = \langle P, T^5, \mathcal{F}, m_0 \rangle$ is a WF-net (Workflow net) iff [4]:

- (i) N has two special places: i (input) and o (output). Place i is a source place: $\bullet i = \emptyset$. Place o is a sink place: $o^\bullet = \emptyset$.
- (ii) If a transition t^* is added to N that connects place o with i (i.e., $\bullet t^* = \{o\}$ and $t^{*\bullet} = \{i\}$), then the resulting extended Petri net \overline{PN} is strongly connected.

⁵ The transitions in a WF-net typically represent tasks, which is why, for syntactic simplicity and legibility, the notation t is overloaded for both transitions and tasks (or T for the respective sets).

The example of a Petri net in Figure 3.1 already represents a WF-net. It can be regarded as the mapping of the BPMN workflow model of the “determine market value” process into a WF-net representation, in which t_1 represents the “determine market value” task, and t_2 stands for the “control computation” task. Moreover, there is a source place, and a sink place whereas all other nodes are on a path between them.

In order to be able to elaborate on possible task execution sequences of a given workflow net, systems nets are introduced. This will particularly be relevant to grasp complete but also incomplete execution sequences.

Definition 3.23 (System Net, Full and Terminal Firing Sequences). *A System Net (SN) defines a set of sequences, each one starting from the initial marking and ending in the final marking. A SN is a tuple $SN = (N, m_{start}, m_{end})$, where N is a WF-net and the two last elements define the initial and final marking of the net, respectively. The set $F_{SN} = \{\sigma \mid (N, m_{start})[\sigma](N, m_{end})\}$ denotes all the full firing sequences of SN .*

In this work, the notion of an end marking or final state differs from a terminal state. Final states, such as m_{end} , are defined. Terminal states represent the states (markings) that do not allow the firing of any further transition. Accordingly, the set $\mathcal{T}_N = \{\sigma \mid (N, m_0)[\sigma](N, m') \wedge m' \in [m_0] \wedge \nexists t \in T : m'[t]\}$ denotes all the terminal firing sequences of the net N with its initial marking m_0 . Accordingly, based on an initial marking m_0 , the set of terminal markings $\{m \mid m \in [m_0] \wedge \nexists t \in T : m[t]\}$ is denoted by $[m_0]_{\mathcal{T}}$.

For example, for the WF-net in Figure 3.1, the set of full firing sequences of the system net $SN = (N, \{p_0\}, \{p_2\})$ is $F_{SN} = \{t_1, t_2\} = \mathcal{T}_N$. That way, it is possible to link reachability with completability because a non-reachable end marking may indicate an unsatisfiable workflow. The fact that the firing sequence of the example net encodes safety properties (e.g., t_1 occurs before t_2) and liveness properties (e.g., eventually t_2 occurs) underlines that each property of an initially marked Petri net can be composed of its safety and liveness properties [13].

3.1.2.4 Workflow Soundness

The requirements stated for a WF-net in Definition 3.22 are minimal requirements [4]. Even if these requirements are satisfied, it is still possible to determine a workflow process definition with potential deadlocks. Hence, WF-nets must meet some properties to avoid unexpected results subsumed in the subsequent soundness

criteria [7]. To a large extent, these assumptions often refer in some way to the absence or existence of deadlocks, which underlines, in turn, the effort to ensure the completion of the workflow. Based on the typical properties of a P/T net and WF-structuredness, a WF-net is sound if and only if the following three requirements are satisfied:

Definition 3.24 (Soundness of a Workflow Net). *A WF-net with input place i and output place o is sound if the following conditions are met:*

- (i) *Option to complete: For each case and any marking reached from the initial marking it is still always possible to reach a state that marks the end place o , i.e., $\forall m \in R(N, i), m(o) \in R(N, m)$;*
- (ii) *Proper completion: If o is marked all other places are empty for a given case, i.e., $\forall m \in R(N, i)$, if $o \in m$ then $m = \{o\}$;*
- (iii) *No dead transitions: It is possible to execute an arbitrary activity by following the appropriate firing sequence through the WF-net, i.e., $\forall t \in T, \exists m \in R(N, i)$ such that $(N, m)[t]$.*

To analyze these criteria, the structural characteristics of the regarded nets are again decisive. While, for a complex WF-net, it may be intractable to decide on its soundness, if the structure of a WF-net is, for example, free-choice, the soundness problem is solvable in polynomial time [7]. In contrast, the soundness problem in asymmetric choice WF-nets has been proven to be Co-NP-Hard [136].

Block-Structure: As a further way to facilitate soundness analysis, another approach that aims to obtain a structural characterization of “good” workflows is to assume block-structured models [7]. Although, in general, there are different definitions in this respect [105, 140, 212], the property of such block-structured models usually refers to the need for synchronization of splits and joins of the paths in the control flow. These blocks are entered as well as left at a single point. That way, there is only a single branch going in and out, before and after the block. Thus, these blocks are also called single-entry single-exit (SESE [40, 110]) regions. Subsequently, the fundamental types of such blocks in workflows, which are also known as basic workflow patterns [6], will be illustrated (building on the concise summary of the patterns from Carmona et al. [40]). Thereby, their similarity to the previously introduced basic Petri net examples will become evident.

- Sequential Pattern:** The sequence pattern forms a strong relationship between the connected tasks (see Figure 3.10). It restricts the process in such a way that activities are only executed in the sequential order specified for the respective tasks.



Figure 3.10 Sequential pattern

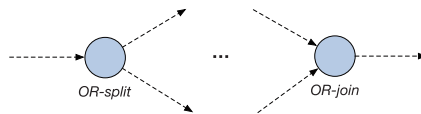


Figure 3.11 Exclusive pattern

- Exclusive Choice:** The exclusive choice pattern depicted in Figure 3.11 can be used to represent a conflict or choice in a process (e.g., the conflicting activities t_1 and t_2 in the net in Figure 3.5). Two modeling constructs help in designing models that contain choices in a block-oriented way: first, the exclusive split (or OR/XOR-split), which indicates the mutually exclusive alternatives to continue at a specific point, and second, the exclusive join, which brings together alternative branches in the model to continue from there on a common path.
- Parallel Execution:** There may be activities in a process, which all need to be performed, but there are no dependencies between them (e.g., the activities t_1 and t_2 in the net in Figure 3.4). Thus, the activities can be performed in parallel. Parallel execution starts at the parallel split (or AND-split). It shows that all outgoing branches are executed independently. Within the branches that leave the parallel split, other workflow patterns or block structures can still be used (i.e., “nested”). If the execution of the parallel branches is terminated and any further activity in the process depends on these branches one after the other, all these branches will need to be completed before moving on. To capture this synchronization in a process model, a parallel join is used. It has several incoming branches and a single outgoing branch. The parallel join operation ensures that the process cannot continue until all incoming branches are completed. Figure 3.12 depicts the respective split and join.
- Loop:** The loop pattern can be realized by a combination of the exclusive split and the join (see Figure 3.13). Unlike with the normal exclusive pattern, it is

possible to choose to go back to an earlier point in the process to indicate that one wishes to repeat the activities represented by the respective tasks starting from that point. That way, the control flow forms a loop that can have as many iterations as desired. Each time the same decision point is reached, it can be decided whether to go back and do another iteration or to leave the loop and continue the process. It is important to note that such a loop or cycle begins with an exclusive join operation that merges the current control flow with a branch that may only be taken in the future. Hence, such a loop has a single-entry and a single-exit point as well.

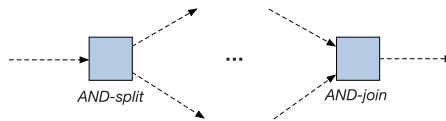


Figure 3.12 Parallel pattern

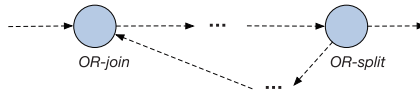


Figure 3.13 Loop pattern

Based on these blocks, it is possible to build more complex process models by nesting these workflow patterns. Despite the specific pattern, it can be observed that splits initiate branching and thus set paths in either a parallel or exclusive relationship to each other. Joins reunite these paths, i.e., they synchronize them. Hence, two parallel flows initiated by a parallel split must not be joined by an exclusive join. Equivalently, two alternative flows created via an exclusive split, must not be synchronized by a parallel join. Instead, a parallel join should complement a parallel split, and an exclusive join should complement an exclusive split [7]. If a model consists only of such blocks, or such SESE regions, respectively, the model is called block-structured (or well-structured). “Unstructured models” describe models that do not have this property. To define this characteristic of balancing equivalent splits and joins for WF-net models, the existence of PT- and TP-handles will be examined.

Definition 3.25 (Well-Handled). A Petri net PN is well-handled iff it has neither PT -handles nor TP -handles.

A Petri net that is well-handled has several desirable properties, for example, strongly connectedness and well-formedness coincide [7]. That way, a well-handled extended WF-net can be live and bounded for the initial marking $m = \{i\}$.

Definition 3.26 (Well-Structured). A WF-net PN is well-structured iff its extended net \overline{PN} is well-handled.

Due to the well-structuredness property, it is possible to deduce further properties from WF-nets. To decide if a well-structured WF-net is sound can be verified in polynomial time. Moreover, a sound well-structured WF-net is safe [7].

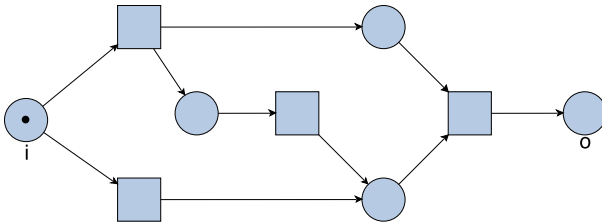


Figure 3.14 Example of a live, safe, FC, but not well-structured WF-Net

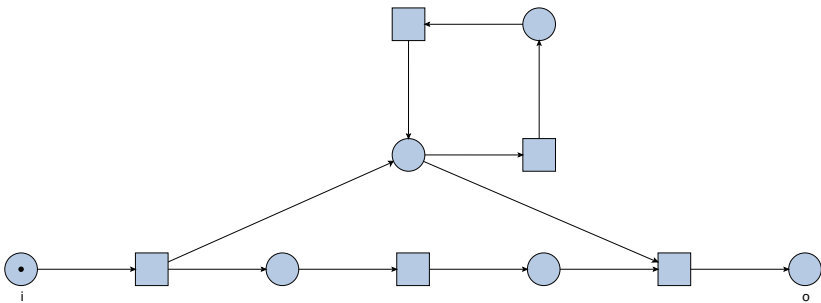


Figure 3.15 Example of a live, safe, AC, well-structured, but not FC WF-Net

SESE regions support applying specific reductions as well [200], i.e., net reduction methods allow transforming Petri nets into simpler nets while keeping properties of the initial net. Because the problem size of a net is reduced that way, net reductions can facilitate later analysis.

Block-Structured Free-Choice WF-Nets: Regarding the different identified net classes, it can be summarized that SMs admit no synchronization, MGs admit no conflicts, FCs admit no confusion, and ACs allow asymmetric confusion (but disallow symmetric confusion) [155]. When these subclasses are associated with block-structured models, it seems that SESE models already imply a certain “good structure” as well. However, the subsequent counter-examples are constructed to show that block-structured models do not necessarily imply free-choice or even asymmetric choice. There are live bounded free-choice WF-nets that are not well-structured (Figure 3.14) as well as well-structured live bounded WF-nets that are not free-choice (Figure 3.15) or not even AC (Figure 3.16).

To unite the advantages of free-choice and block-structured models for the control flow of the process, this thesis assumes WF-nets that are both FC and SESE. Besides the already illustrated mild computational aspects of FC nets and block-structured models, working with a block structure that allows for all required modeling constructs will be beneficial for modeling further constructs into the model as well.

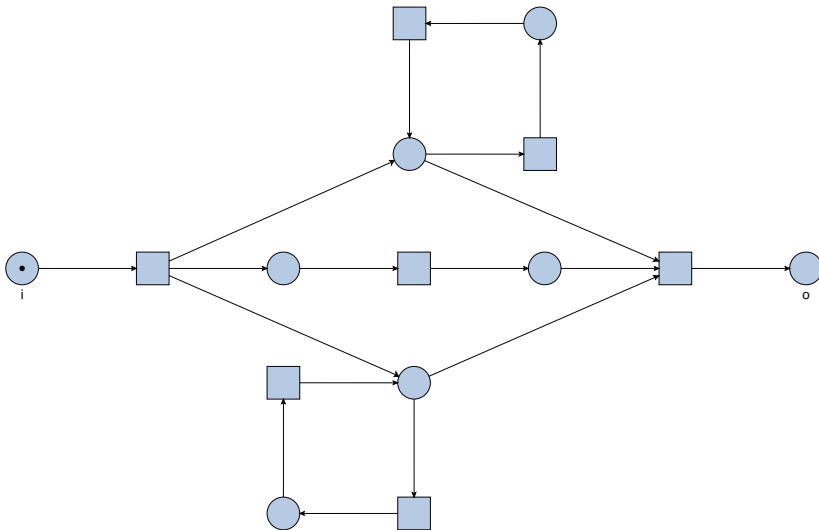


Figure 3.16 Example of a live, safe, well-structured, but neither FC nor AC WF-Net

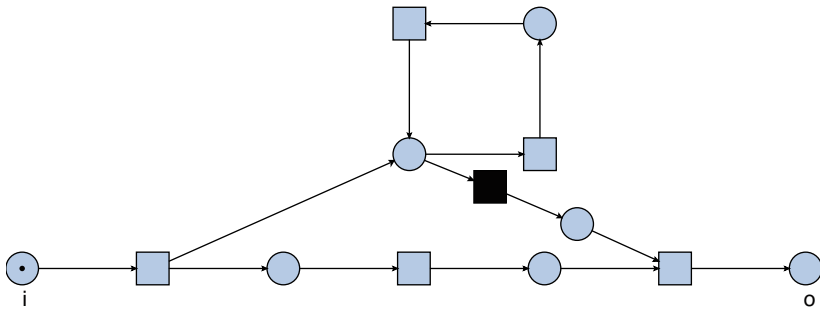


Figure 3.17 Example of a live, safe, well-structured, and now also FC WF-Net

In particular, such SESE regions will help to identify blocks, for which parts of the authorization policy and constraints have to be modeled, namely, blocks that encode sequential, exclusive, parallel, and loop patterns. Loops, or loop-affected blocks, can be identified easily because the block structure explicitly determines how loops may occur.

Still, there may be the problem that a given block-structured WF-net model is not free-choice. However, for the envisaged WF-net type, this is a surmountable difficulty because there are transformations for arbitrary Petri nets into free-choice nets [124, 187, 188]. In general, such transformations increase the structural complexity of the nets, in particular the number of nodes and arcs. However, given block-structured models, the considered NFC nets are not that “arbitrary” anymore and already show such a structural limitation that they can often be transformed into a free-choice net by means of only a few modifications. For example, to make the net from Figure 3.15 free-choice, only a single so-called silent transition (which merely acts for routing purposes) and a single place needs to be added (see Figure 3.17). Thus, the increase in the structural complexity that the transformation requires remains moderate. However, such a modification, consequently extends the state space. It creates intermediate steps and intermediate states, which must be considered as such to ensure the interpretation of the original net.

To illustrate a comprehensive example of such a WF-net net, the CEW process from the previous chapters will be considered again. As indicated before, there are straightforward transformations into Petri nets for a subset of BPMN elements. Figure 3.18 depicts this basic mapping that goes back to Dijkman et. al [79]. Due to the block-structure of the BPMN model in Figure 3.19, transforming the BPMN model results in a block-structured and also free-choice Petri net, which can be seen in Figure 3.20.

3.1.2.5 Policy-Aware Workflow Net

To prepare a directly processable formal input for the approach that entails not only the so far emphasized control flow of the process but also the whole process specification, the definitions of authorizations and constraints will also be directly related to the WF-Net. This will prepare a process-oriented definition of the policy as well. The definitions are adapted from the WSP-related works of Crampton et al. [49, 60] because they allow for a simple, though, comprehensive representation of authorization and relevant constraints.

User-Task Authorization: In practice, the authorization policy will not explicitly be defined as a user-task authorization. Instead, the authorization policy will be inferred from typical access control data structures [58]. Because, for common access control policies (e.g., RBAC), it is straightforward to derive the tasks for which users are authorized, a user-task authorization will be used in order to simplify the exposition. By making the inputs simple, the idea of keeping the net constructs involved as simple as possible, as described in the introduction to this chapter, is underlined.

Definition 3.27 (User-Task Authorization). *Because the transitions into a WF-net represent tasks, given a workflow net $N = \langle P, T, \mathcal{F}, m_0 \rangle$ and a set of users U , an authorization policy (or a user-task authorization) for N can be denoted as a relation $TA \subseteq U \times T$, which may be represented as a set of authorization lists $\mathcal{A} = \{TA(u) : u \in U\}$, where $TA(u) = \{t \in T : (u, t) \in TA\}$ denotes the set of tasks for which u is authorized. Analogously, it may be represented as a set of task-assignment lists $\mathcal{T}\mathcal{A} = \{TA(t) : t \in T\}$, where $TA(t) = \{u \in U : (u, t) \in TA\}$ denotes the set of users that are authorized for t . User u is authorized to perform task t iff $(u, t) \in TA$. It is assumed that for every task $t \in T$, there is some user $u \in U$ such that $(u, t) \in TA$ ⁶.*

For example, the user-task authorization depicted in Figure 3.21(a) is formalized as $TA = \{(Alice, t_1), (Alice, t_2), (Bob, t_1)\}$, the authorization list as $TA(Alice) = \{t_1, t_2\}$ and $TA(Bob) = \{t_1\}$.

⁶ This implies that trivial cases of unsatisfiability are avoided.

Constraints: As mentioned before, although further constraints regarding workflow satisfiability analysis have already been investigated [61], the focus is on common user-independent constraints, namely SoD/BoD-related binary constraints, which suffice to reach an obstructed state (ROA-4). For the sake of completeness, it should be mentioned that an authorization list may be interpreted as a unary constraint as well, where the scope of the constraint is a single task [50]. However, as elaborated in Chapter 2, a separated view on these both types will be maintained.

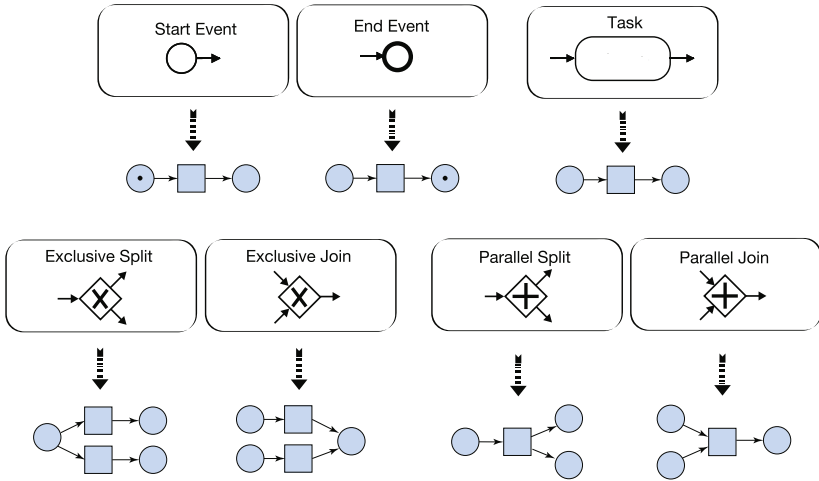


Figure 3.18 Basic BPMN to WF-net patterns

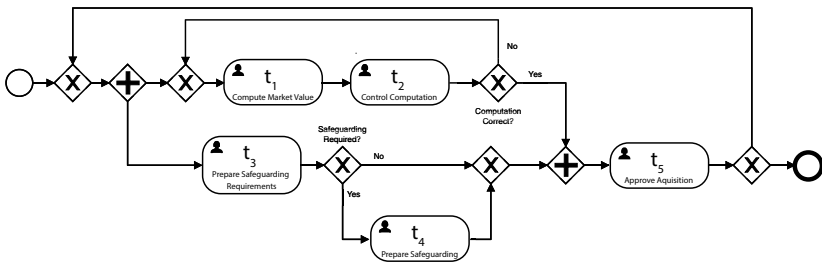


Figure 3.19 CEW process in BPMN

Definition 3.28 (Constraints). A constraint $c \in C$ may be viewed as a pair (T_c, Θ) , where $T_c \subseteq T$ is the scope of c and Θ is a set of functions from T_c to U , specifying the assignments of steps in T_c to users in U that satisfy the constraint. In practice, the elements of Θ are not enumerated. Instead, its members are defined by implicitly using some constraint-specific syntax. For example, (t, t', ρ) can be written, where $t, t' \in T$ and ρ is a binary relation defined on U , to denote a constraint that has scope $\{t, t'\}$ and is satisfied by any assignment $\pi : T \rightarrow U$ such that $(\pi(t), \pi(t')) \in \rho$. In particular, the separation-of-duty constraint (t, t', \neq) requires t and t' to be performed by different users. Similarly, the binding-of-duty constraint $(t, t', =)$ requires t and t' to be performed by the same user.

For example, the SoD constraint indicated in Figure 3.21(b) can be denoted as (t_1, t_2, \neq) . Although the focus is on such SoD and BoD constraints, the possibilities of considering further constraints will be addressed as well at the end of this chapter. Based on these definitions, it is now possible to define the input for the approach, namely a policy-aware workflow net. It comprises a WF-net with policy formalizations tailored to it and thus bundles the different aspects of a security-aware process specification. In other words, it contains all inputs required for the desired representation.

Definition 3.29 (Policy-aware Workflow Net). A policy-aware workflow net N^{pol} is represented as a tuple $\langle N, U, TA, C \rangle$, where N is a WF-net, U is a set of users, $TA \subseteq U \times T$ is the user-task authorization, and C is a set of constraints.

Thus, the complete security-aware specification of the DMV process can, for example, now be formalized as a policy-aware WF-net $N^{pol} = \langle N, U, TA, C \rangle$, where $N = \langle \{p_1, p_2, p_3\}, \{t_1, t_2\}, \{\langle p_1, t_1 \rangle, \langle t_1, p_2 \rangle, \langle p_2, t_2 \rangle, \langle t_2, p_3 \rangle\}, \langle 1, 0, 0 \rangle \rangle$, $U = \{Alice, Bob\}$, $TA = \{(Alice, t_1), (Alice, t_2), (Bob, t_1)\}$, and $C = \{(t_1, t_2, \neq)\}$.

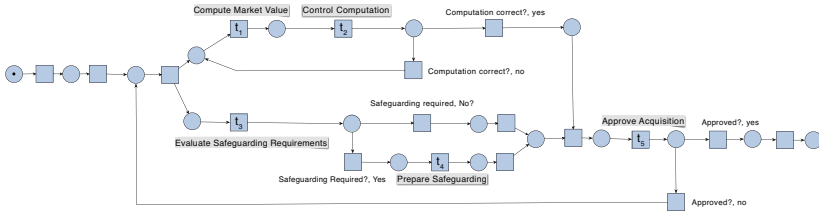


Figure 3.20 Comprehensive example of a block-structured free-choice WF-net

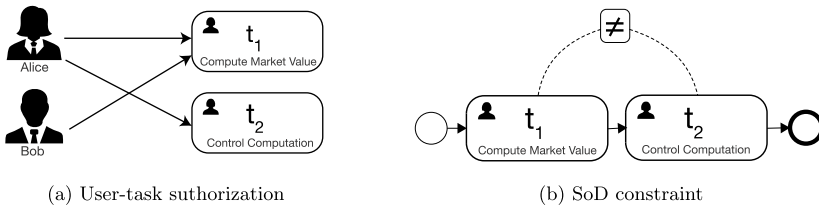


Figure 3.21 Determine the market value [39]

3.2 The SecANet Solution

This section proposes the SecANet approach as a way to analyze and capture process obstructions in PAISs. Its methodology will allow the flattening of a workflow and the corresponding authorization data into a Security-Aware Petri net (which gives the approach its name). That way, a process representation that not only includes functional and behavioral but organizational process aspects as well emerges. Besides desirable Petri net properties, language-related properties are of particular interest to examine the behavior of the different process aspects in the resulting representation. In this regard, the approach is supposed to preserve the integrity of all process aspects involved, which will be regarded on different levels.

- Integrity of Inputs:** In this thesis, the different process aspects manifest themselves in the process model and its policies. Each such input constitutes a common (abstract) representation used in a PAIS. The encoding of such input is supposed to preserve its initial behavior. Moreover, it should be possible to unambiguously retrace results from the representation back to given inputs. In case of an obstruction, this allows deducing conclusions or improvements for the process design.

- Integrity of the Overall Representation:** The representation is supposed to provide a consistent full picture of the overall process in a PAIS. For this, the overall representation needs to preserve the behavior of the initial inputs. Moreover, each part of the overall representation is supposed to allow traceability to its corresponding initial input as well.

Based on an example, this section will first depict the general idea behind the principle of flattening and resulting possibilities for analyses. For behavioral observation, it will first be sufficient to consider related full firing sequences. A more thorough language-oriented examination will be performed after the introduction of the SecANet definitions. Based on the example, the approach will be generalized for authorization policies as well as SoD and BoD constraints. That way, based on a WF-net, obstructions can be captured with an “obstruction marking”. The representation will already be prepared in such a way that it will allow encoding cycles (loops) as well. For the sake of simplicity and efficiency, this generalized encoding will be described on acyclic nets first. After introducing the formal basis for language-related observations on Petri nets, this section will decompose the approach to

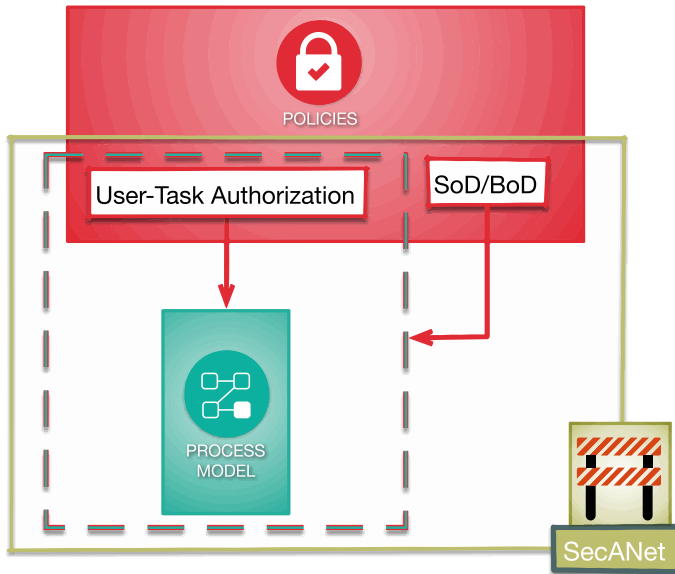


Figure 3.22 The SecANet approach

examine its language. The resulting SecANet language will allow investigating the behavioral integrity of inputs as well as the integrity of the overall process. Then, nets containing cycles will be introduced. They have a more complex encoding but preserve the idea of the encoding described in the acyclic case. An example will show the flattening of a process specification that involves a comprehensive workflow structure. It will be used to illustrate obstructability analysis resulting from the execution semantic of a SecANet.

3.2.1 The Principle of “Flattening”

Figure 3.22 depicts the SecANet approach, in particular, its idea of flattening: Based on a policy-aware workflow net (Definition 3.29), the policy is supposed to be encoded into the given workflow net step by step. As elaborated before, the authorization and constraints can be understood in a process-oriented way. User-task authorization is the process of granting user access to execute a specific task. Constraints impose restrictions on this authorization process. Because the assignment of a user to a task can only be done before its execution, the general idea is to model policy specific sub-processes that operate as a precondition to the actual activities of the control flow of the business process. In the context of Petri nets, this means that a task (transition) affected by the policy may only be enabled if the corresponding

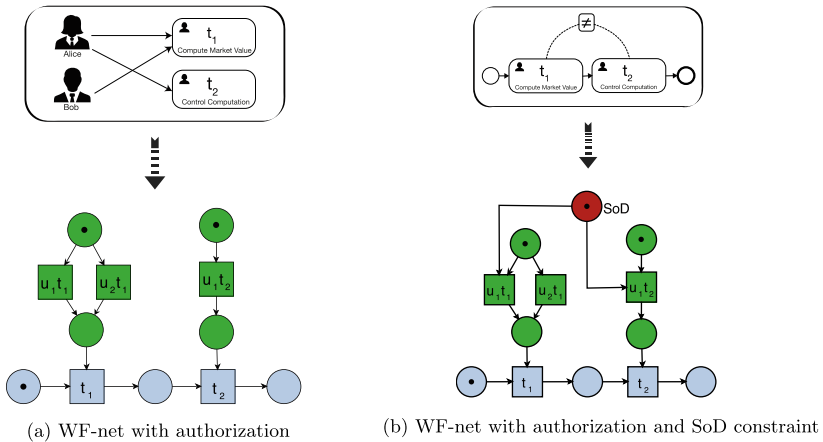


Figure 3.23 Flattening authorization and SoD into the DMV WF-net

user-task authorization or constraint is fulfilled. Thus, the idea is to put a net construct that models the part of the policy required to execute the corresponding task in each pre-area of the respective transition. Based on this general idea, the principle of flattening will first be described with the help of the example policy for the DMV process presented in Figure 3.21. Here, the authorization policy and constraints will be flattened into the DMV WF-net step by step. The resulting net will illustrate how the states provided by Petri nets can be used to capture an obstruction and how to analyze satisfiability and obstructability. Thereby, desirable Petri net properties and characteristics against the background of the established requirements will be indicated. A more detailed examination of the properties of the created net constructs will follow along with the generalization of the approach.

3.2.1.1 Modeling Authorization

Because of the absence of ambiguous gateways in the BPMN model of the DMV process, the workflow can straight-forwardly be transformed into the WF-net in Figure 3.1. For better clarity, the input place p_0 of the WF-net will subsequently be represented by p_i and the output place p_2 by p_o . To model access control, the simple access control model without roles from Figure 3.21a is assumed, namely $TA = \{(u_1, t_1), (u_1, t_2), (u_2, t_1)\}$. Such user-task authorizations represent safety properties that explicitly state what is allowed. As mentioned before, a user-task authorization can also be regarded as a unary constraint that relates authorized user accesses to the scope of a single task. These possible assignments of users to tasks are mutually exclusive since only one of the authorized users can eventually be assigned to execute a task. Based on these considerations, Figure 3.23a depicts the first flattening step of the DMV WF-net example. It illustrates the initial user-task authorization and its flattened counterpart with authorization-related Petri net elements (the Petri net construct above the initial WF-net). The user-task assignments encode the corresponding transitions. For example, transition $t_{u_1 t_1}$ encodes that user u_1 is permitted to access t_1 . Each transition pre-area consists of the same single incoming place marked with a single token such that a mutually exclusive choice must be made. Hence, a marked incoming place represents the state in which no user has yet been assigned to the corresponding task. It enables all transitions that encode the possible user-task assignments provided by the authorization policy for the corresponding task. Firing $t_{u_1 t_1}$, for instance, then represents the decision that u_1 is assigned to the execution of task t_1 . In the post-area of all user-task transitions corresponding to the same task, there is the same single outgoing place. The firing of $t_{u_1 t_1}$, for example, then produces a single token in this place. Hence, a marked outgoing place indicates the state that a user has been assigned to a task. However,

the related task has not yet been executed. That way, pre-assignments (ROA-1) may also be considered (in case of the unordered version of obstructability, cf. Chapter 2).

Based on this example encoding, the full and terminal firing sequences can now be observed. For a more intuitive understanding of the net behavior, the number of possible permutations of these sequences is limited by conducting a “lazy” workflow-oriented execution. This execution strategy aims to complete the workflow with the least effort by always trying to execute the workflow tasks in the first place. If some next workflow is not executable, the transitions in its pre-area are fired in such a way that they reach an enabling marking for that task. Based on this, the set of full and the set of terminal firing sequences for the WF-net with authorization, where $m_{start} = \{p_1, p_{t_1-}, p_{t_2-}\}$, and $m_{end} = \{p_3\}$, is

$$F_{SNTA} = \{\langle t_{u_1 t_1}, t_1, t_{u_1 t_2}, t_2 \rangle, \\ \langle t_{u_2 t_1}, t_1, t_{u_1 t_2}, t_2 \rangle\}, \\ \text{and} \\ \mathcal{T}_{NTA} = F_{SNTA}.$$

The individual user-task transitions directly indicate (and allow to retrace) the underlying authorization policy (e.g., as a user-activity matrix), thus preserving the behavior of the exemplary user-task authorization input $TA = \{(u_1, t_1), (u_1, t_2), (u_2, t_1)\}$.

3.2.1.2 Modeling Constraints

Contrary to the authorization policy, which determines allowed user-task assignments, constraints specify safety properties that explicitly state violations (i.e., what is not allowed). These constraints act upon the authorization policy and are mutually exclusive for the respective user-task assignments between two (sets of) tasks (i.e., binary constraints). More specifically, constraints mutually exclude that the same user is assigned to a scope of tasks (SoD) or that different users are assigned to a scope of tasks (BoD). That way, constraints restrict possible user-task assignments. Consequently, only users authorized for the tasks that lie in the scope of a constraint can be constrained at all. Hence, there is the implicit assumption that the users affected by an SoD or BoD constraints are part of the user-task authorization as well. For Petri net modeling, this means that constraints are supposed to act upon the user-task assignment transition and do not operate on the tasks themselves for which they are defined (although, for example, the constraint (t_1, t_2, \neq) itself may suggest this). Hence, the envisaged encoding of constraints is supposed to restrict the given user-task assignments.

Regarding the modeling of constraints, it can be observed that a Petri net without any places and a non-empty set of transitions allows for any “unconstrained” behavior involving the activities represented by these transitions. Adding a place can then be compared to the introduction of a constraint. The underlying fundamental idea of Petri nets is that transitions are independent (i.e., concurrent) unless specified otherwise [1]. Concurrency also applies to the occurrence of the user-task assignments for different tasks because they occur independently from other task-assignments. Hence, places that establish relations between the user-task transitions for different tasks can be used to constrain their occurrence as well. In particular, such places can be used to model a choice (or conflict) between the user-task transitions for different tasks. For a BoD constraint on two tasks, the choice, which of the users executes both tasks, has to be made. This choice then means that all other user-task assignments involving other users for the given tasks are excluded. For an SoD constraint on two tasks, the choice, which of the users executes the one or the other task, has to be made.

The introductory example will focus on SoD constraints to illustrate the idea behind constraint modeling. These observations will then be used for the generalization of constraint modeling, where BoD constraints will be covered as well. For tasks affected by an SoD constraint, a place is added for every two corresponding user-task assignments. The place is then connected to the user-task assignment transitions. It encodes a choice between the connected transitions. That way, such a place and its outgoing arcs pointing to the involved two user-task transitions in conflict reflect an SoD constraint on the two activities for which the same user is authorized. Based on this, the example in Figure 3.23 shows how the SoD constraint (t_1, t_2, \neq) can be flattened into the WF-net with authorization (highlighted in red). By introducing the place encoding choice (or, “choice-place”), denoted as “SoD”, the set of full firing sequences, where $m_{start} = \{p_1, p_{t_1-}, p_{t_2-}, p_{SoD}\}$ and $m_{end} = \{p_3\}$ is constrained (depicted by a comparison with the full firing sequences of $F_{SN_{TA+SoD}}$):

$$F_{SN_{TA+SoD}} = \left\{ \langle \cancel{t_{u_1 t_1}}, t_1, \cancel{t_{u_1 t_2}}, t_2 \rangle, \langle t_{u_2 t_1}, t_1, t_{u_1 t_2}, t_2 \rangle \right\}.$$

Hence, only the latter firing sequence of $F_{SN_{TA+SoD}}$ meets the SoD constraint. The set of terminal firing sequences is now not in accordance with the full firing sequences anymore.

$$\mathcal{T}_{N_{TA+SoD}} = \{ \langle t_{u_1 t_1}, t_1 \rangle, \langle t_{u_2 t_1}, t_1, t_{u_1 t_2}, t_2 \rangle \}.$$

Based on this, it can be observed that the set of full firing sequences is reduced. More specifically, the terminal firing sequences of the user-task authorization construct is restricted by the modeled constraint. Although each transition is enabled for some marking of the net (i.e., it is simply live), this may not mean that all enabled transitions are part of a single full firing sequence. The full firing sequences themselves stay the same. Hence, only the latter firing sequence meets the SoD constraint in the example and some full firing sequences are not allowed anymore. However, it can be noted that the leftover full firing sequences do not change in themselves. This will be of particular interest for behavioral observations.

3.2.1.3 Example SecANet Analysis

Given that the authorization policy and constraints have now been flattened into the SecANet example with the initial marking represented in Figure 3.23b, playing the token game is going to show how it can be used for analysis and to capture an obstructed state. The set of (full) firing sequences encompasses not only the process activities and their execution order but also the information who is assigned to the corresponding activities or tasks. Hence, the transition firing sequences of a SecANet can be regarded as traces that can be used to check which safety and liveness security properties they may or may not fulfill. Because safety properties are encoded and enforced by the flattening of the policy, the examination of the liveness of the overall system is of particular interest, i.e., if the WF-net execution can be completed with the given policy (or, in other words, if the process goal can be achieved).

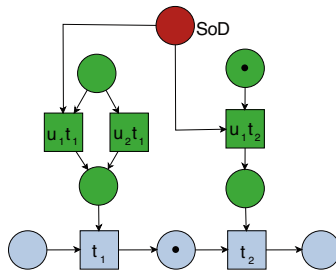


Figure 3.24 Obstructed marking in flattened WF-net

Satisfiability: Chapter 2 has already shown that the DMV process and its policy is satisfiable. The example SecANet can now be used to illustrate this as well. By firing $t_{u_2u_1}$ (u_2 is assigned to t_1) and firing t_1 (the execution of the first task), followed by firing $t_{u_1t_2}$ (u_2 is assigned to t_2) and then firing t_2 (the execution of the second task), the output place is marked. This means that the workflow and its policy are satisfiable. Thus, satisfiability can be assessed by looking at the set of full firing sequences of the SecANet, which is $\langle u_2u_1, t_1, u_1t_2, t_2 \rangle$ for the system net $SN_{\text{SecANet}} = (N_{\text{SecANet}}, m_{\text{start}}, m_{\text{end}})$, where $m_{\text{start}} = \{p_i, p_{t_1-}, p_{t_2-}, p_{\text{SoD}}\}$ and $m_{\text{end}} = \{p_o\}$. Concerning the WSP, the set of full firing sequences represents synthesized plans to conduct satisfiable workflow executions. If such plans exist, such sequences fulfill the safety properties given by the policy as well as the liveness property of process completion. An empty set of full firing sequences would therefore mean that the workflow is not satisfiable.

Capturing Obstructions: As identified in Chapter 2, besides its satisfiability, the respective process specification inherits an obstruction. By playing the token game again, the obstruction can easily be identified. More specifically, firing $t_{u_1t_1}$ and t_1 (t_1 has been executed by u_1) results in an obstructed state, in which no further transition is enabled (as illustrated in Figure 3.24). This state of obstruction can be denoted by the marking $m_{\otimes} = \{p_1, p_{t_2-}\}$, where \otimes is the symbol that denotes “obstruction” (in its abbreviated form “ox”). The obstructing execution sequence can be denoted by $\sigma_{\otimes} = \langle u_1t_1, t_1 \rangle$. The reachability of this obstructed marking can be denoted by $m_0[t_{u_1t_1}, t_1]m_{\otimes}$. Hence, there is a decisive difference between the two nets, which can be observed if not only the full firing sequences but all terminal firing sequences are considered. Such terminal firing sequences only reach markings in which no further transition is enabled, and which are also known as “deadlock markings”. For the WF-net with authorization N_{TA} , the terminal firing sequences are just the same as its full firing sequences $F_{SN_{TA}}$. However, after adding the SoD constraint, the set of full firing sequences $F_{SN_{TA+SoD}}$ loses a sequence of $F_{SN_{TA}}$. Moreover, $\mathcal{T}_{N_{TA+SoD}}$ now contains the additional terminal firing sequence $\langle t_{u_1t_1}, t_1 \rangle$, which does not reach the intended final marking (and is thus not in the set $F_{SN_{TA+SoD}}$). Still, $\langle t_{u_1t_1}, t_1 \rangle$ is a feasible sequence of transitions. It only contains the first two transitions of the missing full firing sequence $\langle t_{u_1t_1}, t_1, t_{u_1t_2}, t_2 \rangle$ of $F_{SN_{TA}}$ and represents a synthesized partial plan, which, however, leads to an obstruction. Hence, based on a WF-net with authorization, the modeling of constraints can now, for the first time, lead to deadlocks in the resulting nets, in the sense that the output place p_o of the WF-net can no longer be reached. Such a deadlock manifests the obstructive conflict between the organizational aspect and the functional aspect of the workflow. Hence, apart from the end marking, a marking in a SecANet,

in which no transition is enabled, represents an obstructed state. Here, a decisive advantage of Petri nets comes to light. They allow having a single representation of the system that captures the event sequences and the state at the same time. In this way, Petri nets do not only record how the obstruction occurred but they also capture the system state of the obstruction of the affected process in a PAIS. That way, a SecANet can be used to synthesize partial plans that encode obstructed workflow executions. Such obstructed partial plans or sequences fulfill and enforce the policy-based safety properties as well. However, these partial plans lack in satisfying the liveness property of process termination, i.e., they can not be used to complete the workflow.

Obstructability: Obstructability analysis is supposed to detect and capture obstructions. Based on the observations from the example, deadlocks that do not reach the final marking indicate obstructions. For small nets, such obstruction markings can be found by hand, for example, with the token game as done above. For bigger Petri nets, there is a variety of methods for analyzing and detecting deadlocks, which is a fundamental issue in Petri net analysis [45, 51, 157, 219]. Their suitability will be illustrated after the subsequent generalization of the flattening example.

3.2.2 Generalizing Flattening

Based on the presentation of the basic idea of flattening, the SecANet approach will subsequently be generalized. The aim is to flatten the user-task authorization TA and the set of constraints C of a policy-aware workflow net $N^{pol} = \langle N, U, TA, C \rangle$ into a single net N_{TA+C} . As a first step, the user-task authorization TA is considered and encoded into the net N_{TA} . Based on this, further SoD and BoD constraints are flattened into the net, denoted by $N_{TA+C_{SoD}}$ and $N_{TA+C_{BoD}}$.

3.2.2.1 Flattening of User-Task Authorization

As described before, the user-task authorization is flattened with the intention that for every possible assignment, a transition denoted by user and transition name (e.g., $t_{u_1t_1}$) is introduced. Each user-task transition can consume tokens from the single place marked with a single token. This ensures that the transition can only be executed once by a specific user. Moreover, this user can execute the corresponding task only once.

Definition 3.30 (Flattening User-Task Authorization). Given a policy-aware workflow $N^{pol} = \langle N, U, TA, C \rangle$, flattening the user-task authorization TA into N is as follows:

1. For each transition t_i in N , create a place p_{t_i-} and a place p_{t_i+} representing the state that no user is assigned ($-$) or a user was assigned ($+$) to execute t_i , respectively, and mark each of the p_{t_i-} places with one token.
2. For each user-task authorization $(u_j, t_i) \in TA$, create a transition $t_{u_j t_i}$.
3. For every place p_{t_i-} and its corresponding transition(s) $t_{u_j t_i}$, create an arc $\langle p_{t_i-}, t_{u_j t_i} \rangle$.
4. For every transition $t_{u_j t_i}$ and its corresponding place p_{t_i+} , create an arc $\langle t_{u_j t_i}, p_{t_i+} \rangle$.
5. For every place p_{t_i+} and its dedicated transition t_i , create an arc $\langle p_{t_i+}, t_i \rangle$.

After performing these steps, the net $N_{TA} = \langle P_{TA}, T_{TA}, F_{TA}, m_{TA_0} \rangle$ is obtained, where

$$\begin{aligned}
 P_{TA} &= P \cup \{p_{t_1-}, p_{t_2-}, \dots, p_{t_i-}\} \cup \{p_{t_1+}, p_{t_2+}, \dots, p_{t_i+}\}, \\
 T_{TA} &= T \cup \{t_{u_1 t_1}, t_{u_1 t_2}, t_{u_2 t_1}, t_{u_2 t_2}, \dots, t_{u_j t_i}\}, \\
 F_{TA} &= F \cup \{\langle p_{t_1-}, t_{u_1 t_1} \rangle, \langle p_{t_2-}, t_{u_1 t_2} \rangle, \langle p_{t_1-}, t_{u_2 t_1} \rangle, \langle p_{t_2-}, t_{u_2 t_2} \rangle, \dots, \\
 &\quad \langle p_{t_i-}, t_{u_j t_i} \rangle\} \cup \{\langle t_{u_1 t_1}, p_{t_1+} \rangle, \langle t_{u_1 t_2}, p_{t_2+} \rangle, \langle t_{u_2 t_1}, p_{t_1+} \rangle, \langle t_{u_2 t_2}, p_{t_2+} \rangle, \dots, \\
 &\quad \langle t_{u_j t_i}, p_{t_i+} \rangle\} \cup \{\langle p_{t_1+}, t_1 \rangle, \langle p_{t_2+}, t_2 \rangle, \dots, \langle p_{t_i+}, t_i \rangle\} \\
 &\text{and the marking } m_{TA_0} = \langle 1, 0, 0, \dots, 0, 1, 0, 1, 0, \dots, 1, 0 \rangle \text{ with } m_{TA_0} \\
 &\text{according to the order } p_{input}, p_2, \dots, p_{output}, p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}, \dots, \\
 &\quad p_{t_i-}, p_{t_i+}.
 \end{aligned}$$

The five steps from Definition 3.30 are applied to the running example based on its net N and the user-task authorization TA . First, the places p_{t_1-} , p_{t_1+} , p_{t_2-} , and p_{t_2+} are created for the transitions t_1 and t_2 , and p_{t_1-} and p_{t_2-} are marked with one token. Afterwards, the user-task transitions $t_{u_1 t_1}$, $t_{u_2 t_1}$, and $t_{u_1 t_2}$ are created based on the user-task authorization. Then, the arcs $\langle p_{t_1-}, t_{u_1 t_1} \rangle$ and $\langle p_{t_1-}, t_{u_2 t_1} \rangle$ are created for p_{t_1-} and its corresponding transitions $t_{u_1 t_1}$ and $t_{u_2 t_1}$, and the arc $\langle p_{t_2-}, t_{u_1 t_2} \rangle$ is created for p_{t_2-} and its transition $t_{u_1 t_2}$. After that, the arcs $\langle t_{u_1 t_1}, p_{t_1+} \rangle$ and $\langle t_{u_2 t_1}, p_{t_1+} \rangle$ are created for transitions $t_{u_1 t_1}$ and $t_{u_2 t_1}$ and its corresponding place p_{t_1+} , and the arc $\langle t_{u_1 t_2}, p_{t_2+} \rangle$ is created for $t_{u_1 t_2}$ and its corresponding place p_{t_2+} . The idea of the last step is to then connect all the created Petri net parts with the initial WF-net, which is realized by adding arcs from $\langle p_{t_1+}, t_1 \rangle$ to $\langle p_{t_2+}, t_2 \rangle$. That way, TA is flattened into the WF-net N , which results in the net N_{TA} , where $P_{TA} = \{p_1, p_2, p_3, p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}\}$, $T_{TA} = \{t_1, t_2, t_{u_1 t_1}, t_{u_2 t_1}, t_{u_1 t_2}\}$,

$F_{TA} = \{\langle p_1, t_1 \rangle, \langle p_1, t_2 \rangle, \langle t_2, p_3 \rangle, \langle p_{t_1-, t_{u_1 t_1}} \rangle, \langle p_{t_1-, t_{u_2 t_1}} \rangle, \langle p_{t_2-, t_{u_1 t_2}} \rangle, \langle t_{u_1 t_1}, p_{t_1+} \rangle, \langle t_{u_2 t_1}, p_{t_1+} \rangle, \langle t_{u_1 t_2}, p_{t_2+} \rangle\}$ and the marking $m_{TA_0} = \{p_1, p_{t_1-}, p_{t_2-}\}$.

3.2.2.2 Flattening of SoD Constraints

Figure 3.25a depicts the encoding of SoD constraints. The basic idea behind the flattening of SoD constraints is to introduce a choice-place for all users authorized for conflicting tasks (as depicted in Figure 3.23b). Hence, choice-places are added for every user-task transition pair involving the same user for the tasks affected by the constraint. Every such choice-place will then prevent the firing of a user-task transition containing the same user. Moreover, arcs are added from each choice-place to every conflicting user-task transition pair. Note that an SoD choice-place is introduced only for user-task assignments that conflict with each other (see SoD_{u_1} and SoD_{u_2}). Furthermore, the restriction on the sequential execution of t_i and t_j can be dropped, e.g., t_i and t_j can be concurrent (as reflected in Figure 3.25a).

Definition 3.31 (Flattening SoD Constraints). *Given a policy-aware workflow net $N^{pol} = \langle N, TA, C \rangle$, after transforming the user-task authorization TA into N resulting in N_{TA} (according to Definition 3.30), flattening of SoD constraints $c_{SoD} \in C$ of the form (t_k, t_l, \neq) into N_{TA} is as follows:*

1. *For each pair of user-task transitions $t_{u_j t_k}$ and $t_{u_j t_l}$ of each transition t_k and t_l of each SoD constraint (t_k, t_l, \neq) , create a place $SoD_{u_j t_k t_l}$ and mark it with one token. Because there will be no SoD transitions, the usual place notation p that indicates the SoD place $p_{SoD_{u_j t_k t_l}}$ is omitted for better readability.*
2. *For every created place $SoD_{u_j t_k t_l}$ and its corresponding user-task transitions $t_{u_j t_k}$ and $t_{u_j t_l}$, create the arcs $\langle SoD_{u_j t_k t_l}, t_{u_j t_k} \rangle$ and $\langle SoD_{u_j t_k t_l}, t_{u_j t_l} \rangle$.*

After performing these steps, the net $N_{TA+SoD} = \langle P_{TA+SoD}, T_{TA+SoD}, F_{TA+SoD},$

$m_{TA+SoD_0} \rangle$ is obtained, where

$$P_{TA+SoD} = P_{TA} \cup \{SoD_{u_1 t_1 t_2}, SoD_{u_2 t_1 t_2}, \dots, SoD_{u_j t_k t_l}\},$$

$$T_{TA+SoD} = T_{TA},$$

$$F_{TA+SoD} = F_{TA} \cup \{\langle SoD_{u_1 t_1 t_2}, t_{u_1 t_1} \rangle, \langle SoD_{u_1 t_1 t_2}, t_{u_1 t_2} \rangle,$$

$$\langle SoD_{u_2 t_1 t_2}, t_{u_2 t_1} \rangle, \langle SoD_{u_2 t_1 t_2}, t_{u_2 t_2} \rangle, \dots, \langle SoD_{u_j t_k t_l}, t_{u_j t_k} \rangle, \langle SoD_{u_j t_k t_l}, t_{u_j t_l} \rangle\}$$

and the marking $m_{TA+SoD_0} = \langle 1, 0, 0, \dots, 0, 1, 0, 1, 0, \dots, 1, 0, 1, 1, \dots, 1 \rangle$

with m_{TA+SoD_0} according to the order $p_{input}, p_2, \dots, p_{output}, p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}, \dots, p_{t_i-}, p_{t_i+}, SoD_{u_1 t_1 t_2}, SoD_{u_2 t_1 t_2}, \dots, SoD_{u_j t_k t_l}$.

These two steps are exemplified with the SoD constraint (t_1, t_2, \neq) of the running example. First, an SoD place $SoD_{u_1t_1t_2}$ is created for the user-task-transition pair $t_{u_1t_1}$ and $t_{u_1t_2}$, and one token is added to it. Secondly, the arcs $\langle SoD_{u_1t_1t_2}, t_{u_1t_1} \rangle$ and $\langle SoD_{u_1t_1t_2}, t_{u_1t_2} \rangle$ are created to make the SoD place act as a choice place (see Figure 3.23b). Flattening c_{SoD} into the net N_{TA} results in the net N_{TA+SoD} , with $P_{TA+SoD} = \{p_1, p_2, p_3, p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}, SoD_{u_1t_1t_2}\}$, $T_{TA+SoD} = \{t_1, t_2, t_{u_1t_1}, t_{u_2t_1}, t_{u_1t_2}\}$, $F_{TA+SoD} = \{\langle p_1, t_1 \rangle, \langle p_1, t_2 \rangle \langle t_2, p_3 \rangle \langle p_{t_1-}, t_{u_1t_1} \rangle, \langle p_{t_1-}, t_{u_2t_1} \rangle, \langle p_{t_2-}, t_{u_1t_2} \rangle, \langle t_{u_1t_1}, p_{t_1+} \rangle, \langle t_{u_2t_1}, p_{t_1+} \rangle, \langle t_{u_1t_2}, p_{t_2+} \rangle \langle SoD_{u_1t_1t_2}, t_{u_1t_1} \rangle, \langle SoD_{u_1t_1t_2}, t_{u_1t_2} \rangle\}$ and the marking $m_{TA+SoD_0} = \{p_1, p_{t_1-}, p_{t_1+}, SoD_{u_1t_1t_2}\}$. Figure 3.25b depicts this net, which is just the net of Figure 3.23a with additional place annotations.

3.2.2.3 Flattening of BoD Constraints

Fig. 3.26(a) depicts how BoD constraints are encoded. Here, choice places are added as well. However, the arcs are connected differently. The idea is that, for a BoD constraint on two tasks, every choice place aims to prevent the firing of a user-task transition that contains another user. Therefore, for each user-task assignment transition for a task, a choice place is introduced and connected to that user-task transition. Then, from each choice place, arcs are added to every user-task transition for the other task that does not contain the user for which the choice place has been introduced. Note that, just like in SoD flattening, t_i and t_j can be concurrent.

Definition 3.32 (Flattening BoD Constraints). *Given a policy-aware workflow net $N^{pol} = \langle N, U, TA, C \rangle$, after transforming the user-task authorization TA into N resulting in N_{TA} (according to Definition 3.30), flattening of BoD constraints $c_{BoD} \in C$ of the form $(t_k, t_l, =)$ into N_{TA} is as follows:*

1. *For each user-task transition $t_{u_jt_k}$ or $t_{u_jt_l}$ of each transition t_k and t_l of each BoD constraint $(t_k, t_l, =)$, create a place $BoD_{u_jt_kt_l}$ and mark it with one token. If the respected user-task transition is already connected to a BoD place regarding the respective constraint, proceed to the next user-task transition or terminate if there are none left.*
2. *For every created place $BoD_{u_jt_kt_l}$ and its corresponding user-task transition $t_{u_jt_k}$, create an arc $\langle BoD_{u_jt_kt_l}, t_{u_jt_k} \rangle$. For every user-task transition for t_l except $t_{u_jt_l}$, create an arc $\langle BoD_{u_jt_kt_l}, t_{u_l} \rangle$.*

Analogous to N_{TA+SoD} , after performing these steps, the net $N_{TA+BoD} = \langle P_{TA+BoD}, T_{TA+BoD}, F_{TA+BoD}, m_{TA+BoD_0} \rangle$ is obtained, where

$$\begin{aligned}
P_{TA+BoD} &= P_{TA} \cup \{BoD_{u_1t_1t_2}, BoD_{u_2t_1t_2}, \dots, BoD_{u_jt_kt_l}\}, \\
T_{TA+BoD} &= T_{TA}, \\
F_{TA+BoD} &= F_{TA} \cup \\
&\{\langle BoD_{u_1t_1t_2}, t_{u_1t_1} \rangle\} \cup \{\langle BoD_{u_1t_1t_2}, t_{u_1t_2} \rangle | t_{u_1t_2} \in T_{TA} - \{t_{u_1t_2}\} \wedge u \in U \wedge (u, t_2) \in \\
&TA\} \cup \\
&\{\langle BoD_{u_2t_1t_2}, t_{u_2t_1} \rangle\} \cup \{\langle BoD_{u_2t_1t_2}, t_{u_2t_2} \rangle | t_{u_2t_2} \in T_{TA} - \{t_{u_2t_2}\} \wedge u \in U \wedge (u, t_2) \in \\
&TA\} \cup \dots \cup \\
&\{\langle BoD_{u_jt_kt_l}, t_{u_jt_k} \rangle\} \cup \{\langle BoD_{u_jt_kt_l}, t_{u_jt_l} \rangle | t_{u_jt_l} \in T_{TA} - \{t_{u_jt_l}\} \wedge u \in U \wedge (u, t_l) \in \\
&TA\} \\
&\text{and the marking } m_{TA+BoD_0} = \langle 1, 0, 0, \dots, 0, 1, 0, 1, 0, \dots, 1, 0, 1, 1, \dots, 1 \rangle \\
&\text{with } m_{TA+BoD_0} \text{ according to the order } p_{input}, p_2, \dots, p_{output}, p_{t_1-}, p_{t_1+}, \\
&p_{t_2-}, p_{t_2+}, \dots, p_{t_i-}, p_{t_i+}, BoD_{u_1t_1t_2}, BoD_{u_2t_1t_2}, \dots, BoD_{u_jt_kt_l}.
\end{aligned}$$

To illustrate this, suppose that the SoD constraint in the example is replaced by the BoD constraint $(t_1, t_2, =)$, such that the computation of the market value (t_1) and its control (t_2) is supposed to be done by the same user. Applying the first step of Definition 3.32 then creates the place $BoD_{u_1t_1t_2}$ for the user-task transition $t_{u_1t_1}$ and $BoD_{u_2t_1t_2}$ for $t_{u_2t_1}$. Secondly, the arcs $\langle BoD_{u_1t_k1_1}, t_{u_1t_1} \rangle$ are created. Here, the second arc can not be created because there is no other user-task transitions for t_2 than $t_{u_1t_2}$. Then, the arcs $\langle BoD_{u_2t_k1_1}, t_{u_2t_1} \rangle$ and $\langle BoD_{u_2t_k1_1}, t_{u_1t_2} \rangle$ are created. Flattening c_{BoD} into the net N_{TA} results in the net N_{TA+BoD} , with $P_{TA+BoD} = \{p_1, p_2, p_3, p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}, BoD_{u_1t_1t_2}, BoD_{u_2t_1t_2}\}$, $T_{TA+BoD} = \{t_1, t_2, t_{u_1t_1}, t_{u_2t_1}, t_{u_1t_2}\}$, $F_{TA+BoD} = \{\langle p_1, t_1 \rangle, \langle p_1, t_2 \rangle, \langle t_2, p_3 \rangle, \langle p_{t_1-}, t_{u_1t_1} \rangle, \langle p_{t_1-}, t_{u_2t_1} \rangle, \langle p_{t_2-}, t_{u_1t_2} \rangle, \langle t_{u_1t_1}, p_{t_1+} \rangle, \langle t_{u_2t_1}, p_{t_1+} \rangle, \langle t_{u_1t_2}, p_{t_2+} \rangle, \langle BoD_{u_1t_1t_2}, t_{u_1t_1} \rangle, \langle BoD_{u_2t_1t_2}, t_{u_2t_1} \rangle, \langle BoD_{u_2t_1t_2}, t_{u_1t_2} \rangle\}$ and the marking $m_{TA+BoD_0} = \{p_1, p_{t_1-}, p_{t_2-}, BoD_{u_1t_1t_2}, BoD_{u_2t_1t_2}\}$. Figure 3.26b depicts this net graphically.

3.2.2.4 SecANet Obstructions

Based on the SecANet flattening and the observations from the example, the state of obstruction and the affected WF-tasks can now be grasped formally. Obstructions may not only obstruct a single task during the execution of a workflow, as in the example. Depending on the structure of the considered process, for instance, in a parallel branch, multiple tasks may become obstructed at the same time in a single execution as well.

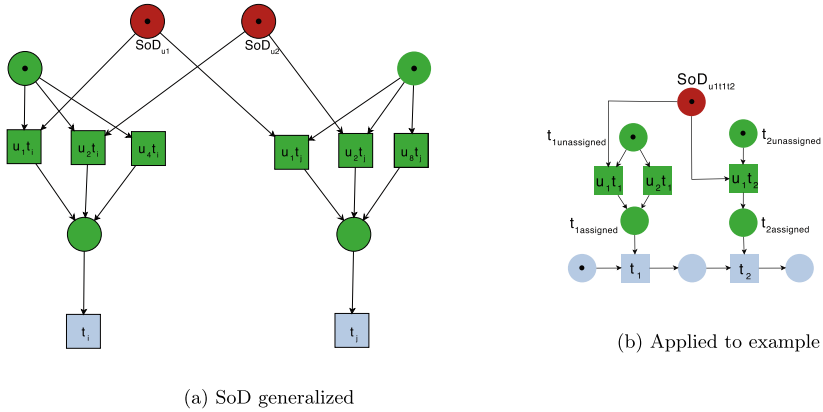


Figure 3.25 Generalized SoD with application to simplified payment workflow

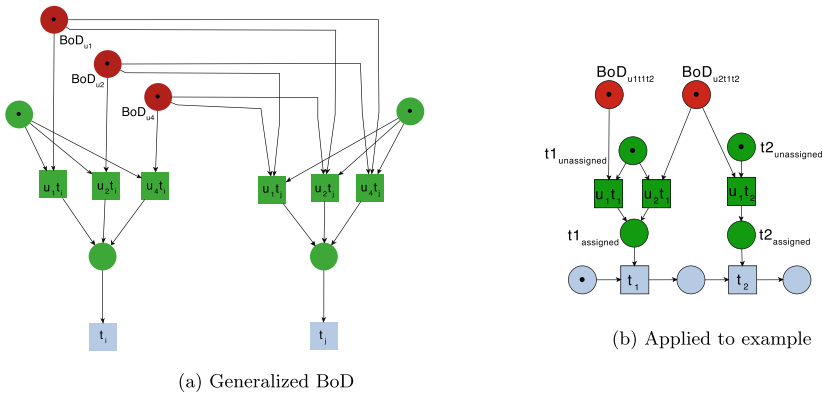


Figure 3.26 Generalized BoD with application to simplified payment workflow

Definition 3.33 (Obstruction Marking, Obstructed Task). Let N_{TA+C} be a SecANet with an initial marking $m_o \in P_{TA+C}$ and the output place $p_o \in P$ of the WF-net N . Based on the set of reachable markings from m_o , an obstruction marking m_\otimes represents a terminal marking that does not contain the output place p_o of the workflow net, i.e., $m_\otimes = m \in [m_o]$ iff $\nexists t \in T_{TA+C} : m[t]$ and $p_o \not\subseteq m$. Accordingly, the set of obstruction markings M_\otimes is represented by all

obstruction markings, i.e., $M_{\otimes} = \{m \in [m_0] \mid \nexists t \in T_{TA+C} : m[t] \wedge p_o \not\subseteq m\}$. Moreover, an obstructed task t_{\otimes} denotes the task of the WF-net that can not be executed due to an obstruction. It represents the post-area of a marked WF-place p^{\bullet} contained in the obstruction marking m_{\otimes} . Accordingly, the set of obstructed tasks for an obstruction marking m_{\otimes} in a workflow is defined as $T_{\otimes} = \{p^{\bullet} \mid p \in m_{\otimes} \wedge p \in P\}$.

For example, the set of obstruction markings of the SecANet in Figure 3.24 is the singleton marking $M_{\otimes} = \{\{p_1, p_{t_2-}\}\}$. The set of obstructed tasks for this obstruction marking is $T_{\otimes} = \{p_1^{\bullet}\} = \{t_2\}$. In contrast to the output place of the workflow determining complete workflow executions, it is not known which of all the other places will indicate an obstructed workflow execution. In this regard, it must be noted that there may be markings that do not represent terminal markings but already obstruct some workflow task(s). For example, there may still be enabled, rather non-relevant user-task transitions for tasks that were not supposed to be executed due to exclusive branching. However, especially regarding loops, one can, in general, not rule out that the firing of some user-task transition may not enable further transitions that eventually allow progression in the execution of the workflow. Therefore, in order to exclude such uncertainties, the more restrictive type of obstruction markings is assumed, i.e., markings in which no further transitions are enabled. Given such an obstructed marking, it can then be identified easily which workflow task(s) t_{\otimes} were affected by an obstruction and which of the fired user-task transitions were not relevant at all.

Due to the observations on satisfiability and obstructability in the example, the SecANet encoding allows defining obstructed and satisfiable workflow executions. On the one hand, transition sequences that reach markings that do not enable further transitions and moreover do not contain the output place of the WF-net represent obstructed workflow executions, or, in other words, obstructed partial plans. On the other hand, transition sequences that reach markings that contain the output place p_0 of the WF-net represent complete workflow executions, or, in other words, satisfiable execution plans.

Definition 3.34 (Obstructed and Satisfiable Firing Sequences). *The set of obstructed firing sequences can be denoted as $\mathcal{T}_{\otimes N_{TA+C}} = \{\sigma \mid (N_{TA+C}, m_0)[\sigma](N_{TA+C}, m') \wedge m' \in [m_0] \wedge \exists t \in T_{TA+C} : m'[t] \wedge p_o \notin m\}$. To adequately consider the case that an end marking may not only contain the place p_0 but there may be further tokens left, which were not consumed due to the given encoding, the system net to determine the set of full firing sequences is $SN_{TA+C} = \{N_{TA+C}, m_o, m_{end} \mid m_{end} \in [m_0] \wedge m_{end} \geq p_o\}$. Hence, the set of full firing sequences of the system SecANet $F_{SN_{TA+C}}$ represents complete satisfiable workflow executions. Based on this, the set of obstructed firing sequences can equivalently be defined by the set of terminal sequences without the set of full firing sequences, i.e.,*

$$\mathcal{T}_{\otimes N_{TA+C}} = \mathcal{T}_{N_{TA+C}} - F_{SN_{TA+C}}.$$

For example, the set of obstructed firing sequences of the SecANet in Figure 3.24 is $\mathcal{T}_{\otimes N_{TA+C}} = \{(t_{u_1 t_1}, t_1)\}$. The set of full firing sequences of the system SecANet example is $F_{SN_{TA+C}} = \{(t_{u_2 t_1}, t_1, t_{u_1 t_2}, t_2)\}$ ⁷. Full firing sequences (or plans) of a SecANet can be used to pre-assign all users to tasks before the actual execution of the WF-task. Such pre-assignment can be used to ensure, for example, that each task of the workflow can be executed and, that way, every potential execution sequence is feasible. However, such plans may encode that each user-task assignment is done only when required directly before the execution of the task (i.e., “on-demand”, so to speak, or comparable to the “lazy” execution). In the latter case, if respective plans reach the end of the workflow, the user-task transitions of the tasks that have not been executed, e.g., due to exclusive branching, would still be enabled by the marking reached by the plan. However, although the marking resulting from a plan may still enable transitions, it always completes the workflow because its output place p_o is marked.

3.2.2.5 Capturing Indicators by Costs

Based on the previously identified need for an indicator-based process security that manifests itself in the requirement to consider costs (cf. Chapter 2), the SecANet representation is supposed to capture costs as well. That way, the basis for a security-sensitive solutions will be laid. Formally, to consider indicators in the SecANet, a cost function $c : T, P \rightarrow \mathbb{R}^+$ can be defined to add costs to the

⁷ Again, both sets of firing sequences consider a “lazy” workflow-oriented execution that pursues to execute WF-tasks in the first place.

nodes of the net (transitions or places). Graphically, such further information can be annotated to the respective net elements (i.e., the respective nodes), such that the SecANet places and transitions can be annotated with the costs. Based on different execution sequences and the costs assigned to the involved transitions, costs would allow to determine the least costly user-task assignment based on a certain utility function. For example, given that there would be multiple paths of a satisfiable firing sequences, based on the cost assigned to the user-task transition involved in these sequences, a least costly sequences (or execution plan) could be determined. Assigning costs to transitions will allow to express costs of user-task assignment decisions. Moreover, determining costs for places could encode the cost to add a token to certain places. Here, for example, a violation of a constraint could be encoded in order to allow for a user-task assignment that allows to complete an obstructed workflow execution. That way, even initially “impossible” or “violating” executions sequences could be considered as long as the associated cost would be lower than the risk of obstruction. That way, the SecANet encoding allows capturing indicators by costs. The consideration of these costs will particularly be relevant for finding solutions to obstructions, which will follow after this chapter.

3.2.3 SecANet Properties

Based on the provided generalization, it is now possible to observe the properties of SecANets. This elaboration will first consider the properties of the nets resulting from the flattening of the authorization policy. Afterwards, it will be regarded how the properties change when further constraints are encoded. Figure 3.25 and Figure 3.26 will serve to illustrate the properties of the respective user-task construct (labelings beginning with $u \dots$) and the different constraint constructs (labelings beginning with $SoD \dots$, $BoD \dots$, respectively).

3.2.3.1 Net Properties of Modeled User-Task Authorization

The modeling intention for SecANets is to obtain **safe** (i.e., 1-bounded) nets. Safeness allows for decidability and staying within polynomial complexity bounds for a large number of Petri net problems [87], which aligns with the desire to construct nets that allow for efficient analyses. Moreover, safeness also addresses the requirement of behavioral integrity. It is strongly related to the condition and event principle, which can be connected to the way how Carl Adam Petri introduced his nets [162]. He decomposed state elements to such an extent that each component can be understood as a condition. Such a condition is fulfilled in some situations and is not in others. A token in a place then indicates that a condition is fulfilled. In

this view, a place can have at most one token, just as a condition cannot be fulfilled more than once. If the firing of a transition resulted in a second token in a place, the condition and event principle would be violated. It would be considered as an error because the examined space of possible states of places as conditions is left (analogously, the division by 0 or the addition of a number with a truth value leave the intuition of division and addition for numbers). A system model that uses the intuitive term “condition” reasonably excludes such situations [173]. For example, in the aforementioned WF-nets, places represent conditions and transitions represent tasks (or events). The condition-event principle represents the guideline in the modeling of the user-task assignments based on an authorization policy as well. Assigning a user to a task conditions that no user has yet been assigned. The state of assignment is determined by the presence or absence of a token in the place indicating unassignment. The presence or absence of a token in the place indicating assignment determines that a user is assigned or not. The assignment, in turn, conditions the execution of the activity of the process, i.e., the task of the WF-net. The meaning of a place with two tokens, which might even be produced by different user-task transitions into the same place, would therefore be unclear and would constitute an error as well. This further implies that the net is plain (i.e., all arc weights are at most one).

Based on these considerations, the safeness of the authorization construct from Definition 3.30 will be examined. First, it can be observed that every user-task transition has the same pre- and post-area. Moreover, each transition introduced is bounded to fire only once due to the incoming place marked with a single token. To limit the maximum number of tokens a place may contain, in particular, to obtain 1-boundedness, the Definition 3.30 mimics the so-called “complementary-place transformation” (see Murata [155]). The complementary-place transformation creates additional net components in such a way that they ensure that the number of tokens in a place may not exceed the desired capacity. It first creates a complementary-place p' for each place p . Subtracting the initial marking of the place p from the desired maximum number of tokens in the place p results in the initial marking of the place p' . Then, for all transitions that are connected to each place p , complementary-arcs that connect the complementary-place p' with the transitions in reverse order are created, i.e., an incoming (outgoing) arc of place p results in an outgoing (incoming) arc for the place p' . That way, the sum of tokens for each pair of place p and its place-complement p' equals the maximum number of tokens for each place p before and after firing the regarded transitions [155].

Hence, to show that each user-task construct itself (without its connection to the workflow task) is safe, its identical counterpart resulting from the application of the complementary-place transformation will subsequently be created. Based on

the fundamental choice construct, the basic net depicted in Figure 3.27 models the choice between an arbitrary number of users to be assigned to a single task. Its place is marked with a single token because only one user-task transition is supposed to be selected. Moreover, based on the considerations on safeness above, the maximum number of tokens (i.e., the place capacity) is supposed to be “1” as well.

Based on this, a complementary-place p_{t+} is first built for each place p_{t-} . Given that p_{t-} is initially marked with a single token, and that this is also the maximum number of tokens p may contain, this results in an initial marking of p_{t-} of zero tokens. Then, for all outgoing transitions connected to place p_{t-} , incoming arcs are created that connect each transition with the complementary place p_{t+} , as depicted in Figure 3.28. Hence, the net resulting from the complementary-place transformation of a basic net that models the choice between different transitions and whose maximal number of tokens is bound to 1 exactly models the task-specific authorization resulting from the steps 1-4 from Definition 3.30. Because the workflow task, which is added in the fifth step of Definition 3.30, only additionally consumes a token of the place indicating assignment p_{t+} , it does not contribute to an increase of tokens at all.

In terms of the integrity of inputs, a user-task authorization modeled this way makes both the meaning of each possible state and the associated events (or transitions) unambiguous, thus ensuring traceability to the original authorization policy and its state in a PAIS. In terms of the integrity of the composition, the additional constructs constitute a further precondition for the transitions of the WF-net. That way, although the added transitions extend the language of the initial WF-net, the order of execution is supposed to be preserved. A closer look at this aspect will be taken in the course of a language decomposition.

Besides safeness, there are further properties of the user-task encoding, which are easy to ascertain based on the provided behavioral and structural definitions related to Petri nets. Each user-task authorization construct for a WF-task (i.e., the parts in Figure 3.25a where the labeling begins with $u \dots$) represents a **state machine** (which implies FC) because each transition has exactly one input place

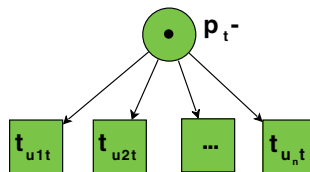


Figure 3.27 Basic choice net for different user assignments for a single task t

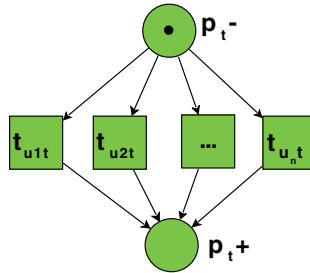


Figure 3.28 Choice net with complement-place p_t+ and -arcs

and one output place. An authorization construct is **plain** (all arc weights are 1) and **pure** (no self-loops), **well-handled** (there are no PT/TP handles for any of the resulting PT or TP pairs). Moreover, the choice construct implies that each transition is enabled for some marking (simply live). Since the encoding introduces further input (or source) places, the overall net is not a WF-net anymore.

3.2.3.2 Net Properties of Modeled Constraints

The modeling of SoD and BoD constraints share the same modeling principle. They both introduce choice places (positioned in the upper area in Figure 3.25 and Figure 3.26) to model mutual exclusion in some way. Therefore, the properties of these nets will be subsumed under the notion of constraint nets.

Analogously to the intention of safeness in the user-task constructs, the choice place that is used for the modeling of constraints indicates whether the constraint affects the user-task assignment or not. Accordingly, the meaning of multiple tokens in such an SoD or BoD place would be unclear as well. For example, two tokens in such a place would directly represent an error, i.e., they would violate the intended constraint since both transitions affected by the choice place would then be fireable, and that way they circumvent the mutual exclusion of the two conflicting transitions. Hence, constraint modeling constructs are supposed to be safe as well. The safeness of the constraint constructs, i.e., a marked choice place connected to several user-task transitions, directly results from the fact that these arcs only consume tokens. Hence, the marking of the initially marked place may not increase. The safeness of constraint constructs connected to some user-task-related constructs results from this observation as well. The modeling of constraints only uses arcs that consume tokens from choice places. They represent a further condition for the affected user-task-transitions to fire. Hence, the overall number of produced tokens does not increase due to the modeling of constraints, i.e., the net construct remains safe.

However, with the introduction of places that connect multiple initially free-choice authorization constructs, the overall regarded construct modeling the policy is not free-choice anymore. Indeed, structurally, some rather simple constructs may model an asymmetric choice, for example, if one considers the BoD place $p_{BoD_{u_1t_1}}$ and the affected user-task construct from Figure 3.26 in isolation. However, each user-task construct affected by a constraint usually then depends on some further places influenced by the firing of transitions beyond the scope of the transitions within a single user-task construct, i.e., they are not free-choice. For instance, the firing of $t_{u_1t_2}$ in the SecANet example reduces the choice between the transitions $t_{u_1t_1}$ and $t_{u_2t_1}$. Hence, the FC-property that, if any output transition of the place p_i — is enabled, all output transitions of that place are enabled, does not hold. Apart from the structural downside, this, is nevertheless just in line with the desired interpretation of the encoded policy elements. Given that there are SoD or BoD constraints, the choice between the users that can be assigned to a task is not free anymore but depends on whether the users are involved in other tasks as well.

Further, since the connection with the WF-net finally allows drawing two simple paths from the constraint place to a respective transition in the WF-net (PT-Handle), the overall net is not well-handled anymore. Although important properties for an efficient computation are remained, e.g., the overall net still represents a safe P/T net, the effect of this modeling on complexity will be considered in the evaluation of the approach. For instance, as indicated before, there are ways to make this construct hold the free-choice property again, whereby, the structural complexity increases to some extent though.

In terms of the integrity of inputs, the constraints are applied to the specific user-task authorizations. Hence, they do not directly allow following and tracing the policy in an explicit way as it was possible in the modeling of the authorization policy with explicit user-task transitions. Rather, it is the case that they manifest themselves in what is not allowed, thereby restricting the possibilities given by the user-task assignment. However, based on the given choice-place, it is at least traceable which constraint has had an impact on the execution. For example, because an SoD place denotes the user and its two conflicting tasks, an empty SoD place encodes that the constraint is in force for the two conflicting user-task assignments. In connection with the fired user-task assignment transition, it is further traceable, which of the users has been assigned to which activity, thereby adhering to the constraint. Hence, the presence or absence of a token in an SoD place traces if the SoD constraint is applied or not. The name of the SoD constraint and its arcs allow retracing tasks affected by it. The user-task assignment itself is then a result of the applied SoD constraint. In this way, the encoding of the SoD constraint preserves its intended behavior.

In terms of the integrity of the overall composition, the impact that results from constraint modeling is observed. Because the constraint restricts given allowed states, what is not allowed can be depicted by comparing the constrained allowed states with what would be allowed if the constraint was not there. With each choice place, the corresponding user task transition is limited, such that the firing sequences reflect the constraint behavior. A closer look at these aspects will be taken in the course of the language-related examination.

3.2.4 Language-Based SecANet Properties

The observation of the full firing sequences of the example SecANet shows that, whereas the authorization policy provides different possibilities to assign users to tasks, constraints restrict these possibilities. In particular, constraints do not add new behavior. Rather, they determine which of the full firing sequences given by the user-task assignment are not allowed anymore. So far, it has seemed that the encoding of authorizations preserves the integrity of the overall representation. It appears that this is also the case for the encoding of constraints. The following examination of language-related properties will deepen these considerations, in particular for proving the behavioral integrity of the SecANet approach.

For this, formal details on the language of Petri nets and useful language-related operators will be introduced first. The SecANet will then be broken down into its smaller increments and reassembled again. This will allow to reason on language-related properties on different subnet levels and to finally determine the overall SecANet language.

3.2.4.1 Language-Related Operators and Nets

The theory of formal languages provides a large number of operations on languages, some of which will be briefly introduced here. The subsequent language-related Petri net definitions were adapted from Priese and Wimmel [167].

To begin with, an alphabet of a language is usually a finite, non-empty set.

Definition 3.35 (Word and Language). A word w over an alphabet Σ consists of a finite (possibly empty) sequence of letters from Σ . The empty word is denoted by ϵ . $|w|$ denotes the number of letters in w , $w(i)$ for $1 \leq i \leq |w|$ stands for the i -th letter of the word w . Furthermore, $\#_a(w)$ is the number of occurrences of the letter $a \in \Sigma$ in the word w . A language over Σ is a set of words over Σ . The set of all words over Σ is denoted by Σ^* ; the empty language by \emptyset . For a language L , the alphabet $\Sigma_L := \{a \mid \exists w \in L \exists i \in \{1, \dots, |w|\} : w(i) = a\}$ denotes the (minimal) alphabet, which consists of all letters that actually occur in at least one word of L . Therefore, for $L \subseteq \Sigma^*$, $\Sigma_L \subseteq \Sigma$ holds. The Parikh image $P(w)$ of a word $w \in \Sigma^*$ is the vector in \mathbb{N}^Σ with $P(w)(a) := \#_a(w)$ for $a \in \Sigma$. The mapping $P : \Sigma^* \rightarrow \mathbb{N}^\Sigma$ with $P(w)$ as the Parikh image of w is also called “Parikh mapping”. The set of all prefixes of a word $w \in \Sigma^*$ is defined as $\text{Pre } w := \{v_1 \in \Sigma^* \mid \exists v_2 \in \Sigma^* : v_1 v_2 = w\}$.

A language containing only the empty word, i.e., $L = \{\epsilon\}$, has the alphabet $\Sigma_L = \emptyset$. Thus, by definition, this Σ_L is not an alphabet at all. For the sake of simplicity, this case is also referred to as the minimal “alphabet” Σ_L for L .

Definition 3.36 (Language Operations). Let L_1 and L_2 be languages of an alphabet Σ . The following operations are defined:

- **Union (\cup):** The union of two languages $L_1 \cup L_2$ is defined as $L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$.
- **Intersection (\cap):** The intersection of two languages $L_1 \cap L_2$ is defined as $L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$. A special case of intersection is the restriction of a language:
- **Language Restriction ($(\cdot)_{\Sigma_0}$):** Let $L \subseteq \Sigma^*$ be a language and Σ_0 an alphabet (usually with $\Sigma_0 \subseteq \Sigma$), then $L|_{\Sigma_0} := L \cap \Sigma_0^*$ ⁸.
- **Concatenation (\cdot):** The concatenation $L_1 \cdot L_2$ or $L_1 L_2$ is defined as $L_1 L_2 := \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$.
- **Kleene-Star ($*$):** The Kleene-Star-Closure of L_1 is given by $L_1^* := \bigcup_{i \geq 0} L_1^i$ with $L_1^0 := \{\epsilon\}$ and $L_1^{i+1} := L_1^i \cdot L_1$.

⁸ The restriction of a language must not be confused with the restriction for two languages, for which a special operator will subsequently be introduced as well.

Moreover, in order to label the Petri net transitions with letters, the definition of a homomorphism will be used.

Definition 3.37 (Homomorphism). *Given that Γ and Σ are alphabets and h is a map $h : \Gamma \rightarrow \Sigma^*$, the homomorphism $\hat{h} : \Gamma^* \rightarrow \Sigma^*$ is inductively given by $\hat{h}(\epsilon) := \epsilon$ and for $w_1, w_2 \in \Gamma^*$: $\hat{h}(w_1w_2) := \hat{h}(w_1) \cdot \hat{h}(w_2)$. Usually, there is no distinction between h and \hat{h} . For a language $L_1 \subseteq \Gamma$, $h(L_1)$ is defined as $h(L_1) := \{h(w) | w \in L_1\}$.*

- A homomorphism $h : \Gamma \rightarrow \Sigma \cup \{\epsilon\}$ is also called a **fine homomorphism**.
- A homomorphism $h : \Gamma \rightarrow \Sigma$ is also called a **very fine homomorphism**.
- A homomorphism $h : \Gamma \rightarrow \Gamma$ with $h(a) = a$ for all $a \in \Gamma$ is called the **identity** on Γ and denoted id_Γ .
- A fine homomorphism $\delta_\Sigma : \Gamma \rightarrow (\Gamma - \Sigma) \cup \{\epsilon\}$ with $\delta_\Sigma(a) = \epsilon$ for all $a \in \Sigma$ and $\delta_\Sigma(a) = a$ for all $a \in \Gamma - \Sigma$ is also called **deletion homomorphism**. If Σ is a singleton, for example $\Sigma = \{a\}$, this can be denoted as δ_a instead of $\delta_{\{a\}}$. Thus δ_Σ “deletes” all symbols in Σ .

A language operator that is particularly interesting for Petri nets is the shuffle operator. It builds a link to reflect the concurrent nature of Petri nets in the sequential nature of word sequences determining a language. It can be visualized by “sticking” two stacks of playing cards together while shuffling. The single cards stand for letters and the two stacks represent two words.

Definition 3.38 (Shuffle (\sqcup)). *The shuffle for two words $w_1 \sqcup w_2$, $w_1, w_2 \in \Sigma^*$ is defined as $w_1 \sqcup \epsilon := \{w_1\}$, $\epsilon \sqcup w_2 := \{w_2\}$, $\forall a, b \in \Sigma : w_1a \sqcup w_2b := (w_1a \sqcup w_2b) \cup (w_1 \sqcup w_2)ba$. The shuffle is canonically extended to languages $L_1, L_2 \subseteq \Sigma^*$ as $L_1 \sqcup L_2 := \bigcup_{w_1 \in L_1, w_2 \in L_2} w_1 \sqcup w_2$.*

For example, for the two languages $\{ab, ad\}$ and $\{ef\}$ is

$$\{ab, ad\} \sqcup \{ef\} = \{abef, aebf, aefb, eabf, eafb, efab, adef, aedf, aefd, eadf, eafd, efad\}.$$

The shuffled words are thus composed of one word from each of the languages involved. While the letters of the same word remain in the given order, the letters of different words in the shuffle can appear in any order. There are no causal dependencies between these words. In this sense, the shuffle \sqcup reflects concurrency that is not directly recognizable in a target word. If the inherently concurrent behavior of Petri nets is modeled using shuffle, one also speaks of interleaving models. Moreover, the shuffle operator now allows the definition of the restriction of two languages.

Definition 3.39 (Restriction Operator (\mathbb{R})). *The restriction operator \mathbb{R} for two languages L_1 and L_2 is defined over (the minimally selected) alphabets Σ_{L_1} and Σ_{L_2} as $L_1 \mathbb{R} L_2 := (L_1 \sqcup (\Sigma_{L_2} - \Sigma_{L_1})^*) \cap (L_2 \sqcup (\Sigma_{L_1} - \Sigma_{L_2})^*)$*

In order to assign a language to a Petri net, it is extended with a mapping that assigns a letter to be generated to each transition. Firing a sequence of transitions creates a word. In this case, the transition system of the marking graph determines the language. In order to increase the expressive power of Petri nets, or, their languages, respectively, final states are often assigned to Petri nets, such that the word belonging to a firing sequence lies in the language of the net only if one of the final markings can be reached at the end of the firing sequence. Because the focus of obstructability lies on process completion, a language-generating Petri net is supposed to consider final markings as well. Moreover, a letter assigned to a transition may also be the empty word ϵ , or the same letter is assigned to multiple transitions. Hence, the labeling of a Petri net may be free (i.e., an identity homomorphism), ϵ -free (i.e., a very fine homomorphism), or arbitrary (i.e., a fine homomorphism, such that multiple transitions may have the same letter assigned to them).

Definition 3.40 (Language-Generating Petri Net). *A language-generating Petri Net N is a 7-tuple $N = \langle P, T, F, m_0, M_f, \Sigma, h_\ell \rangle$, consisting of a Petri net $\langle P, T, F, m_0 \rangle$ with an initial marking m_0 , a finite set of final markings $M_f \subseteq \mathbb{N}^P$, an alphabet Σ and a labeling $h_\ell: T \rightarrow \Sigma \cup \{\epsilon\}$. h_ℓ is continued on T^* , by defining $h_\ell(\epsilon) = \epsilon$ and $h_\ell(\sigma_1 \sigma_2) = h_\ell(\sigma_1) h_\ell(\sigma_2)$, such that h_ℓ is then a homomorphism.*

Every Petri Net $N = \langle P, T, F, m_0 \rangle$ is canonically identified with the language-generating Petri Net $\langle P, T, F, m_0, \emptyset, T, id \rangle$, where the labeling

function is the identity. From now on, the word “language-generating” will be omitted as it is always clear what is meant in each case.

The labeling τ is identified for transitions with the empty word ϵ . Thus, h_ℓ can be extended to a homomorphism in the usual way $h_\ell : T^* \rightarrow \Sigma^*$:

$$\begin{aligned} h_\ell(\epsilon) &:= \epsilon, \\ h_\ell(t) &:= \begin{cases} t, & \text{if } t \in \Sigma - \{\tau\} \\ \epsilon, & \text{if } t = \tau \end{cases} \\ h_\ell(\sigma t) &:= h_\ell(\sigma)h_\ell(t) \end{aligned}$$

A Petri net $N = \langle P, T, F, m_0, M_f, \Sigma, h_\ell \rangle$ in which $h_\ell(t) \neq \epsilon$ for all $t \in T$ is called “ ϵ -free”. The net N wherein $\Sigma = T$ and $h_\ell(t) = t$ for all $t \in T$ is called “free”. For an arbitrary Petri net $N = \langle P, T, F, m_0, M_f, \Sigma, h_\ell \rangle$, $N^f = \langle P, T, F, m_0, M_f, T, id \rangle$ is the free version of N .

τ is just another name for the empty word ϵ . In Petri net theory, τ has become customary. One notion is that transitions labeled with τ are supposed to be invisible to an observer of a Petri net. The symbol λ can often be found for τ as well. In case that Petri nets are used to model business processes, it is not unusual to distinguish between so-called regular transitions (i.e., t labeled with some letter in $\Sigma - \{\tau\}$) and silent transitions (i.e., t labeled with τ). Silent transitions do not represent the execution of a process activity and are used for routing purposes or to achieve a more compact representation of the net. That way, for example, optional process activities can be modeled as a choice between a silent transition and the process activity itself. Moreover, silent transitions may also be introduced in case that a net is transformed in order to obtain desirable net properties. For example, if an arbitrary net is transformed into a free-choice net, it may be describable that the transitions added in the course of the free-choice transformation are silent as well. Silent transitions are not considered in the language determined by (completed) firing sequences.

Based on such a (somehow) labeled Petri net, the set of all possible firing sequences resulting from the execution that starts in an initial marking and terminates in some final marking determines a language. The term “final” must indeed be considered in a more differentiated way: Whereas arbitrary markings can be defined as final markings, the notion of “terminal” markings describes markings given by all states that do not allow the firing of any further transition, namely deadlock markings.

Based on these considerations, the study of Petri nets in formal language theory has introduced several notions of Petri net languages. The standard notion of a Petri net language accepts sequences of transition labels in a run from an initial to a final marking. The prefix language considers all markings to be final. Here, each prefix of a word is itself a word of the language. The covering language accepts sequences leading to markings that are greater or equal to a given set of final markings. Finally, the terminal languages only accept sequences leading to a deadlock [160, 161]:

Definition 3.41 (Petri Net Language). *Let $N = \langle P, T, F, m_0, M_f, \Sigma, h_\ell \rangle$ be a Petri net.*

The language of N accepts firing sequences to the final marking, i.e., the set of final markings is defined by a finite marking set. It is defined as $L(N) = \{w \in \Sigma^ | \exists \sigma \in T^* \exists m_f \in M_f : m_0[\sigma]m_f \wedge h(\sigma) = w\}$.*

The prefix language of N accepts all transition sequences (in other words, all reachable markings are final markings) and is defined as $P(N) = \{w \in \Sigma^ | \exists \sigma \in T^* : m_0[\sigma] \wedge h(\sigma) = w\}$.*

The covering language of N accepts all transition sequences reaching a marking that is greater than or equal to any element of the set of final markings. It is defined as $C(N) = \{w \in \Sigma^ | \exists \sigma \in T^* \exists m_f \in M_f \exists m \in \mathbb{N}^P : m_0[\sigma]m \wedge m \geq m_f \wedge h(\sigma) = w\}$.*

The terminal language N accepts runs to deadlock markings, such that the set of final markings results from any terminal marking (a marking in which no transition is enabled). The terminal language is defined as $T(N) = \{w \in \Sigma^ | \exists \sigma \in T^* \exists m \in \mathbb{N}^P : m_0[\sigma]m \wedge h(\sigma) = w \wedge \nexists t \in T : m[t]\}$.*

Petri net languages represent the interleaving semantic of Petri nets. As indicated before, Petri net languages can model the inherently concurrent behavior of Petri nets with the help of the shuffle operator. It is assumed, however, that narrowing down the view from true concurrency to interleaving behavior is not significant for the consideration of business process activities in the context of this thesis. Indeed, it is unlikely and irrelevant whether parallel tasks will take place at exactly the same time such that they happen in one step (true concurrency). Rather, the point is

that parallel tasks in business processes (or parallel user-task assignments) can be executed independently from each other.

Hence, the three different labeling functions and the four different language types result in twelve different language classes. The focus of this work can be limited to non-prefix-closed languages of predominantly free nets. Hence, the prefix type and the terminal type (which is non-prefix-closed) do not require the definition of a final marking. P-type languages represent the sequence semantics, i.e. each possible fire sequence corresponds to one word of the language of the net. T-type languages stand for the complete sequence semantics, which reduces the language of the net to those words that are created by completed terminal firing sequences. However, both languages can be represented with the help of final markings as well. On the one hand, the prefix language $P(N)$ can be expressed as the covering language of the marking that assigns zero to all places, i.e., $P(P, T, F, m_0, \emptyset, \Sigma, h_\ell) = C(P, T, F, m_0, \mathbf{0}, \Sigma, h_\ell)$, or equivalently, $C(P, T, F, m_0, \emptyset, \Sigma, h_\ell)$. That way, the prefix-closure could also be reproduced by defining each reachable marking as a final state of the language $L = C(P, T, F, m_0, [m_0], \Sigma, h_\ell)$. On the other hand, the terminal language $T(N)$ can be represented as the language L of the set of final markings that contains all terminal markings, that is, $T(P, T, F, m_0, \emptyset, \Sigma, h_\ell) = L(P, T, F, m_0, [m_0]_{\mathcal{T}}, \Sigma, h_\ell)$.

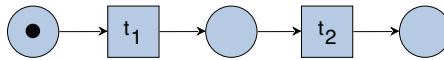


Figure 3.29 Determine Market Value workflow as WF-net with initial marking

These different perspectives are also underlined by observing the different languages of the basic WF-net of the DMV example from Figure 3.29, i.e., $N = \langle P, T, F, m_0, M_f, T, id \rangle$ where $M_f = \{\{p_0\}\}$, $\Sigma = \{t_1, t_2\}$, and $h_\ell(t_1) = t_1$, $h_\ell(t_2) = t_2$. Here, $L(N) = C(N) = T(N) = \{t_1 \cdot t_2\}$ and $P(N) = \{\emptyset, t_1, t_1 \cdot t_2\} = Pre L(N)$. For a different final marking $M_f = \{\{p_2\}\}$ only the L- and C-type languages change, namely $L(N) = \{t_1\}$ and $C(N) = \{t_1, t_1 \cdot t_2\}$. In this regard, Peterson has already recognized that the language types L and T are equally expressive, that is, for every net N , there is a net N' with $L(N) = T(N')$ and vice versa. The types P and C have less expressive power [161].

3.2.4.2 SecANet Language Types

Based on these language-related definitions, the language of a SecANet will now be examined. First of all, in order to investigate language related properties, a labeling

function will assign an alphabet Σ to the transitions of the SecANet. Because the SecANet definitions encode each transition distinctively, the labeling function represents the identity function. Hence, the labeling of a SecANet is free, that means, there are no arbitrary labelings or ϵ -labelings. That way, the language types can be directly related to the set of full and terminal, obstructed and satisfiable firing sequences of a SecANet:

- In analogy to full firing sequences for some workflow net, the language $L(N)$ of the SecANet $N = \{P, T, F, m_0, M_f, T, id_T\}$ contains all words leading to the set of final markings containing p_o , i.e., $M_f = \{m | m \in [m_0] \wedge m \geq p_o\}$.
- Similar to terminal firing sequences, the terminal language $T(N)$ of the SecANet $N = \{P, T, F, m_0, \emptyset, T, id_T\}$ contains all words that lead to deadlock markings.
- Comparable to satisfiable firing sequences, a SecANet is satisfiable in case that the covering language $C(N)$ of the SecANet $N = \{P, T, F, m_0, p_o, T, id_T\}$ is not empty. It contains the words leading to all markings that contain the output place p_o . Words of such a covering language may still enable further transitions as well. Each of the words of the covering language is therefore at least a prefix of or identical with words from the terminal language.
- Analogously to obstructed sequences, $L_{\otimes} = L(\{P, T, F, m_0, M_{\otimes}, T, id_T\})$ is the obstruction language and determines all sequences leading from the initial marking to obstruction markings, i.e., terminal markings that do not contain the output place of the workflow. Here, the obstruction words also represent words of the terminal language that do not relate to any word in the covering language. Therefore, a SecANet contains obstructions if the terminal language $T(N)$ of the net without the elements of the covering language $C(N)$ is not \emptyset , such that $L_{\otimes}(N) = T(N) - C(N)$. A SecANet is obstruction-free if the words of the terminal language represent a subset of the words in the covering language, i.e., all T-type words are also C-type words, such that $T(N) - C(N) = \emptyset$.

In order to obtain an expressive SecANet language that is able to encode satisfiable but also obstructed firing sequences, its language-type is supposed to contain words that reach final markings that are terminal. Because satisfiable words can contain prefixes that are already satisfiable (i.e., words of the covering language), this means no loss of generality, i.e., the terminal words encode all relevant firing sequences. Based on these considerations, the subsequent language observations assume T-type languages. Because L-type languages are able to express all different language-types with a corresponding set of final markings, the terminal SecANet language $T(\text{SecANet})$ is equal to the SecANet language $L(\text{SecANet})$ with the finite marking

set $[m_0]_T$. Hence, the language-generating SecANet that is subsequently considered is a free Petri net of the form $N = \langle P, T, F, m_0, [m_0]_T, T, id_T \rangle$.

However, determining the language of a SecANet already appears to be difficult for the supposedly simple example of the DMV SecANet in Figure 3.25. Thus, to determine the language of a SecANet, further operations that allow to decompose the net such that the language of smaller parts of the net can actually be determined are required. These parts of the language can then be combined again in a further step, whereby the language of the entire net can be determined. In the following, this decomposition and composition will first be illustrated by determining the language of the example SecANet.

3.2.4.3 Decomposition

To obtain the language of a particular Petri net, it will first be decomposed into smaller subnets whose languages are straightforwardly determinable. To do this, the subnet are constructed by a selection of disjoint subsets of places. The place subsets together with their incoming and outgoing transitions will constitute the elements of the resulting subnets. Their initial and final markings will be determined based on the markings of the initial net. If the subnets are small enough, their languages can then be determined easily.

As a preparation for the subsequent definition of a subnet, projections will be useful in order to obtain the initial and final markings of the subnets. They are denoted by π . For a set P , \mathbb{N}^P stands for the set of all mappings $f : P \rightarrow \mathbb{N}$. For finite sets $P = \{p_1, \dots, p_n\}$, \mathbb{N}^P can be identified with $\mathbb{N}^{|P|}$, which denotes the set of all $|P|$ -dimensional column vectors over \mathbb{N} . If p is a distinguished element in P and $P' \subseteq P = \{p_1, \dots, p_n\}$ is a distinguished subset of P , then $\pi_p : \mathbb{N}^P \rightarrow \mathbb{N}$ and $\pi_{p'} : \mathbb{N}^P \rightarrow \mathbb{N}^{P'}$ are the projections with $\pi_p(v) = v(p)$ and $\pi_{p'}(v) = (v(p_{i_1}), \dots, v(p_{i_k}))^{\top 9}$ for a vector v and $P' = \{p_{i_1}, \dots, p_{i_k}\}$ with $i_1 < \dots < i_k$. A vector $v \in \mathbb{N}^{P'}$ is understood as an element of \mathbb{N}^P with $v(p) = 0$ for all $p \in P - P'$, which describes the canonical embedding of $\mathbb{N}^{P'}$ in \mathbb{N}^P .

Definition 3.42 (Subnet). *Given a Petri net $N = \langle P, T, F, m_0, M_f, \Sigma, h_\ell \rangle$ and $P' \subseteq P$, the following can be defined:*

$$m_{0,P'} := \pi_{P'}(m_0), \quad M_{f,P'} := \pi_{P'}(M_f).$$

⁹ \top represents the transpose and allows a row-based notation of the column vector.

The subnet generated by P' is defined as

$N(P') := \langle P', T', F|_{(P' \times T') \cup (T' \times P')}, m_{0,P'}, \Sigma, h_{\ell|_{T'}}, M_{f,P'} \rangle$, where $T' := \bullet P' \cup P' \bullet$. For $P' \subseteq P$ with $N' = N(P')$, N' represents a so-called “closed” subnet of N . Additionally, a net with $|P'| = 1$ is called “elementary”.

Whereas, in the course of the flattening, all inputs were integrated into the WF-net step by step, for a decomposition, it will be necessary to decompose the elements of the flattened net into smaller tuples or subsets. Therefore, a dot-notation, which is supposed to unambiguously indicate which specific set or tuple is regarded, will be introduced for better clarity.

Definition 3.43 (Dot-Notation). *Let Q be some set and s a description of some subset of Q . The dotted-set notation \dot{Q}_s determines the subset of Q consisting only of the elements described with the subscripted s . The dot-notation is canonically extended for tuples such that for some tuple $Y = \langle Q_1, Q_2, \dots, Q_n \rangle$, the dot-notation is continued onto its elements, i.e., $\dot{Y}_s = \langle \dot{Q}_{1s}, \dot{Q}_{2s}, \dots, \dot{Q}_{ns} \rangle$.*

For the example SecANet, the dotted \dot{P}_{TA+SoD} thus determines that this set does only contain the subscripted net elements, e.g., in this case, only $TA + SoD$ places from user-task or SoD-constraint modeling without WF-net places, i.e., $\dot{P}_{TA+SoD} = P_{TA+SoD} - P = \{p_{t1-}, p_{t1+}, p_{t2-}, p_{t2+}, SoD_{u_1t_1t_2}\}$.

Decomposition of the Example SecANet: The decomposition of a free Petri net into subnets is demonstrated by means of the SecANet example depicted in Figure 3.23b. First, the set of places is divided into two disjoint subsets $P = \{p_i, p_2, p_o\}$ and $\dot{P}_{TA+SoD} = \{p_{t1-}, p_{t1+}, p_{t2-}, p_{t2+}, SoD_{u_1t_1t_2}\}$. The closed subnets $N = N(P)$ and $\dot{N}_{TA+SoD} = N(\dot{P}_{TA+SoD})$ are illustrated in Figure 3.30. The initial marking of the two subnets N and \dot{N}_{TA+SoD} is $m_{0,P} = \{p_i\}$ and $m_{0,\dot{P}_{TA+SoD}} = \{p_{t1-}, p_{t2-}, SoD_{u_1t_1t_2}\}$. Because the final markings for the example nets result from the terminal markings of the SecANet $M_f = [m_0]_{\mathcal{T}} = \{\{p_o\}, \{p_2, p_{t2-}\}\}$, the projection of this marking onto the subnets $N(P)$ and

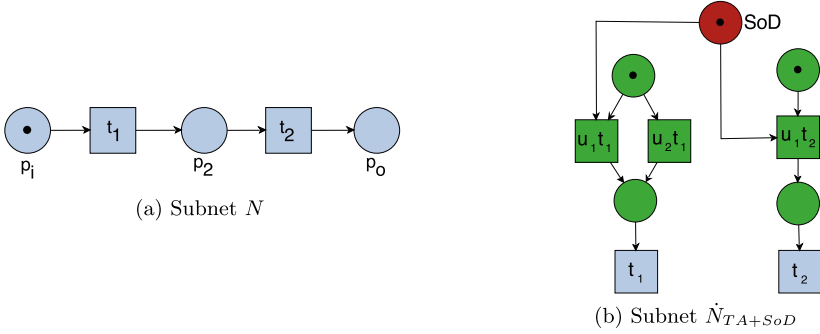


Figure 3.30 Example decomposition of the SecANet example

N_{TA+SoD} is $M_{f,P} = \{\{p_o\}, \{p_2\}\}$, and $M_{f,\dot{P}_{TA+SoD}} = \{\emptyset, \{p_{t_2-}\}\}$, respectively. The language generated by the subnet $N = \{P, T, F, m_0, M_{f,P}, T, id_T\}$ and its alphabet is now easy to determine, namely

$$\begin{aligned} L(N) &= \{t_1 t_2, t_1\} \text{ and} \\ \Sigma_N &= \{t_1, t_2\}. \end{aligned}$$

In order to determine the language of the other subnet $L(\dot{N}_{TA+SoD})$, a further decomposition is necessary. The set of places of \dot{N}_{TA+SoD} is divided into two disjoint subsets $\dot{P}_{TA} = \{p_{t_1-}, p_{t_1+}, p_{t_2-}, p_{t_2+}\}$ and $\dot{P}_{SoD} = \{SoD_{u_1 t_1 t_2}\}$. That way, the closed subnets $\dot{N}_{TA} = N(\dot{P}_{TA})$ and $\dot{N}_{SoD} = N(\dot{P}_{SoD})$ are obtained, as illustrated in Figure 3.31. The projection of the final markings of the subnet $N(\dot{P}_{TA+SoD})$, namely $M_{f,\dot{P}_{TA+SoD}} = \{\emptyset, \{p_{t_2-}\}\}$, onto the subnets $N(\dot{P}_{TA})$, and $N(\dot{P}_{SoD})$ is $M_{f,\dot{P}_{TA}} = \{\emptyset, \{p_{t_2-}\}\}$ and $M_{f,\dot{P}_{SoD}} = \{\emptyset\}$, respectively. The languages and alphabets generated by \dot{N}_{TA} and \dot{N}_{SoD} are

$$\begin{aligned} L(\dot{N}_{TA}) &= \{(t_{u_2 t_1} \cup t_{u_1 t_1}) t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1\} \text{ with} \\ \Sigma_{TA} &= \{t_{u_1 t_1}, t_{u_2 t_1}, t_{u_1 t_2}, t_1, t_2\}, \text{ and} \\ L(\dot{N}_{SoD}) &= \{(t_{u_1 t_1} \cup t_{u_1 t_2}) \text{ with} \\ \Sigma_{SoD} &= \{t_{u_1 t_1}, t_{u_1 t_2}\}, \text{ respectively.} \end{aligned}$$

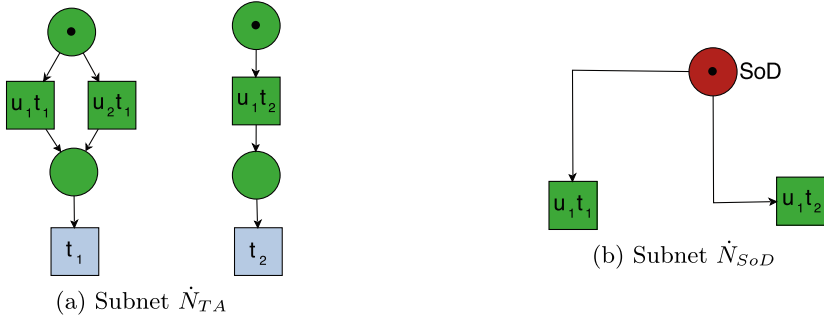


Figure 3.31 Example decomposition of N_{TA+SoD}

Note that, $(t_{u_1 t_1} \cup t_{u_1 t_2})$, for example, represents the union of two disjoint sets (or symmetric difference) and stands for the logical exclusive-or (XOR) for sets. It results in the SoD subnet language $\{t_{u_1 t_1}, t_{u_1 t_2}\}$, which, thus, directly encodes the exclusive choice between two conflicting user-task transitions.

3.2.4.4 Composition

Based on the decomposition, it can now be considered how to determine the language of a free Petri net if the languages of its subnets are known. Apparently, in order to assemble the subnets, transitions that occur in several subnets must be “merged” in some way. This points to the use of a further operation in the theory of formal languages, namely synchronization. Indeed, a couple of operations such as intersection, shuffle and restriction are simple special cases of synchronization, and synchronization is a quite natural operation for Petri nets. Unfortunately, as a language operation, it is all the more complicated [167]:

Definition 3.44 (Synchronization of Languages). Let Σ_1, Σ_2 be two alphabets, Σ a finite set and $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ two languages. The Σ -Synchronization $L_1 \text{ sy}_\Sigma L_2$ of L_1 and L_2 is defined by

$$L_1 \text{ sy}_\Sigma L_2 := g((h(L_1) \sqcup (\Sigma_2 - \Sigma)^*) \cap (L_2 \sqcup h((\Sigma_1 - \Sigma)^*))),$$

where $h: \Sigma_1^* \rightarrow (\Sigma'_1 \cup \Sigma)^*$ is the labeling with $h(a) := a$ for $a \in \Sigma$ and $h(a) := a'$ for $a \in \Sigma_1 - \Sigma$, where Σ'_1 is the alphabet $\Sigma'_1 := \{a' \mid a \in \Sigma_1\}$,

and $g : (\Sigma_2 \cup (\Sigma_1 - \Sigma))^* \rightarrow (\Sigma_1 \cup \Sigma_2 \cup \Sigma)^*$ is the labeling with $g(a') := a$ for $a' \in \Sigma_1'$ and $g(a) := a$ for $a \in \Sigma_2$. Σ can also be denoted as the synchronization alphabet and may—in contrast to usual alphabets—also be empty.

Note that, for an empty synchronization alphabet, it is directly apparent that

$$\begin{aligned} L_1 s_{y\emptyset} L_2 &= g((h(L_1)\Sigma_2^*) \cap (L_2\Sigma'^*)) \\ &= L_1 \sqcup L_2. \end{aligned}$$

Because the restriction is just defined as $L_1 \textcircled{R} L_2 = (L_1(\Sigma_2 - \Sigma_1)^*) \cap (L_2(\Sigma_1 - \Sigma_2)^*)$ according to Definition 3.39, this means that:

$$\begin{aligned} L_1 s_{y\Sigma_1 \cap \Sigma_2} L_2 &= g((h(L_1) \sqcup (\Sigma_2 - (\Sigma_1 \cap \Sigma_2)))^* \\ &\quad \cap (L_2 \sqcup h(\Sigma_1 - (\Sigma_1 \cap \Sigma_2)))^*) \\ &= g((h(L_1) \sqcup (\Sigma_2 - \Sigma_1))^* \cap (L_2 \sqcup h(\Sigma_1 - \Sigma_2))^*) \\ &\stackrel{(*)}{=} (L_1 \sqcup (\Sigma_2 - \Sigma_1)^*) \cap (L_2 \sqcup (\Sigma_1 - \Sigma_2)^*) \\ &= L_1 \textcircled{R} L_2. \end{aligned}$$

(*) can be observed straightforwardly because h renames the letters in $\Sigma_2 - \Sigma_1$, i.e., exactly those letters that do not exist in L_2 anyway. So h and g can also be dropped [167].

The synchronization of languages can canonically be extended to Petri nets. It stands for the counterpart of the decomposition into subnets and allows to combine nets and other languages. While subnets are determined by means of a selection of a set of disjoint places, the synchronization of nets aims at combining transitions labeled identically. The definition of the synchronization of nets by Priese and Wimmel [167] will subsequently be adapted to the definitions provided in this thesis.

Definition 3.45 (Synchronization of Nets). Let $N_i = \langle P_i, T_i, \mathcal{F}_i, m_{o_i}, M_{f,i}, \Sigma_i, h_{\ell_i} \rangle$ for $i \in \{1, 2\}$ be two Petri nets and Σ a (possibly empty) synchronization alphabet ($\tau \notin \Sigma$). Without losing

generality, it is assumed that the nets are disjoint. The Σ -Synchronization of the nets N_1 and N_2 is defined as

$N_1 \text{ sy}_\Sigma N_2 := (P_1 \dot{\cup} P_2, \hat{T}_1 \cup T^\times \cup \hat{T}_2, \hat{F}_1 \cup \mathcal{F}^\times \cup \hat{F}_2, (m_1, m_2), \Sigma_1 \cup \Sigma_2, h_\ell, M_{f,1} \times M_{f,2})$ with the following components:

$\hat{T}_i := \{t \in T_i \mid h_{\ell_i}(t) \notin \Sigma\}$ for $i \in \{1, 2\}$,

$T^\times := \{(t_1, t_2) \in T_1 \times T_2 \mid h_{\ell_1}(t_1) = h_{\ell_2}(t_2) \in \Sigma\}$,

$\hat{F}_i = \mathcal{F}_i - \{\langle t, p \rangle, \langle p, t \rangle \in \mathcal{F}_i \mid t \in T_i \wedge h_{\ell_i}(t) \in \Sigma\}$,

$\mathcal{F}^\times = \{(\langle (t_1, t_2), p \rangle \mid (t_1, t_2) \in T^\times \wedge (\langle t_1, p \rangle \in \mathcal{F}_1 \vee \langle t_2, p \rangle \in \mathcal{F}_2)) \cup \{\langle p, (t_1, t_2) \rangle \mid (t_1, t_2) \in T^\times \wedge (\langle p, t_1 \rangle \in \mathcal{F}_1 \vee \langle p, t_2 \rangle \in \mathcal{F}_2)\}$.

Figure 3.32 depicts the Σ -synchronization of \dot{N}_{TA} and \dot{N}_{SoD} of the example net with the synchronization alphabet $\Sigma = \{t_{u_1 t_1}, t_{u_1 t_2}\}$. It illustrates how the synchronization takes place via the transitions whose labels lie within the synchronization alphabet Σ . Each single transition of two nets N_1 and N_2 with the same labeling of Σ is combined (or “merged”) with each other, including its former pre- and post-areas. The resulting transitions are in T^\times . All other transitions and places are simply transferred to the new net. Note that $L(N_1) \text{ sy}_\Sigma L(N_2) = L(N_1 \text{ sy}_\Sigma N_2)$ [167].

For the synchronization of two subnets $N(P_1)$ and $N(P_2)$ resulting from the decomposition of a free Petri net (without isolated transitions) $N = \langle P, T, F, m_0, M_f, T, id \rangle$ with $P_1 \cap P_2 = \emptyset$ and $P_1 \cup P_2 = P$, and the synchronization alphabet representing the common transitions of the two subnets $\Sigma = T_{N(P_1)} \cap T_{N(P_2)}$, the following applies (for a detailed proof see Priese et al. [167]): If the final state set of N

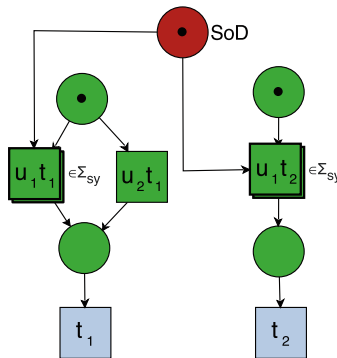


Figure 3.32 $\dot{N}_{TA} \text{ sy}_{\{t_{u_1 t_1}, t_{u_1 t_2}\}} \dot{N}_{SoD}$ (transitions with a label from the synchronization alphabet are synchronized, all other transitions are simply taken over)

is just the product of the two final state sets of the subnets, i.e., $M_f = M_{f,1} \times M_{f,2}$, the synchronization of the two subnets just constitutes the original net again, i.e., $N = N(P_1)_{s,y_\Sigma} N(P_2)$. Then, the synchronization $N(P_1)_{s,y_\Sigma} N(P_2)$ is identical to N , except for the negligible renaming of the transition names. Accordingly, the synchronization of the language of the subnets then determines the language of the original net. More precisely, this represents the restriction because, as described above, $L_1_{s,y_{\Sigma_1 \cap \Sigma_2}} L_2 = L_1 \textcircled{R} L_2$. This does not only apply to free Petri nets that have the same final marking as the product of the final marking sets of the subnets. Indeed, for the general case of such a free Petri net that is decomposed, its language can be composed by applying the restriction operator onto the languages of the subnets. Priese and Wimmel show this by transforming the net and its subnets into a normal form that encloses each (sub)net. This normal form transforms all final states of a net into a single final state. That way, it is achieved that the product of the two subnets in normal form (see Definition 3.45) is the final marking of the original net such that the synchronization of the subnets results in the initial net again, i.e., $N = N(P_1)_{s,y_\Sigma} N(P_2)$. A special case of this restriction is represented by the case in which the languages of two nets N_1 and N_2 do not have any common letters, such that the synchronization alphabet of the languages of two nets is empty. Because $L_1_{s,y_\emptyset} L_2 = L_1 \sqcup L_2$, the shuffle can then be used to combine the two languages, such that, for $\Sigma_{L(N_1)} \cap \Sigma_{L(N_2)} = \emptyset$, $L(N_1) \textcircled{R} L(N_2) = L(N_1) \sqcup L(N_2)$ ¹⁰.

Hence, the restriction operator can be applied if the synchronization alphabet of the languages of the two nets involved in the synchronization represents the intersection of both alphabets, which will be denoted by Σ_\cap . That way, the restriction provides an interface that is able to combine the common transitions that occur in the alphabets of both languages. In case there are multiple subnets from a decomposition of a free Petri net, the composition of the languages of two such subnets results in the language of a further subnet. In this case, the languages of the initial net will need to be obtained by a step-by-step composition of the subnet languages. Depending on whether Σ_\cap is empty for all such compositions, the restriction or the shuffle operator needs to be applied. The subsequent definition bases on the presented findings on the synchronization of free (sub)nets and provides a compact notation for the synchronization applied on the languages of a set of subnets.

¹⁰ This synchronization with an empty synchronization alphabet is similar to the binary merge net operator \parallel , which juxtaposes two marked and labeled Petri nets with disjoint nodes [99].

Definition 3.46 (Composition of Subnet Languages). Given a free Petri net (without isolated transitions) $N(P)$ with $P = \{p_1, \dots, p_{|P|}\}$ and some positive integer n of disjoint subsets $P' \in \{P'_1, \dots, P'_n\}$ with $P'_1 \cup \dots \cup P'_n = P$ and $P'_1 \cap \dots \cap P'_n = \emptyset$ for all $P' \subseteq P$, the subsequent notation describes the synchronization of the languages of all subnets $L(N(P'))$ to compose the language of the net $L(N(P))$. Based on the synchronization alphabet $\Sigma_\cap = \Sigma_{L_1} \cap \Sigma_{L_2}$ that represents the intersection of the alphabets of the two languages L_1 and L_2 involved in the synchronization, two cases may occur:

(1) For $|\Sigma_\cap| > 0$ (in all synchronizations):

$$\begin{aligned} L(N(P)) &= L(N(P'_1))_{sy_{\Sigma_\cap}} L(N(P'_2))_{sy_{\Sigma_\cap}} \dots_{sy_{\Sigma_\cap}} L(N(P'_n)) \\ &= L(N(P'_1)) \textcircled{\text{R}} L(N(P'_2)) \textcircled{\text{R}} \dots \textcircled{\text{R}} L(N(P'_n)) \\ &= \textcircled{\text{R}}_{i=1}^n L(N(P'_i)) \end{aligned}$$

(2) For $\Sigma_\cap = \emptyset$ (in all synchronizations):

$$\begin{aligned} L(N(P)) &= L(N(P'_1))_{sy_\emptyset} L(N(P'_2))_{sy_\emptyset} \dots_{sy_\emptyset} L(N(P'_n)) \\ &= L(N(P'_1)) \sqcup L(N(P'_2)) \sqcup \dots \sqcup L(N(P'_n)) \\ &= \sqcup_{i=1}^n L(N(P'_i)) \end{aligned}$$

The synchronization only considers the intersection of the alphabets of the involved languages, which means that, based on the commutativity of the intersection operator, the order of how such a synchronization is applied does not matter in the end. Note that after the restriction has been applied on two languages, e.g. $L_1 \textcircled{\text{R}} L_2$, a restriction with a further language, for example $L_1 \textcircled{\text{R}} L_2 \textcircled{\text{R}} L_3$, then considers the alphabet of this further language, i.e., Σ_{L_3} , as well as the alphabet of the language resulting from the first restriction, i.e., $\Sigma_{L_1 \textcircled{\text{R}} L_2}$ (and not only Σ_{L_2}).

Composition of the Example SecANet: Because the subnets resulting from the decomposition of the example SecANet above are free as well, the restriction operator will be used to determine the languages of the compositions of the subnets. More specifically, the language of the DMV example SecANet will be composed step by step in reverse order to the order of the decomposition. Therefore, the languages of the two subnets N_{TA} and N_{SoD} will be synchronized first. In a further step, the

combination of these languages will be synchronized with the initial WF-net N . That way, the overall language of the SecANet example will be determined. For the composition of the language of \dot{N}_{TA} and \dot{N}_{SoD} , the synchronization alphabet is $\Sigma_{TA} \cap \Sigma_{SoD} = \{t_{u_1 t_1}, t_{u_1 t_2}\}$. This underlines the fact that the previously indicated special case of the synchronization, namely the restriction, is at hand, such that $L(N_{TA}) \text{sy}_{\Sigma_{TA} \cap \Sigma_{SoD}} L(N_{SoD}) = L(N_{TA}) \textcircled{R} L(N_{SoD})$ [167]. Hence, to obtain the language of the two policy-related subnets of the example, the restriction operator for two languages can directly be applied:

$$\begin{aligned}
L_{TA} \textcircled{R} L_{SoD} &= ((L_{TA} \sqcup (\Sigma_{SoD} - \Sigma_{TA})^*) \cap (L_{SoD} \sqcup (\Sigma_{TA} - \Sigma_{SoD})^*)) \\
&= (((t_{u_2 t_1} \cup t_{u_1 t_1}) t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1) \sqcup (\Sigma_{SoD} - \Sigma_{TA})^*) \\
&\quad \cap ((t_{u_1 t_1} \cup t_{u_1 t_2}) \sqcup (\Sigma_{TA} - \Sigma_{SoD})^*) \\
&= (((t_{u_2 t_1} \cup t_{u_1 t_1}) t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1) \sqcup (\emptyset)^*) \\
&\quad \cap ((t_{u_1 t_1} \cup t_{u_1 t_2}) \sqcup (\{t_{u_2 t_1}, t_1, t_2\})^*) \\
&= (((\{t_{u_2 t_1}, t_{u_1 t_1}\}) t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1) \\
&\quad \cap (\{t_{u_1 t_1}, t_{u_1 t_2}\}) \sqcup (\{t_{u_2 t_1}, t_1, t_2\})^*) \\
&= (((\{t_{u_2 t_1} t_1, t_{u_1 t_1} t_1\}) \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1) \\
&\quad \cap (\{t_{u_1 t_1}, t_{u_1 t_2}\}) \sqcup (\{t_{u_2 t_1}, t_1, t_2\})^*) \\
&= (((\{t_{u_2 t_1} t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1\}) \\
&\quad \cap (\{t_{u_1 t_1}, t_{u_1 t_2}\}) \sqcup (\{t_{u_2 t_1}, t_1, t_2\})^*) \\
&= \{t_{u_2 t_1} t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1\}.
\end{aligned}$$

For the composition of the language of this subnet with the language of the WF-net, i.e., $L_N \text{sy}_{\Sigma_N \cap \Sigma_{TA+SoD}} L_{TA+SoD}$, the synchronization alphabet is $\Sigma_N \cap \Sigma_{TA+SoD} = \{t_1, t_2\}$. Hence, the synchronization can be conducted by using the restriction operator again:

$$\begin{aligned}
L_N \textcircled{R} L_{TA+SoD} &= L(N) \text{sy}_{\Sigma} \cap \Sigma_{TA+SoD} L(\dot{N}_{TA+SoD}) \\
&= L(N) \text{sy}_{\{t_1, t_2\}} L(\dot{N}_{TA+SoD}) \\
&= ((L \sqcup (\Sigma_{TA+SoD} - \Sigma)^*) \\
&\quad \cap (L_{TA+SoD} \sqcup (\Sigma - \Sigma_{TA+SoD})^*)) \\
&= ((\{t_1 t_2, t_1\} \sqcup (\{t_{u_1 t_1}, t_{u_2 t_1}, t_{u_1 t_2}\})^*) \\
&\quad \cap (\{t_{u_2 t_1} t_1 \sqcup t_{u_1 t_2} t_2, t_{u_1 t_1} t_1\} \sqcup (\emptyset)^*)) \\
&= ((\{t_1 t_2, t_1\} \sqcup (\{t_{u_1 t_1}, t_{u_2 t_1}, t_{u_1 t_2}\})^*)
\end{aligned}$$

$$\begin{aligned}
 & \cap \{t_{u_2t_1}t_1 \sqcup t_{u_1t_2}t_2, t_{u_1t_1}t_1\} \\
 = & \{(t_{u_2t_1}t_1 \sqcup t_{u_1t_2}t_2), t_{u_1t_1}t_1\}
 \end{aligned}$$

Thus, the words that correspond to the set of full firing sequences (or plans) of the example SecANet are given by the set $\{(t_{u_2t_1}t_1 \sqcup t_{u_1t_2}t_2), t_{u_1t_1}t_1\}$. The set of obstructed firing sequences (or partial plans) relates to the remaining word that does not contain the end activity t_2 , namely $\{t_{u_1t_1}t_1\}$.

3.2.4.5 SecANet Language

In order to determine the SecANet language, the decomposition and the composition of the example SecANet will be generalized. Figure 3.33 depicts the big picture of the SecANet decomposition, in particular the subnets that are to be separated, namely, the WF-net, the user-task subnets and the constraint subnets (i.e., the SoD and BoD subnets). The following considerations will go through this decomposition

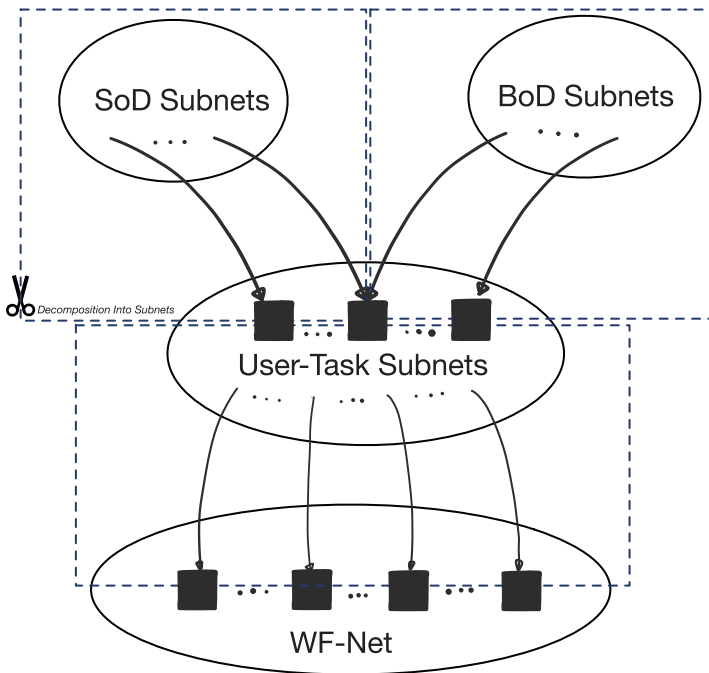


Figure 3.33 SecANet decomposition

step by step in order to determine the essential place subsets in a comprehensive way.

Given a SecANet N_{TA+C} , its disjoint place subsets P and \dot{P}_{TA+C} allow to determine the WF-net N and the net modeling the policy \dot{N}_{TA+C} . The subnet \dot{N}_{TA+C} contains the disjoint place subsets \dot{P}_{TA} and \dot{P}_C , which allow to obtain the subnets \dot{N}_{TA} and \dot{N}_C . Based on the disjoint place subsets $\dot{P}_{C_{SoD}}$ and $\dot{P}_{C_{BoD}}$, the subnet \dot{N}_C can be decomposed into the subnets $\dot{N}_{C_{SoD}}$ and $\dot{N}_{C_{BoD}}$. This results in the four basic disjoint place subsets P , \dot{P}_{TA} , $\dot{P}_{C_{SoD}}$, and $\dot{P}_{C_{BoD}}$, which will determine the subnets N , \dot{N}_{TA} , $\dot{N}_{C_{SoD}}$, and $\dot{N}_{C_{BoD}}$, respectively. The considered place subsets can easily be identified based on the previous SecANet definitions, namely

$$\begin{aligned}
 P &= & P & \quad (\text{the places of the initial WF-net}), \\
 \dot{P}_{TA} &= & P_{TA} - P & \quad (\text{the places of the user-task subnets}), \\
 \dot{P}_{C_{SoD}} &= & P_{TA+SoD} - P_{TA} & \quad (\text{the places of the SoD subnets}), \text{ and} \\
 \dot{P}_{C_{BoD}} &= & P_{TA+BoD} - P_{TA} & \quad (\text{the places of the BoD subnets}).
 \end{aligned}$$

The policy-related subnets will then be decomposed further into basic subnets that allow to determine their languages straightforwardly. The language of each subnet $N(P')$ will be obtained according to the initial marking $m_{0,P'}$ and the set of final markings $M_{f,P'}$ given by the projection of the initial marking and the terminal markings of the SecANet onto the places P' of the considered subnet, that is, $m_{0,P'} = \pi_{P'}(m_0)$ and $M_{f,P'} = \pi_{P'}(\{m_0\}_T)$, respectively. The listing below provides an overview of the subnets and languages resulting from the subsequent decomposition and composition.

- Workflow Subnet N
 - Language of Workflow Subnet L_N
- User-Task Subnets \dot{N}_{TA}
 - User-Task Subnet N_{t-+}
 - Language of the User-Task Subnet L_{t-+}
- SoD Subnets $\dot{N}_{C_{SoD}}$
 - SoD Subnet N_{\neq}
- BoD Subnets $\dot{N}_{C_{BoD}}$
 - BoD Subnet $N_{=}$
- Constraint Subnet N_C
 - Language of Constraint Subnet L_C
- Composition of Languages
 - Language of User-Task Subnets L_{TA}

- Language of Constraint Subnet L_C
- Language of Policy Subnets L_{TA+C}
- Language of SecANet L_{N+TA+C}

After the languages of the smallest subnets N_{I-+} and N_C are determined, they will be combined with the languages of the other subnets step by step in the course of the composition of languages. Through this step-by-step composition of the languages of all subnets in reverse order to the decomposition, the language of the SecANet will finally be obtained.

To begin with, the workflow subnet can be determined through the places of the initial WF-net. Although structurally, it is equivalent to the initial WF-net, the final markings may differ.

Definition 3.47 (Workflow Subnet). *Based on a SecANet $N_{\text{SecANet}} = \langle P_{TA+C}, T_{TA+C}, F_{TA+C}, m_0, [m_0]_{\mathcal{T}}, T_{TA+C}, \text{id} \rangle$ with $m_o \in P_{TA+C}$, the workflow subnet*

$N = N(P) = \langle P, T, \mathcal{F}, m_{0,P}, M_{f,P}, T, \text{id} \rangle$ *is obtained, where*

$$P = \{p_i, p_1, \dots, p_o\},$$

$$T = \{t_1, t_2, \dots, t_{|T|}\},$$

$$F = \{\langle p_i, t_1 \rangle, \langle t_1, p_1 \rangle, \dots, \langle t_{|T|}, p_o \rangle\},$$

$$m_{0,P} = \pi_P(m_0) \text{ and } M_{f,P} = \pi_P([m_0]_{\mathcal{T}}).$$

For example, the initial and final markings of the workflow subnet N in Figure 3.30a is $m_{0,P} = \{p_i\}$ and their set of final markings is $M_{f,P} = \{\{p_o\}, \{p_2\}\}$.

Since the SecANet formalism is given for arbitrary well-structured free-choice WF-nets, whose comprehensive structure can involve parallelism or exclusive paths, the language of such WF-net can only be defined in general terms. However, the SecANet encoding allows to draw conclusions on the alphabet of the language of the workflow subnet. In fact, the language of the initial WF-net can differ from the language of the WF-net that is decomposed from the SecANet because their final markings may be different. On the one hand, if the workflow is satisfiable, the set of final markings of the workflow subnet still contains $\{p_o\}$ (as it was the case for the initial WF-net). On the other hand, in case of obstructions, the set of final markings of the workflow subnet may not (only) contain the output place anymore but the markings of the places $p \in P$ that are in the set of obstruction markings $m \in M_{\otimes}$. Hence, since the alphabet of a language consists of all letters that actually occur in at least one word of the language, the actual alphabet of the workflow subnet

language determined by the terminal markings of the SecANet can differ from the alphabet of the language of the initial net.

In case of a satisfiable execution, the alphabet of the language of the WF-net contains all workflow tasks. However, in case the workflow is not satisfiable, it does not contain the final marking set $\{p_o\}$ and therefore not all workflow tasks of the workflow net are reflected in its language Σ_N . In workflow nets with exclusive branches, it may moreover be possible that only one path is satisfiable, whereas another is not. This would also result in an alphabet Σ_N of the language of the net N that would not contain all tasks either.

Definition 3.48 (Language of the Workflow Subnet). *Based on the language definition applied on the workflow subnet, the language of a workflow net is $L_N = L(N(P)) = \{w \in T^* \mid \exists \sigma \in T^* \exists m_f \in M_{f,P} : m_0, P[\sigma]m_f \wedge h(\sigma) = w\}$.*

The alphabet of $L(N(P))$ is then:

$$\Sigma_N = \{a \in T^* \mid \exists m_f \in M_{f,P} : m_f = \{p_o\} \wedge a \in \{t \mid \forall t \in T : m_0[t_1 t_2 \dots t_n]m_f\} \vee m_f \in \{P - p_o\} \wedge a \in \{t \mid \forall t \in T : m_0[t_1 t_2 \dots t_n]m_f \wedge t_i \notin {}^*p_o\}\}.$$

Hence, in case of an unsatisfiable net (or path), the alphabet of the language of the workflow subnet Σ_N may be a subset of the alphabet Σ defined in the definition of the language-generating workflow subnet. This observation will also be key for the alphabets of the languages for the user-task subnets since, as illustrated in Figure 3.33, a user-task subnet contains the tasks of the workflow subnet as well.

Definition 3.49 (User-Task Subnets). *Based on the flattening of user-task authorizations in Definition 3.30, the subnet*

$\dot{N}_{TA} = N(\dot{P}_{TA}) = \langle \dot{P}_{TA}, \dot{T}_{TA}, \dot{\mathcal{F}}_{TA}, m_0, \dot{P}_{TA}, M_{f, \dot{P}_{TA}}, \dot{T}_{TA}, \text{id} \rangle$ *is obtained, where*

$$\dot{P}_{TA} = \{p_{t_1-}, p_{t_2-}, \dots, p_{t_i-}\} \cup \{p_{t_1+}, p_{t_2+}, \dots, p_{t_i+}\},$$

$$\dot{T}_{TA} = T_{TA} = T \cup \{t_{u_1 t_1}, t_{u_1 t_2}, t_{u_2 t_1}, t_{u_2 t_2}, \dots, t_{u_j t_i}\},$$

$$\dot{\mathcal{F}}_{TA} = \{\langle p_{t_1-}, t_{u_1 t_1} \rangle, \langle p_{t_2-}, t_{u_1 t_2} \rangle, \langle p_{t_1-}, t_{u_2 t_1} \rangle, \langle p_{t_2-}, t_{u_2 t_2} \rangle, \dots, \langle p_{t_i-}, t_{u_j t_i} \rangle\} \cup \{\langle t_{u_1 t_1}, p_{t_1+} \rangle, \langle t_{u_1 t_2}, p_{t_2+} \rangle, \langle t_{u_2 t_1}, p_{t_1+} \rangle, \langle t_{u_2 t_2}, p_{t_2+} \rangle, \dots, \langle t_{u_j t_i}, p_{t_i+} \rangle\} \cup \{\langle p_{t_1+}, t_1 \rangle, \langle p_{t_2+}, t_2 \rangle, \dots, \langle p_{t_i+}, t_i \rangle\},$$

$$m_0, \dot{P}_{TA} = \pi_{\dot{P}_{TA}}(m_0) \text{ and } M_{f, \dot{P}_{TA}} = \pi_{\dot{P}_{TA}}(\{m_0\}_T).$$

For instance, the initial markings and final markings of the user-task subnets \dot{N}_{TA} in Figure 3.31a is $m_{0, \dot{P}_{TA}} = \{p_{t_1-}, p_{t_2-}\}$ and its set of final markings is $M_{f, \dot{P}_{TA}} = \{\{p_{t_2-}\}, \emptyset\}$. The language of each user-task authorization can now be obtained by building the basic subset for each pair of assigned and unassigned places for a single workflow task.

Definition 3.50 (User-task subnet). For each set of pairs $P_{t-+} = \{p_{t-}, p_{t+}\}$, where $t \in T$ and $P_{t-+} \subseteq \dot{P}_{TA}$, the subnet

$N_{t-+} = N(P_{t-+}) = \langle P_{t-+}, T_{P_{t-+}}, \mathcal{F}_{P_{t-+}}, m_{0, P_{t-+}}, M_{f, P_{t-+}}, T_{P_{t-+}}, \text{id} \rangle$ is obtained, where

$$P_{P_{t-+}} = \{p_{t-}, p_{t+}\},$$

$$T_{P_{t-+}} = \{t\} \cup \{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\},$$

$$\mathcal{F}_{P_{t-+}} = \{\langle p_{t-, t_{u_1t}} \rangle, \langle p_{t-, t_{u_2t}} \rangle, \dots, \langle p_{t-, t_{u_jt}} \rangle\} \cup \{\langle t_{u_1t}, p_{t+} \rangle, \langle t_{u_2t}, p_{t+} \rangle, \dots, \langle t_{u_jt}, p_{t+} \rangle\} \cup \{\langle p_{t+}, t \rangle\}$$

$$m_{0, P_{t-+}} = \pi_{P_{t-+}}(m_0) \text{ and } M_{f, P_{t-+}} = \pi_{P_{t-+}}([m_0]_{\mathcal{T}}).$$

Accordingly, Figure 3.34 depicts the user-task subnet $N(P_{t_2-+})$ for the task t_2 and $P_{t_2-+} = \{p_{t_2-}, p_{t_2+}\}$ of the example SecANet. Its initial marking is $m_{0, P_{t_2-+}} = \{p_{t_2-}\}$ and its set of final markings is $M_{f, P_{t_2-+}} = \{\{p_{t_2-}\}, \emptyset\}$. Hence, in the example of the user-task subnet $N(P_{t_2-+})$ resulting from the DMV SecANet, there are actually two final markings, i.e., \emptyset relates to the successful execution and $\{p_{-}\}$ corresponds to the obstruction marking. Indeed, there may also be a third final marking $\{p_{+}\}$, in which only the place indicating assignment is marked (i.e., the state that a user-task transition has been fired).

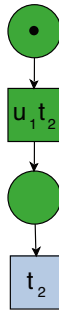


Figure 3.34 User-task subnet for t_2

Whereas for each user-task subnet decomposed from a SecANet, the projection π of the initial marking always results in the initial marking $\langle 1, 0 \rangle$ (according to the order $\{p_-\}, \{p_+\}$) or $\{p_-\}$ in set notation, for the general case, the projection of the terminal marking may result in the maximum number of three final markings, namely $\langle 1, 0 \rangle$, $\langle 0, 1 \rangle$, and $\langle 0, 0 \rangle$, or $\{p_-\}, \{p_+\}$, and \emptyset respectively. Hence, based on the definition of the L-type language $L(N) = \{w \in \Sigma^* | \exists \sigma \in T^* \exists m_f \in M_f : m_0[\sigma]m_f \wedge h(\sigma) = w\}$ (with $\Sigma = T$ and $T = T_{P_{t-+}}$), the initial marking $\{p_-\}$ and the three possible final markings (1) $\{p_-\}$, (2) $\{p_+\}$, and (3) \emptyset , the following three transition firing sequences σ and their corresponding (sets of) words w can be identified.

For $\{p_-\}[\sigma]\{p_-\}$, this results in

$$(1) \{h(\sigma)|\sigma = \langle \epsilon \rangle\} = \{w | w = \epsilon\},$$

$\{p_-\}[\sigma]\{p_+\}$ yields

$$(2) \{h(\sigma)|\sigma \in \{\langle t_{ut} \rangle | t_{ut} \in \{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\}\}\} = \{w | w \in \{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\}\},$$

and $\{p_-\}[\sigma]\emptyset$ is reflected in

$$(3) \{h(\sigma)|\sigma \in \{\langle t_{ut}, t \rangle | t_{ut} \in \{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\}\}\} = \{w | w \in \{\langle t_{u_1t}, t_{u_2t}, \dots, t_{u_jt} \rangle \cdot t\}\}.$$

(1) and (2) represent the cases in which the workflow task that is associated with the user-task assignment is not executed. Both cases can be caused either by an obstruction or an exclusive path that has not been taken. Here, especially (1) directly indicates an obstruction, in which no transition has been fired at all. However, if the obstruction affects an exclusive path that has not been taken and the workflow is completed (i.e., a marking involving the output place p_o is reached anyway), the obstruction has no effect. However, it could also have been the obstruction itself that lead to the decision to not take the affected path in the first place. In (2), only user-task transitions have been fired. On the one hand, this may encode an assignment of a user to a workflow task in an exclusive path that has not been taken. On the other hand, this may also result from an obstruction beforehand that blocks the execution of further workflow tasks, which users have already been assigned to. In (3), the final marking stands for successful user-task assignments that involve the execution of the workflow task. A firing sequence consists of a user-task transition followed by the transition of the corresponding workflow task.

According to these three cases, the language $L(N(P_{t-+}))$ can be denoted for each of the three possible final markings, namely

$$\begin{aligned}
L(N(P_{t-+}), M_{f, P_{t-+}} = \{\{p_{-}\}\}) &= \{\epsilon\}, \\
L(N(P_{t-+}), M_{f, P_{t-+}} = \{\{p_{+}\}\}) &= \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\}, \\
L(N(P_{t-+}), M_{f, P_{t-+}} = \{\emptyset\}) &= \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot t\}.
\end{aligned}$$

Due to the initial marking provided by the SecANet, the possible permutations of these three final markings are limited. This is because the initial marking directly enables all user-task transitions, such that every user-task subnet is always able to execute each of their user-task transitions (as in case (2)) at least once. A fired user-task transition may, however, only be a prefix of a successful user-task execution (as in case (1)). Hence, the set of final markings of a user-task subnet always contains at least $\{p_{t+}\}$ or \emptyset . Based on this and given some user-task subnet and some possible combinations of the three final markings, the overall language of the user-task subnet can be obtained by the union of the set of words resulting from each marking of the set of final markings. For example, for $M_{f, P_{t-+}} = \{\{p_{-}\}, \{p_{+}\}\}$, the language $L(N(P_{t-+})) = \{\epsilon\} \cup \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\}$ is obtained, the set of final markings $M_{f, P_{t-+}} = \{\{p_{-}\}, \{p_{+}\}, \emptyset\}$ results in the language $L(N(P_{t-+})) = \{\epsilon\} \cup \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cup \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot t\}$, and $M_{f, P_{t-+}} = \{\emptyset\}$ has the language $L(N(P_{t-+})) = \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot t\}$. Hence, similarly to the workflow subnet, the alphabet of the language of a user-task subnet may not always involve the workflow tasks, or, more formally:

Definition 3.51 (Language of the User-Task Subnet). *Based on the definition of the language applied to the user-task subnet, i.e., $L(N) = \{w \in T_{P_{t-+}}^* \mid \exists \sigma \in T_{P_{t-+}}^* \exists m_f \in M_{f, P_{t-+}} : m_{0, P_{t-+}}[\sigma]m_f \wedge h(\sigma) = w\}$, the language of a user-task subnet $N(P_{t-+})$ is*

$$\begin{aligned}
L_{t-+} = L(N(P_{t-+})) = \{w \in T_{P_{t-+}}^* \mid & \exists m_f \in M_{f, P_{t-+}} : m_f = \{p_{-}\} \wedge \epsilon = w \\
& \vee m_f = \{p_{+}\} \wedge w \in \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \\
& \vee m_f = \emptyset \wedge w \in \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot t\}\}.
\end{aligned}$$

The alphabet of L_{t-+} is denoted as $\Sigma_{t-+} = (\Sigma_N \cap \{t\}) \cup \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\}$.

For example, the language of each user-task subnet of the DMV SecANet with its different sets of final markings is $L(N(P_{t_1-+})) = \{\{t_{u_1t_1}, t_{u_2t_1}\} \cdot t_1\}$ and $L(N(P_{t_2-+})) = \{\epsilon, t_{u_1t_2}, t_2\}$.

Analogously to the decomposition of the user-task subnet \dot{N}_{TA} , constraint-related subnets will now be decomposed in order to determine their language. The definitions of SoD- and BoD-related subnets will be explained in order to understand how they can finally be subsumed under the notion of “constraint subnets”.

Definition 3.52 (SoD Subnets) *Based on the flattening of SoD constraints in Definition 3.31, the subnet*

$\dot{N}_{C_{SoD}} = N(\dot{P}_{C_{SoD}}) =$
 $\langle \dot{P}_{C_{SoD}}, \dot{T}_{C_{SoD}}, \dot{\mathcal{F}}_{C_{SoD}}, m_0, \dot{P}_{C_{SoD}}, M_f, \dot{P}_{C_{SoD}}, \dot{T}_{C_{SoD}},$
 $\text{id} \rangle$ is obtained, where

$$\dot{P}_{C_{SoD}} = \{SoD_{u_1t_1t_2}, SoD_{u_2t_1t_2}, \dots, SoD_{u_jt_kt_l}\},$$

$$\dot{T}_{C_{SoD}} = \{t_{u_1t_1}, t_{u_1t_2}, t_{u_2t_1}, t_{u_2t_2}, \dots, t_{u_jt_k}, t_{u_jt_l}\},$$

$$\dot{P}_{TA+SoD} = \{\langle SoD_{u_1t_1t_2}, t_{u_1t_1} \rangle, \langle SoD_{u_1t_1t_2}, t_{u_1t_2} \rangle,$$

$$\langle SoD_{u_2t_1t_2}, t_{u_2t_1} \rangle, \langle SoD_{u_2t_1t_2}, t_{u_2t_2} \rangle, \dots, \langle SoD_{u_jt_kt_l}, t_{u_jt_k} \rangle, \langle SoD_{u_jt_kt_l}, t_{u_jt_l} \rangle\}$$

and the markings $m_0, \dot{P}_{C_{SoD}} = \pi_{\dot{P}_{C_{SoD}}}(m_0)$ and

$$M_f, \dot{P}_{C_{SoD}} = \pi_{\dot{P}_{C_{SoD}}}([m_0]_{\mathcal{T}}).$$

Similarly to the basic user-task subnets, the basic SoD subnet can be determined:

Definition 3.53 (SoD Subnet). *For each SoD place $p_{\neq} \in \dot{P}_{C_{SoD}}$, the subnet*

$N_{\neq} = N(p_{\neq}) = \langle \{p_{\neq}\}, T_{p_{\neq}}, \mathcal{F}_{p_{\neq}}, m_0, p_{\neq}, M_f, p_{\neq}, T_{p_{\neq}}, \text{id} \rangle$ is obtained, where

$$p_{\neq} = \{SoD_{u_kt_l}\},$$

$$T_{p_{\neq}} = \{t_{u_k}, t_{u_l}\},$$

$$F_{p_{\neq}} = \{\langle SoD_{u_kt_l}, t_{u_k} \rangle, \langle SoD_{u_kt_l}, t_{u_l} \rangle\}$$

and the markings $m_0, p_{\neq} = \pi_{p_{\neq}}(m_0)$ and $M_f, p_{\neq} = \pi_{p_{\neq}}([m_0]_{\mathcal{T}})$.

Similarly to SoD subnets, BoD subnets also base on choice-places. However, given a large set of users, they may contain multitudes of outgoing arcs (and transitions).

Definition 3.54 (BoD Subnets). *Based on the flattening of BoD constraints in Definition 3.32, the subnet $\dot{N}_{C_{BoD}} = N(\dot{P}_{C_{BoD}}) = \langle \dot{P}_{C_{BoD}}, \dot{T}_{C_{BoD}}, \dot{\mathcal{F}}_{C_{BoD}}, m_0, \dot{p}_{C_{BoD}}, M_f, \dot{p}_{C_{BoD}}, \dot{T}_{C_{BoD}}, \text{id} \rangle$ is obtained, where*

$$\begin{aligned} \dot{P}_{TA+BoD} &= \{BoD_{u_1t_1t_2}, BoD_{u_2t_1t_2}, \dots, BoD_{u_jt_kt_l}\}, \\ \dot{T}_{BoD} &= \\ &\{t_{u_1t_1}\} \cup \{t_{ut_2}|t_{ut_2} \in T_{TA} - \{t_{u_1t_2}\} \wedge u \in U \wedge (u, t_2) \in TA\} \cup \\ &\{t_{u_2t_1}\} \cup \{t_{ut_2}|t_{ut_2} \in T_{TA} - \{t_{u_2t_2}\} \wedge u \in U \wedge (u, t_2) \in TA\} \cup \dots \cup \\ &\{t_{u_jt_k}\} \cup \{t_{ut_l}|t_{ut_l} \in T_{TA} - \{t_{u_jt_l}\} \wedge u \in U \wedge (u, t_l) \in TA\}, \\ \dot{F}_{BoD} &= \\ &\{\langle BoD_{u_1t_1t_2}, t_{u_1t_1} \rangle\} \cup \{\langle BoD_{u_1t_1t_2}, t_{ut_2} \rangle | t_{ut_2} \in T_{TA} - \{t_{u_1t_2}\} \wedge u \in U \wedge (u, t_2) \in TA\} \cup \\ &TA\} \cup \\ &\{\langle BoD_{u_2t_1t_2}, t_{u_2t_1} \rangle\} \cup \{\langle BoD_{u_2t_1t_2}, t_{ut_2} \rangle | t_{ut_2} \in T_{TA} - \{t_{u_2t_2}\} \wedge u \in U \wedge (u, t_2) \in TA\} \cup \dots \cup \\ &TA\} \cup \dots \cup \\ &\{\langle BoD_{u_jt_kt_l}, t_{u_jt_k} \rangle\} \cup \{\langle BoD_{u_jt_kt_l}, t_{ut_l} \rangle | t_{ut_l} \in T_{TA} - \{t_{u_jt_l}\} \wedge u \in U \wedge (u, t_l) \in TA\} \\ &TA\} \end{aligned}$$

and the markings $m_0, \dot{p}_{C_{BoD}} = \pi_{\dot{p}_{C_{BoD}}}(m_0)$ and $M_f, \dot{p}_{C_{BoD}} = \pi_{\dot{p}_{C_{BoD}}}([m_0]_{\mathcal{T}})$.

Analogously to the SoD subnet, the basic BoD subnet only contains one place.

Definition 3.55 (BoD Subnet). *For each BoD place $p_{=} \in \dot{P}_{BoD}$, the subnet $N_{=} = N(p_{=}) = \langle \{p_{=}\}, T_{p_{=}}, \mathcal{F}_{p_{=}}, m_0, p_{=}, M_f, p_{=}, T_{p_{\neq}}, \text{id} \rangle$ is obtained, where*

$$p_{=} = \{BoD_{u_jt_kt_l}\},$$

$$T_{p_{=}} = \{t_{u_jt_k}, t_{ut_l} | t_{ut_l} \in \dot{T}_{C_{BoD}} \wedge \exists \langle BoD_{u_jt_kt_l}, t_{ut_l} \rangle \in \dot{F}_{BoD}\},$$

$$F_{p_{=}} = \{\langle BoD_{u_jt_kt_l}, t_{u_jt_k} \rangle\} \cup \{\langle BoD_{u_jt_kt_l}, t_{ut_l} \rangle | t_{ut_l} \in T_{p_{=}} - \{t_{u_jt_k}\}\}$$

and the markings $m_0, p_{=} = \pi_{p_{=}}(m_0)$ and $M_f, p_{=} = \pi_{p_{=}}([m_0]_{\mathcal{T}})$.

SoD and BoD subnets both represent elementary nets because they contain only one place. Both have a very similar structure because they only have outgoing transitions. In order to keep the language definition as general and simple as possible, SoD and BoD constraint subnets will be subsumed under “constraint subnets”. They allow to encode choice-places with two outgoing user-task transitions affecting the same user (SoD), or one or more outgoing user-task transition(s) affecting different users (BoD).

Definition 3.56 (Constraint Subnet). Let $\dot{P}_C = \{\dot{P}_{SoD} \cup \dot{P}_{BoD}\}$ be the set of places, and $\dot{T}_C = \dot{T}_{SoD} \cup \dot{T}_{BoD}$ be the set of transitions of all constraint subnets. For each constraint place $p_c \in \dot{P}_C$, the subnet $N_c = N(p_c) = (\{p_c\}, T_{p_c}, \mathcal{F}_{p_c}, m_{0,p_c}, M_{f,p_c}, T_{p_c}, \text{id})$ is obtained, where

$$T_{p_c} = \{t_{ut} \mid \exists (p_c, t_{ut}) \in \dot{F}_C \wedge t_{ut} \in \dot{T}_C\},$$

$$F_{p_c} = \{(p_c, t_{ut}) \mid t_{ut} \in T_{p_c}\},$$

$$m_{0,p_c} = \pi_{p_c}(m_0) \text{ and } M_{f,p_c} = \pi_{p_c}([m_0]_{\mathcal{T}}).$$

Accordingly, Figure 3.31b depicts the constraint subnet $N(p_c)$ for the user u_1 and $p_c = p_{SoD_{u_1 t_{12}}}$ from the example SecANet. Its initial marking is $m_{0,p_c} = \{p_{SoD_{u_1 t_{12}}}\}$ and its set of final markings is $M_{f,p_c} = \{\emptyset\}$.

SoD and BoD constraints that are not in force are reflected in marked constraint places (i.e., the final marking set $\{p_c\}$). A constraint that is applied results in an unmarked constraint place (i.e., the final marking set \emptyset). Since constraint subnets only consist of one place, the terminal marking of the initial SecANet can result in the maximum number of two possible final marking sets of the constraint subnet. Similarly to the user-task subnet $N(P_{t-+})$, the firing sequences and the corresponding sets of words resulting from these different final markings can be determined according to the definition of the L-Type language $L(N) = \{w \in \Sigma^* \mid \exists \sigma \in T^* \exists m_f \in M_f : m_0[\sigma]m_f \wedge h(\sigma) = w\}$ (with $\Sigma = T$ and $T = T_{p_c}$).

For $\{p_c\}[\sigma]\{p_c\}$, this results in $\{h(\sigma) \mid \sigma = \langle \epsilon \rangle\} = \{w \mid w = \epsilon\}$,

and $\{p_c\}[\sigma]\emptyset$ determines

$$\{h(\sigma) \mid \sigma \in \{(t_{ut}) \mid \exists (p_c, t_{ut}) \in \dot{F}_C \wedge t_{ut} \in \dot{T}_C\}\} = \{w \mid w \in \{t_{ut} \mid \exists (p_c, t_{ut}) \in \dot{F}_C \wedge t_{ut} \in \dot{T}_C\}\}.$$

In contrast to the user-task subnets, constraint subnets do not allow to relate their final markings to successful or obstructed executions. On the one hand, in the example SecANet, the final marking set of the constraint is always \emptyset , regardless of whether the corresponding firing sequence is satisfiable or obstructed. On the other hand, depending on the number of users that are assigned to tasks affected by a constraint, there may always be constraint places that do take effect and others that do not. In fact, the example “DMV process” rather represents a special case in which the same constraint is applied in every possible firing sequence. The addition of one further user authorized for both tasks would result in terminal markings that would always contain one of the two constraint places. Hence, the set of final markings from all constraint subnets may consist of different combinations of marked constraint places. Analogously to user-task subnets, for each single constraint place and its corresponding subnet, the set of possible markings can be limited as well. In a constraint subnet, the marking \emptyset is always part of the set of final markings. This can easily be understood by considering the initial marking of the SecANet encoding again. Here, the initial marking enables all user-task transitions, including those that are affected by all constraint places. Hence, there is at least one firing sequence for each constraint subnet in which each of its outgoing transitions is fireable at least once (i.e., each transition is simply live). Analogously to the determination of the language of a user-task subnet $L(N(P_{t-+}))$, and to the observation that there are a maximal number of two different final markings, the language of $L(N(p_c))$ can be determined, namely

$$\begin{aligned} L(N(p_c)), M_{f,p_c} = \{p_c\} &= \{\epsilon\}, \\ L(N(p_c)), M_{f,p_c} = \{\emptyset\} &= \{t_{ur} \mid \exists (p_c, t_{ur}) \in \dot{F}_C \wedge t_{ur} \in \dot{T}_C\}. \end{aligned}$$

If the set of final markings of the constraint subnet contains both possible final markings, the union of these sets of words again constitutes the overall language of the constraint subnet.

Definition 3.57 (Language of Constraint Subnet). *Based on the language definition applied to the constraint subnet, i.e., $L(N) = \{w \in T_{p_c}^* \mid \exists \sigma \in T_{p_c}^* \exists m_f \in M_{f,p_c} : m_{0,p_c}[\sigma]m_f \wedge h(\sigma) = w\}$, the language of a user-task subnet $N(p_c)$ is*

$$L_c = L(N(p_c)) = \{w \in T_{p_c}^* \mid \exists m_f \in M_{f, p_c} : m_f = \{p_c\} \wedge \epsilon = w \\ \forall m_f = \emptyset \wedge w \in \{t_{ut} \mid \exists (p_c, t_{ut}) \in \dot{F}_C \wedge t_{ut} \in \dot{T}_C\}\}$$

The alphabet of L_c is denoted as $\Sigma_c = T_{p_c}$.

For example, the language of the constraint subnet of the DMV SecANet with the set of final markings $\{\emptyset\}$ is $L(N(p_c)) = \{t_{u_1 t_1}, t_{u_1 t_2}\}$.

Composition of Subnet Languages: The former introduction of the decomposition and the composition of free nets has revealed that the languages of the nets from which the subnets were decomposed can be obtained by using synchronization. In particular, because all SecANet subnets result from the decomposition of free Petri nets, the languages of a net can be composed from the languages of its subnets by applying the restriction (or shuffle) operator onto those subnets. This composition of languages will be conducted in reverse order to the order of how the subnets were decomposed. That way, the languages of subnets are composed step by step until they constitute the language of the initial SecANet, i.e., the languages of the user-task subnets, the constraint nets, their combination and the overall SecANet net will be obtained.

To begin with, after the language of each user-task subnet has been obtained, the languages of all user-task subnets can be composed. As the synchronization alphabet resulting from the intersection of the transitions of each user-task subnet is empty here, as previously observed, a special case of the restriction can be applied, namely the shuffle. Based on Definition 3.46 of the composition of subnet languages, the composition is conducted step by step such that there are intermediate steps in which the resulting languages represent languages of some further subnets of $N(\dot{P}_{TA})$. When the language of all user-task pairs have been synchronized, the resulting language is the language of $N(\dot{P}_{TA})$. Based on the consideration that at least either $\{p_+\}$ or \emptyset is a final marking of each user-task subnet, it can be concluded that all user-task transitions are in the alphabet of all user-task subnets, namely Σ_{TA} . However, given the case that there is an obstruction that does not allow to execute the workflow task in any firing sequence (i.e. the workflow is not satisfiable), it is possible that not all workflow tasks are in the alphabet of Σ_{TA} . This would however also imply that the considered path of the workflow is not satisfiable at all such that affected tasks that are exclusively in the considered path would not be part of the alphabet of the subnet of the workflow itself, namely Σ_N , either.

Definition 3.58 (Language of the User-Task Subnets). Based on Definition 3.46, the composition of the languages of all user-task subnets $L(N(\dot{P}_{TA}))$ decomposed from $N(\dot{P}_{TA})$, with $P_{t-+} \in P_{-+}$ and $P_{-+} = \{P_{t-+1}, \dots, P_{t-+|P_{-+}|}\}$, $P_{t-+1} \cap \dots \cap P_{t-+|P_{-+}|} = \emptyset$, and $P_{t-+1} \cup \dots \cup P_{t-+|P_{-+}|} = \dot{P}_{TA}$, which results in the language of all user-task subnets $L(N(\dot{P}_{TA}))$, can be denoted by means of the shuffle operator, namely

$$\begin{aligned} L_{TA} = L(N(\dot{P}_{TA})) &= L(N(P_{t-+1}))s_{y\emptyset}L(N(P_{t-+2}))s_{y\emptyset} \dots s_{y\emptyset}L(N(P_{t-+|P_{-+}|})) \\ &= \bigsqcup_{i=1}^{|P_{-+}|} L(N(P_{t-+i})) \end{aligned}$$

The alphabet of L_{TA} is denoted as $\Sigma_{TA} = \Sigma_N \cup \dot{T}_{TA}$.

The language of all user-task subnets for the example net is then $L(N(\dot{P}_{TA})) = \{\{t_{u_1t_1}, t_{u_2t_1}\} \cdot t_1\} \sqcup \{\epsilon, t_{u_1t_2}t_2\} = \{\{t_{u_1t_1}, t_{u_2t_1}\} \cdot t_1, \{t_{u_1t_1}, t_{u_2t_1}\} \cdot t_1 \sqcup t_{u_1t_2}t_2\}$.

The language of constraint subnets is composed by applying the restriction operator. Depending on whether the restriction operator acts on two sets of languages that constrain same elements (e.g., the language of the SoD subnets $\{t_{u_1t_1}, t_{u_1t_2}\}$ and $\{t_{u_1t_1}, t_{u_1t_3}\}$), or disjoint sets (e.g., the language of the SoD subnets $\{t_{u_1t_1}, t_{u_1t_2}\}$ and $\{t_{u_2t_1}, t_{u_2t_2}\}$), it might either restrict the sets of words or only act as a shuffle that allows both sets of words to occur (in case of disjoint sets of transitions), respectively. Because the initial marking allows to fire each user-task transition of a constraint subnet at least once, the alphabet of the resulting language of all constraint subnets encompasses all user-task transitions involved in the constraint subnets.

Definition 3.59 (Language of Constraint Subnets). Based on Definition 3.46, the composition of the languages of all constraint subnets $L(N(\dot{P}_C))$ decomposed from $N(\dot{P}_C)$, with $P_c \in \{P_{c1}, \dots, P_{c|P_C|}\}$, $P_{c1} \cap \dots \cap P_{c|P_C|} = \emptyset$, and $P_{c1} \cup \dots \cup P_{c|P_C|} = \dot{P}_C$, which results in the language of all user-task subnets $L(N(\dot{P}_C))$, can be denoted by means of the restriction operator, namely ¹¹

¹¹ In contrast to the language of TA , constraint subnets may in fact restrict each other because there may be multiple constraints that affect the same user-task transitions.

$$\begin{aligned}
L_C = L(N(\dot{P}_C)) &= L(N(p_{c1})) \textcircled{\text{R}} L(N(p_{c2})) \textcircled{\text{R}} \dots \textcircled{\text{R}} L(N(p_{c|\dot{P}_C|})) \\
&= \textcircled{\text{R}}_{i=1}^{|\dot{P}_C|} L(N(p_{ci})).
\end{aligned}$$

The alphabet of L_C is denoted as $\Sigma_C = \bigcup_{i=1}^{|\dot{P}_C|} T_{p_{ci}} = \dot{T}_C$ with $\dot{T}_C \subseteq \dot{T}_{TA}$.

For the example SecANet, the language of the single SoD place also represents the language of all constraint subnets because there is only one constraint place, such that $L(N(\dot{P}_C)) = \{t_{u_1t_1}, t_{u_1t_2}\}$.

To determine the language of all policy-related subnets, the language of the user-task subnets and the language of the constraint subnet can now be composed by means of the restriction operator and can be defined accordingly, namely

$$\begin{aligned}
L_{TA+C} &= L_{TA} \textcircled{\text{R}} L_C = ((L_{TA} \sqcup (\Sigma_C - \Sigma_{TA})^*) \cap (L_C \sqcup (\Sigma_{TA} - \Sigma_C)^*)) \\
&= ((L_{TA} \sqcup (\emptyset)^*) \cap (L_C \sqcup (\Sigma_{TA} - \Sigma_C)^*)) \\
&= ((L_{TA} \sqcup \{\epsilon\}) \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*)) \\
&= L_{TA} \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*).
\end{aligned}$$

Definition 3.60 (Language of Policy Subnet). *The composition of the languages of all policy-related subnets L_{TA} and L_C with the synchronization alphabet $\Sigma = \Sigma_{TA} \cap \Sigma_C = \Sigma_C$, which results in the language of all policy subnets $L(N(\dot{P}_{TA+C}))$, can be denoted by means of the restriction operator, namely*

$$L_{TA+C} = L_{TA} \textcircled{\text{R}} L_C = L_{TA} \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*)$$

The alphabet of L_{TA+C} is denoted as $\Sigma_{TA+C} = \Sigma_{TA}$.

In the example SecANet, this results in the previously determined language $L_{TA} \textcircled{R} L_{SoD} = \{t_{u_2t_1}t_1 \sqcup t_{u_1t_2}t_2, t_{u_1t_1}t_1\}$.

After the policy language is known, the net can finally be synchronized with the workflow subnet. Again, because all synchronizations base on subnets that are decomposed from free Petri nets, the restriction operator is used to determine the SecANet language, namely

$$\begin{aligned}
 L_{N+TA+C} &= L_N \textcircled{R} L_{TA+C} \\
 &= ((L_N \sqcup (\Sigma_{TA+C} - \Sigma_N)^*) \cap (L_{TA+C} \sqcup (\Sigma_N - \Sigma_{TA+C})^*)) \\
 &= ((L_N \sqcup (\dot{T}_{TA})^*) \cap (L_{TA+C} \sqcup (\emptyset)^*)) \\
 &= ((L_N \sqcup (\dot{T}_{TA})^*) \cap (L_{TA+C} \sqcup \{\epsilon\})) \\
 &= ((L_N \sqcup (\dot{T}_{TA})^*) \cap (L_{TA+C})) \\
 &= (L_N \sqcup (\dot{T}_{TA})^*) \cap L_{TA} \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*).
 \end{aligned}$$

Definition 3.61 (SecANet Language). *The composition of the languages of all workflow and policy-related SecANet subnets L_N and L_{TA+C} with the synchronization alphabet $\Sigma = \Sigma_N \cap \Sigma_{TA+C} = \Sigma_N$, which results in the language of the SecANet $L(N(\dot{P}_{N+TA+C}))$ or, equivalently, the language of the initial SecANet N_{SecANet} , i.e., $L((P_{TA+C}, T_{TA+C}, F_{TA+C}, m_0, [m_0]_T, T_{TA+C}, id))$, can be denoted by means of the restriction operator, namely*

$$\begin{aligned}
 L_{N+TA+C} &= L_N \textcircled{R} L_{TA+C} \\
 &= (L_N \sqcup (\dot{T}_{TA})^*) \cap L_{TA} \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*)
 \end{aligned}$$

The alphabet of L_{N+TA+C} is denoted as

$$\Sigma_{\text{SecANet}} = \Sigma_{N+TA+C} = \Sigma_N \cup \Sigma_{TA}.$$

For the example SecANet, this definition finally determines the previously identified SecANet language $L_N \textcircled{R} L_{TA+SoD} = \{(t_{u_2t_1}t_1 \sqcup t_{u_1t_2})t_2, t_{u_1t_1}t_1\}$.

3.2.4.6 Language Preservation

In the course of the flattening (cf. Definitions 3.30, 3.31 and 3.32), the different process aspects have all been integrated into a single representation, the SecANet. The interpretation of how the policy is applied to the process model follows from these definitions and bases on the considerations on fundamental access control

modeling concepts, as examined in Chapters 1 and 2. The definitions of the flattening were defined in a way that they encode certain conditions. For example, a user-task transition has to be executed before the corresponding task of the workflow net can be performed, or, based on a constraint, two user-task transitions for different tasks may be in a choice relation with each other. The SecANet definitions then allowed to define satisfiable and obstructed workflows and its related markings and firing sequences (cf. Definitions 3.33 and 3.34). Afterward, based on these definitions, the language of the SecANet was deduced by its decomposition and composition. The impetus for studying Petri net languages was to consider the behavioral integrity of the process aspects at the subnet level and the level of the overall net.

On the one hand, the decomposition of the SecANet revealed the individual languages of the resulting subnets. Here, each subnet language directly relates to the behavior of its initial input. Moreover, the workflow subnet is structurally identical to the initial WF-net. Only the final markings may differ, such that L_N (cf. Definition 3.47) may limit or extend the words of the language of the initial WF-net L_{wf} . This is then also reflected in the possibly different corresponding alphabets Σ_N and Σ_{wf} . For the user-task authorization and constraint language, their letters and words directly allow retracing their initial inputs. More specifically, the user-task authorization language L_{TA} (cf. Definition 3.51 and Definition 3.58) describes only allowed accesses and encodes user-task transition labelings. Each user-task letter t_{u_i} describes who is assigned to which tasks before the task t_i is (potentially) executed such that it can easily be retraced to the underlying authorization policy. That way, the modeling of policies provides an expressive language whose words allow to retrace which user has executed which task. Moreover, the constraint language L_C (cf. Definition 3.57 and Definition 3.59) determines the user-task transitions that are affected by SoD or BoD constraints. Its words encode the different choices provided by applying the constraints onto the given user-task authorization. Since the modeling of constraints introduces only places constraining the user-task transitions, it does not entail any new letters. For example, the language of an SoD subnet consisting of a place $SoD_{u_1 t_1 t_2}$ connected to the corresponding outgoing user-task transitions for which the same user u_1 can be assigned to either execute t_1 or t_2 results in the two words $\{t_{u_1 t_1}, t_{u_1 t_2}\}$. Each input that is processed into the representation remains unchanged in its behavior on the subnet level. This is also reflected in the alphabets of the two languages Σ_{TA} and Σ_C (cf. Definition 3.61 and 3.59). Hence, the previously identified *integrity of inputs* (cf. Section 3.2) can directly be observed from the words of the languages of the subnets as well.

On the other hand, the synchronization allowed to recompose the individual subnet languages, namely, the constraint language, the user-task language, and the language of the workflow subnet. The behavior of the overall process in a PAIS, that

is, the SecANet language L_{N+TA+C} (cf. Definition 3.61), results from their composition. Here, analogously to the language of each subnet, the SecANet language eventually follows from the definitions of the SecANet flattening (cf. Definitions 3.30, 3.31 and 3.32) as well. However, it is still open whether the *integrity of the overall representation* is given (cf. Section 3.2), namely, whether the behavior of the different process aspects is also preserved in the overall SecANet representation. This is supposed to be shown in the following by focusing on each of the different process aspects encoded in the SecANet separately and by relating their initial behavior to their behavior in the refined SecANet model. Because the flattening approach incorporates the policy behavior into the workflow, the foremost question is, to what extent the flattening preserves the language of the initial workflow, as depicted in Figure 3.35. It will moreover be considered whether the behavior of the policy is preserved in the overall net as well. The following section will disambiguate and examine language equivalence and language preservation in the given context.

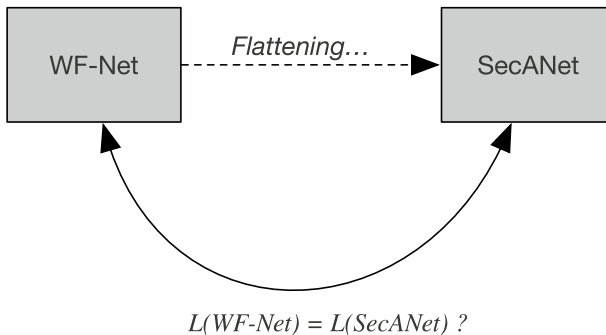


Figure 3.35 Language Preservation

While, in general, the question of language equivalence of two nets is not decidable, it is decidable for bounded nets [167].

Definition 3.62 (Language Equivalence). *Two Petri nets N_1, N_2 , where $L(N_1) = L(N_2)$, are called language equivalent.*

For example, the languages from the initial example DMV workflow net, denoted as N_{wf} , and its SecANet version are $L_{wf} = \{\mathbf{t_1t_2}\}$ with $\Sigma_{wf} = \{t_1, t_2\}$ and $L_{SecANet} = \{(t_{u_2t_1}\mathbf{t_1} \sqcup t_{u_1t_2})\mathbf{t_2}, t_{u_1t_1}\mathbf{t_1}\}$ with $\Sigma_{SecANet} = \{t_1, t_2, u_1t_2, u_2t_1, u_1t_2\}$, respectively (to highlight the difference, t_1 and t_2 are printed in bold). Although the behavior of the initial inputs can be observed by the letters of the WF-net in the overall SecANet to some extent here, the language $L_{SecANet}$ is not equivalent to the language of the initial workflow net L_{wf} , that is, $L_{wf} \neq L_{SecANet}$. However, besides this strict language equivalence, it can be observed that there are words of the language of the SecANet that contain all letters of the word of the language of the original WF-net in the same order. Such a word whose letters occur in the same order of some word but not necessarily one after another represents a so-called subword (sometimes also called “scattered” subword). Because this points to the intended idea behind the preservation of behavior, subwords and related concepts will now be considered in more detail.

A (scattered) subword u of some word v is a word obtained from v by removing any number of letters from arbitrary positions in v . Symmetrically, a superword v is obtained by inserting letters in arbitrary positions of u , or, more formally [174]:

Definition 3.63 (Subword, Superword and Closure). *Let Σ be some alphabet. A word $u = a_1a_2 \dots a_m$ with $a_1, a_2, \dots, a_m \in \Sigma$ is a subword of a word $v \in \Sigma^*$ if there are words $v_0, v_1, \dots, v_m \in \Sigma^*$ with $v = v_0a_1v_1a_2v_2 \dots a_mv_m$. This case is denoted as $u \sqsubseteq v$. If, in addition, $u \neq v$, it is denoted as $u \sqsubset v$ and u is called a proper subword of v . The subword relation can also be understood as the embeddability of one word into another, such that, in case that $u \sqsubseteq v$, it can be said that u embeds in v . Moreover, v then represents a superword of u as well. The subword property can be used to describe the downward closure of a language L , denoted by $L \downarrow = \{u \in \Sigma^* \mid \exists v \in L : u \sqsubseteq v\}$. It is the set of all subwords, i.e., all words that can be obtained from words in L by deleting letters. On the other hand, the upward closure of L , denoted by $L \uparrow = \{v \in \Sigma^* \mid \exists u \in L : u \sqsubseteq v\}$, is the set of all superwords, i.e., all words that can be obtained from words in L by inserting letters.*

For the word $t_{u_2t_1}t_1t_{u_1t_2}t_2$ of $L_{SecANet}$ and the word t_1t_2 of L_{wf} from the example, $t_1t_2 \sqsubset t_{u_2t_1}t_1t_{u_1t_2}t_2$. Hence, the behavior of the word t_1t_2 of the initial language is embedded in this word of the language of the SecANet. In this sense, the behavior of the word of the original net is preserved in the refined net. Based on this observation, the question is, whether for all subwords of the SecANet language, i.e., $\forall w \in$

$\{(t_{u_2t_1t_1} \sqcup t_{u_1t_2})t_2, \underline{t_{u_1t_1t_1}}\}$, the word of the language of the initial WF-net is a subword, that is, $t_1t_2 \sqsubset w$. If the word of L_{wf} , namely t_1t_2 embeds in all the words in $L_{SecANet}$, it then means that the behavior of the whole SecANet $L_{SecANet}$ completely preserves that of L_{wf} . In this case, the initial WF-net language would represent a scattered sublanguage (i.e., a language consisting only of (scattered) subwords) of the SecANet language. The language L_{wf} considered as the “subword language” is only a subset of possible subwords of $L_{SecANet}$. Hence, the aim is not to investigate subword-related properties within one language (e.g., the downward closure) but to compare two languages with regards to the subword property.

This idea of behavioral preservation will be examined more thoroughly by means of the example SecANet and its initial WF-net in the following. Based on the definition of the flattening, it is known that the SecANet introduces new letters. The set of all words of which t_1t_2 is a scattered subword can be overgeneralized as $\{\Sigma^*t_1\Sigma^*t_2\Sigma^*\}$, with $\Sigma = \{t_1, t_2, t_{u_1t_1}, t_{u_2t_1}, t_{u_1t_2}\}$. However, in order to obtain the words in L_{wf} (and, thus, to not change the behavior of the initial workflow net), only the letters that do not occur in the original language must be deleted. Hence, in order to only consider the order of letters that are given from the initial WF-net language (i.e., $\{t_1t_2\}$), all user-task transitions that are not in the initial net are not taken into account, (i.e., the underlined letters $\{(t_{u_2t_1t_1} \sqcup t_{u_1t_2})t_2, \underline{t_{u_1t_1t_1}}, \underline{t_{u_1t_2}t_2}\}$). Therefore, these (underlined) letters are relabeled with silent transitions (which is depicted with black τ -labeled transitions in Figure 3.36). That way, all user-task letters are transformed into the empty word. Such a deletion of only the letters that are not in the words of L_{wf} is supposed to result into proper subwords of the words of $L_{SecANet}$. If the flattening has not changed the behavior of the initial WF-net, these subwords are supposed to be equivalent to all the words of the language of the initial workflow net. Concerning the example, this silencing of all letters that are not in the language of the initial WF-net results in the language $\{(\epsilon t_1 \sqcup \epsilon)t_2, \epsilon t_1\}$. Here, only the first created subword $(\epsilon t_1 \sqcup \epsilon)t_2$, which represents a satisfiable execution, is actually the word of the WF-net, i.e., t_1t_2 ¹². The other subword $\{t_1\}$, which relates to the obstruction, is not in the WF-net language $\{t_1t_2\}$. In contrast, this last word represents a subword of the words in the initial WF-net itself. Thus, it must be noted that the SecANet language with terminal markings, which has been considered for the example so far, does not fully preserve the behavior of the WF-net. Although all words in L_{wf} are subwords of words in $L_{SecANet}$, they are not subwords of all the SecANet words, as required initially. More specifically, the resulting subword

¹² Note that not all subwords may embed in the same SecANet words. For example, in workflows with exclusive branching, different subwords embed in different words of the corresponding SecANet.

t_1 of L_{SecANet} is not a superword of t_1t_2 , and t_1t_2 is not a subword of the word t_1 . Thus, it is not sufficient to require that all words of L_{wf} are subwords of words of L_{SecANet} . In order to achieve a kind of mutual correspondence between the words of the two languages, it must therefore additionally be required that all words in L_{SecANet} must be superwords of words from L_{wf} as well.

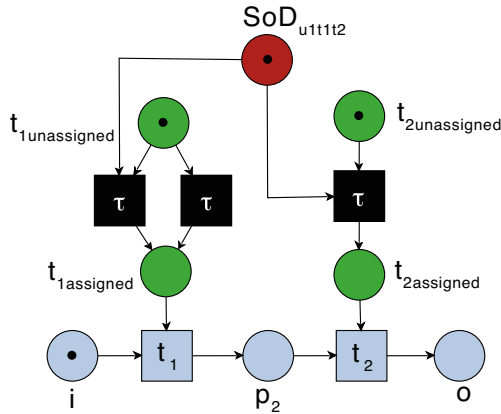


Figure 3.36 Flattened net with silent transitions

In summary, in order to have a refined language (i.e., the SecANet language) that truly preserves the behavior of an original language (i.e., the language of the original workflow net), all words in the original language need to be subwords of the refined language, but symmetrically, all words in the refined language are supposed to be superwords of the original language as well. The latter condition then just expresses that the subword property must apply for all words in the SecANet language. It is therefore not a question of language equivalence in the original sense, but a matter as to whether behavior encoded by subwords or superwords finds its respective counterpart in the original (e.g., WF-Net) or refined (SecANet) language respectively. In order to express this more formally, the notion of “complete subword-preservation” will be introduced in the following.

Definition 3.64 (Complete Subword-Preservation). *Given two languages L_1 and L_2 with the same alphabet Σ . If L_1 only contains subwords of L_2 and L_2 only contains superwords of L_1 , L_1 completely embeds in L_2 . As the behavior of L_1 is then completely embedded in L_2 , it is said that L_2 completely preserves the behavior of the words in L_1 . For this case, the subword notation is extended for languages: $L_1 \sqsubset L_2$ means all words $w \in L_1$ are proper subwords of words in $v \in L_2$ such that $w \sqsubset v$, and all words $v \in L_2$ are proper superwords of the words $w \in L_1$ such that $w \sqsubset v$.*

Thus, the behavior of a language L_1 is only completely preserved in L_2 if the words of L_2 do not allow behavior that is not embeddable from the words of L_1 . Consequently, L_2 represents a subset of the upward closure of $L_1 \uparrow$ and, L_1 is a subset of the downward closure of $L_2 \downarrow$. For a specific set of deletion letters, typically δ_Σ with $\Sigma = \Sigma_{L_2} - \Sigma_{L_1}$, the two languages are then equivalent. Since the notion of language preservation does not focus on language equivalence but on complete subword-preservation, the envisaged examination on subword-preservation is supposed to carefully consider which SecANet language to take into account. In this regard, in order to investigate the preservation of workflow behavior in the SecANet, the language of the initial workflow net can fully be embedded in the SecANet language only in case of no obstruction. Specifically, this is because the SecANet encoding is not intended to impair satisfiable workflow executions. If there are user-task assignments that do not obstruct the initial workflow, the language of the SecANet in which the corresponding user-task transitions are silenced, is then supposed to be equivalent to the language of the initial workflow net. On the other hand, the SecANet encoding exactly intends to manipulate the behavior of the original WF-net in case of an obstruction. Then, the user-task assignments conflict with the WF-net such that its initial behavior is narrowed down. Therefore, the WF-net behavior can not be preserved in obstruction words of the SecANet. Instead, complete subword-preservation of the workflow language can only be required for satisfiable executions. Hence, in order to exclude obstructions, the set of terminal markings that defines the SecANet language additionally needs to involve p_o (i.e., only satisfiable words are possible).

Definition 3.65 (Satisfiable SecANet Language). *The satisfiable SecANet language is denoted as $L_{\text{sat}} = L(\langle P, T, F, m_0, [m_0]_{\mathcal{T}\text{sat}}, T, id_{\mathcal{T}} \rangle)$, with the set of satisfiable terminal markings $[m_0]_{\mathcal{T}\text{sat}} = \{m | m \in [m_0]_{\mathcal{T}} \wedge m \geq p_0\}$.*

That way, the language of the example SecANet can be reduced to only the first set of words, that is, $L_{\text{sat}}(\text{SecANet}) = \{(t_{u_2}t_1 \mathbf{t}_1 \sqcup t_{u_1}t_2) \mathbf{t}_2\}$ such that only complete executions are considered and obstructions are excluded. Silencing the user-task-transitions then results in the set of words $\{(\epsilon \mathbf{t}_1 \epsilon) \mathbf{t}_2\}$. Thus, in case of not reaching obstructions, the flattening preserves the behavior of the initial control flow aspect of the example net. Moreover, $(\Sigma_{\text{SecANet}} - \Sigma_N)^* t_1 (\Sigma_{\text{SecANet}} - \Sigma_N)^* t_2 (\Sigma_{\text{SecANet}} - \Sigma_N)^*$ can be denoted as the subset of the upward or the downward closure of the SecANet language that illustrates the possibly deleted or filled up words in $(\Sigma_{\text{SecANet}} - \Sigma_N)^*$. Interestingly, this can equivalently be expressed by the shuffle operator, namely $t_1 t_2 \sqcup \{\Sigma_{\text{SecANet}} - \Sigma_N\}^*$. This again highlights the intended idea of subword-preservation because the shuffle just encodes that the letters of the words involved in the shuffle can appear in any order but letters of the same word, in particular of the original word $t_1 t_2$, remain in the same order.

Based on these considerations, the complete subword-preservation is now supposed to be shown for the general SecANet language. Analogously to the deleted letters in the example, the selection of letters to be deleted can be generalized as well. This deletion will subsequently be essential to prove the integrity of the behavior of each process aspect in the overall SecANet. As the focus of this thesis lies on reaching the end of the process, the first step is to prove that the language of the original workflow net is not changed by the SecANet encoding. After that, analogously to the WF-net, it will also be shown that the behavior of the policy is preserved by the SecANet encoding. The subsequent theorem relates the language of the original workflow net to the language of the SecANet, which encodes the policy too.

Theorem 3.1 (SecANet Workflow Subword-Preservation). *Let $N^{pol} = \langle N_{wf}, U, TA, C \rangle$ be a policy-aware workflow net with the WF-net $N_{wf} = (P_{wf}, T_{wf}, F_{wf}, m_0, \{p_o\}, T_{wf}, id)$ ¹³ and $N_{SecANet} = \langle P_{TA+C}, T_{TA+C}, F_{TA+C}, m_{TA+C_0}, [m_{TA+C_0}]_{T_{sat}}, T_{TA+C}, id \rangle$ be the corresponding SecANet.*

Then, given that $L_{sat}(N_{SecANet}) \neq \emptyset$, the SecANet completely preserves the behavior of the WF-net $L(N_{wf})$, such that the words of $L(N_{wf})$ are subwords that are completely preserved in the words of $L_{sat}(N_{SecANet})$, i.e., $L_{wf} \sqsubseteq L_{satSecANet}$.

Here, the rather elaborate determination of the SecANet language is finally paying off. With its help, the complete subword-preservation can now be proven by simply silencing all transitions that are not in the net under investigation. The deletion homomorphism (cf. Definition 3.37) will be used to assign the empty word to the respective letters, in particular all letters that are not in the WF-net, that is, $\delta_{\Sigma_{SecANet} - \Sigma_{wf}}$. Because only the satisfiable SecANet language is regarded, the languages and alphabets of the SecANet subnets are determined by the satisfiable terminal markings as well. For better readability, the subscripted “sat” is here only used for clarification in the beginning. Since, as explained before, the different words can result for obstructed, satisfiable, but also simply exclusive branches, all of the subnet languages defined above (cf. Section 3.2.4.5) basically remain unchanged. The consideration of the satisfiable SecANet language will only make a difference for the language of the workflow subnet and its alphabet. Hence, the investigation of complete workflow subword-preservation can be broken down into the following equation. For this step, all letters in $\Sigma_{TA} - \Sigma_{wf}$ are deleted from the SecANet language:

¹³ In order to avoid confusion with the workflow subnet L_N , N_{wf} and L_{wf} explicitly describe the initial WF-Net and its corresponding language. L_N relates to the workflow subnet from the SecANet.

$$L_{\text{sat}}(N_{\text{SecANet}}) = L_{\text{sat}N+TA+C} = (L_N \sqcup (\dot{T}_{TA})^*) \cap L_{TA} \cap (L_C \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*),$$

which results in

$$\begin{aligned} \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{N+TA+C}) &\stackrel{*}{=} (\delta_{\Sigma_{TA}-\Sigma_{wf}}(L_N) \sqcup (\epsilon)^*) \\ &\quad \cap \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{TA}) \cap (\epsilon \sqcup (\Sigma_N \cup \epsilon)^*) \\ &\stackrel{**}{=} L_N \cap \bigsqcup_{i=1}^{|P_{-+}|} \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{t_{-+}}) \cap (\Sigma_N)^* \\ &\stackrel{***}{=} L_N = L_{wf}. \quad \square \end{aligned}$$

(*) $L_C = \epsilon$, because deleting the user-task letters from the language of a single constraint subnet

$$\begin{aligned} L_C = \{w \in T_{p_C}^* \mid &\quad \exists m_f \in M_{f,p_C} : m_f = \{p_C\} \wedge \epsilon = w \\ &\quad \forall m_f = \emptyset \wedge w \in \{t_{ut} \mid \exists \{p_C, t_{ut}\} \in \dot{F}_C \wedge t_{ut} \in \dot{T}_C\} \\ &\quad \text{results in} \\ \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_C) = \{w \in T_{p_C}^* \mid &\quad \exists m_f \in M_{f,p_C} : m_f = \{p_C\} \wedge \epsilon = w \\ &\quad \forall m_f = \emptyset \wedge \epsilon = w\} \\ &\quad = \{\epsilon\}, \end{aligned}$$

$$\text{such that } L_C = \bigsqcup_{i=1}^{|\dot{P}_C|} L(N(p_{Ci})) = \bigsqcup_{i=1}^{|\dot{P}_C|} \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_C) = \bigsqcup_{i=1}^{|\dot{P}_C|} \{\epsilon\} = \{\epsilon\}.$$

(**) Because $L_{TA} = \bigsqcup_{i=1}^{|P_{-+}|} L(N(P_{t_{-+}i}))$ then $\delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{TA}) = \bigsqcup_{i=1}^{|P_{-+}|} \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{t_{-+}})$. Deleting the user-task assignment transitions from the language of a single user-task subnet

$$\begin{aligned} L_{t_{-+}} = \{w \in T_{p_{t_{-+}}}^* \mid &\quad \exists m_f \in M_{f,p_{t_{-+}}} : m_f = \{p_{-}\} \wedge \epsilon = w \\ &\quad \forall m_f = \{p_{+}\} \wedge w \in \{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\} \\ &\quad \forall m_f = \emptyset \wedge w \in \{\{t_{u_1t}, t_{u_2t}, \dots, t_{u_jt}\} \cdot t\}\} \\ &\quad \text{results in} \\ \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{t_{-+}}) = \{w \in T_{p_{t_{-+}}}^* \mid &\quad \exists m_f \in M_{f,p_{t_{-+}}} : m_f = \{p_{-}\} \wedge \epsilon = w \\ &\quad \forall m_f = \{p_{+}\} \wedge \epsilon = w\} \\ &\quad \forall m_f = \emptyset \wedge w \in \{\{\epsilon\} \cdot t\}\}, \\ &\quad \text{such that, for a task } t \text{ that occurs in all executions,} \\ &\quad \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{t_{-+}}) = \{t\}, \\ &\quad \text{and, for a task } t \text{ that does not occur in all executions,} \\ &\quad \delta_{\Sigma_{TA}-\Sigma_{wf}}(L_{t_{-+}}) = \{\epsilon, t\}. \end{aligned}$$

(***) Depending on whether the workflow net contains exclusive branches or not, two cases can be distinguished: (1) In sequential workflows, all tasks need to be executed to complete the workflow, such that $L_{wf} = \{t_1 \dots t_{|T_{wf}|}\}$, that means, the execution order of each task involved in the workflow is unconstrained. Hence, for a corresponding L_{TA} , deleting the user-task letters results in the set of words $L_{TA} = \{t_1 \sqcup \dots \sqcup t_{|T_{wf}|}\}$ (based on Definition 3.30, it directly follows that the number of place pairs $|P_{-+}|$ results from the number of tasks $|T_{wf}|$). In parallel workflows, all tasks occur as well. However, concurrent tasks may occur in different orders, such that L_{TA} would involve all the task as well, namely $L_{TA} = \{t_1 \sqcup \dots \sqcup t_{|T_{wf}|}\}$. Hence, given that every task may only occur once, $t_1 \sqcup \dots \sqcup t_{|T_{wf}|}$ is the least restrictive order in which the tasks of the workflow may occur because all tasks are shuffled arbitrarily. Consequently, it can directly be observed that L_{TA} contains all the words of L_{wf} as well, wherefore the intersection of L_{wf} with L_{TA} is just L_{wf} itself. (2) In workflows that additionally involve exclusive branches, a workflow can be completed even though not all of its tasks are involved. For example, a workflow of three tasks may have the language $L_{wf} = \{t_1 t_2 t_3, t_1 t_3\}$ (i.e., t_2 may be executed or not). With regards to the subsequent language of L_{TA} , this can be subsumed as $L_{wf} = \{t_1 \cdot \{\epsilon, t_2\} \cdot t_3\}$ and would then also be reflected in the corresponding $L_{TA} = \{t_1 \sqcup \{\epsilon, t_2\} \sqcup t_3\}$. In other words, if ϵ is additionally given for a task t , it corresponds to the possibility given by the workflow to leave out this task (or letter). When it comes to tasks in exclusive branches, the languages of the corresponding user-task subnets L_{t-+} is $\{t, \epsilon\}$ (i.e., the branch in which the task t occurs is chosen or not). In a rather theoretic workflow in which all tasks are in mutually exclusive branches this would then result in $L_{TA} = \{\{\epsilon, t_1\} \sqcup \dots \sqcup \{\epsilon, t_{|T_{wf}|}\}\}$. Similarly to (1), for workflows that involve exclusive branches, it can directly be observed that L_{TA} contains all the words of L_{wf} as well because all task that may (or may not) occur can be arbitrarily shuffled. L_{TA} then contains all possible combinations of occurrence and non occurrence of the tasks given by the initial workflow. For example, the fictive workflow would involve all words from ϵ to $t_1 \sqcup \dots \sqcup t_{|T_{wf}|}$. Thus, the intersection of L_{wf} with L_{TA} is just L_{wf} itself again. Hence, for both cases (1) and (2), because L_{TA} does not correspond to the workflow structure encoded in L_N but provides all possible combination of (optional or exclusive) tasks, it represents a less restrictive language than L_{wf} . Moreover, because only satisfiable executions are considered, namely all possible terminal firing sequences of the initial workflow, the language of the workflow subnet and its alphabet are identical to the language and alphabet of the initial workflow, such that $L_N = L_{wf}$ and $\Sigma_N = \Sigma_{wf}$. Thus, the words of L_{wf} completely embed in the words of $L_{\text{satSecANet}}$, such that L_{wf} is a scattered sublanguage of the $L_{\text{satSecANet}}$ and $L_{wf} \sqsubset L_{\text{satSecANet}}$.

Hence, concerning satisfiable executions, the behavior of the original workflow net is completely preserved in the behavior of the SecANet.

In order to investigate whether the policy behavior is preserved in the SecANet, now, the workflow related transitions will now be removed instead. As the focus does not lie on the original WF-net here, it is sufficient to delete only those workflow tasks that actually occur in the language of the net, such that the deletion homomorphism is δ_{Σ_N} . Whereas the policy behavior has been flattened into the initial workflow net, there are no initial policy nets into which the WF-net is incorporated. The integrity of the policy behavior can thus not be examined directly because there is no initial net model of the policy and no initial language of a policy-net, in particular, of an authorization or a constraint net comparable to the initial WF-net. Therefore, this behavioral preservation can be shown to a limited extent only. It will only be examined whether the languages of the subnets, which, as previously determined, fully represent their inputs, are retained in the overall SecANet language if the workflow tasks are neglected. Because satisfiability corresponds to the complete execution of the initial workflow, considering only policy parts that lead to satisfiable executions would limit policy behavior here. Therefore, all possible terminal SecANet executions need to be regarded. That way, the policy behavior is still fully reflected, notwithstanding the consequences for the workflow, namely its satisfiability or obstructability. This is supposed to show that at least the languages of the subnets that encode the policy are fully preserved in the language of the SecANet.

Theorem 3.2 (SecANet Policy Subword-Preservation). *Let $N^{pol} = \langle N, U, TA, C \rangle$ be a policy-aware workflow net with the WF-net $N = \langle P, T, F, m_0, \{p_0\}, T, id \rangle$, the set of users U , the user-task authorization $TA \subseteq \mathcal{U} \times T$ and the set of constraints C and let $N_{\text{SecANet}} = \langle P_{TA+C}, T_{TA+C}, F_{TA+C}, m_{TA+C_0}, [m_{TA+C_0}]_T, T_{TA+C}, id \rangle$ be the corresponding SecANet. Then, the SecANet completely preserves the behavior of the policy, such that it completely embeds the policy-related words of the languages L_{TA} and L_C , which encode the user-task authorization TA and the constraints C , i.e., $\delta_{\Sigma_N}(L_{TA}) \sqsubset L_{\text{SecANet}}$ and $\delta_{\Sigma_N}(L_{TA})(L_C) \sqsubset L_{\text{SecANet}}$.*

The deletion of the WF-task letters in Σ_N from the SecANet language

$$L_{N+TA+C} = (L_N \sqcup (\dot{T}_{TA})^*) \cap L_{TA} \cap (\delta_{\Sigma_N}(L_C) \sqcup (\Sigma_N \cup \{\dot{T}_{TA} - \dot{T}_C\})^*),$$

results in

$$\begin{aligned} \delta_{\Sigma_N}(L_{N+TA+C}) &= (\epsilon \sqcup (\dot{T}_{TA})^*) \cap \delta_{\Sigma_N}(L_{TA}) \cap (\delta_{\Sigma_N}(L_C) \sqcup (\epsilon \cup \{\dot{T}_{TA} - \dot{T}_C\})^*) \\ &= (\dot{T}_{TA})^* \cap \delta_{\Sigma_N}(L_{TA}) \cap (\delta_{\Sigma_N}(L_C) \sqcup \{\dot{T}_{TA} - \dot{T}_C\})^* \\ &= \delta_{\Sigma_N}(L_{TA}) \cap (L_C \sqcup \{\dot{T}_{TA} - \dot{T}_C\})^* \\ &\stackrel{*}{=} \bigsqcup_{i=1}^{|P_{-+}|} \delta_{\Sigma_N}(L_{t_{-+}}) \cap L_C \sqcup \{\dot{T}_{TA} - \dot{T}_C\})^* \end{aligned}$$

$$\stackrel{**}{=} \underbrace{\delta_{\Sigma_N}(L_{TA})}_{L_{TA} \text{ without the workflow transitions}} \cap (L_C \sqcup \underbrace{\{\dot{T}_{TA} - \dot{T}_C\}}_{\text{Kleene closure of user-task transitions not involved in any constraint}}).$$

(*) L_C is not affected by the deletion because it does not contain any workflow task letter and, thus, remains unchanged. The language that encodes user-task assignments

$$\begin{aligned} L_{t_{-+}} &= \{w \in T_{P_{t_{-+}}}^* \mid \exists m_f \in M_{f, P_{t_{-+}}} : m_f = \{p_{-}\} \wedge \epsilon = w \\ &\quad \vee m_f = \{p_{+}\} \wedge w \in \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \\ &\quad \vee m_f = \emptyset \wedge w \in \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot t\}\} \text{ results in} \\ \delta_{\Sigma_N}(L_{t_{-+}}) &= \{w \in T_{P_{t_{-+}}}^* \mid \exists m_f \in M_{f, P_{t_{-+}}} : m_f = \{p_{-}\} \wedge \epsilon = w \\ &\quad \vee m_f = \{p_{+}\} \wedge w \in \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \\ &\quad \vee m_f = \emptyset \wedge w \in \{\{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\} \cdot \epsilon\}\}. \end{aligned}$$

such that, when it comes to user-task transitions that always occur, $\delta_{\Sigma_N}(L_{t_{-+}}) = \{t_{u_{1t}}, \dots, t_{u_{jt}}\}$, and $\delta_{\Sigma_N}(L_{t_{-+}}) = \{\epsilon, \{t_{u_{1t}}, t_{u_{2t}}, \dots, t_{u_{jt}}\}\}$ concerning user-task transitions that are in exclusive relation with user-task transitions for other tasks (due to some constraints). Hence, the behavior of the authorization is still encoded in the name of the transition directly resulting from the construction rule (cf. Definition 3.30), and all these transitions together still constitute the complete picture of the initial authorization policy. The constraints operating on it are still encoded directly in the respective constraint places and the associated outgoing transitions.

Thus, the words in L_{TA} still allow drawing direct conclusions on the underlying initial authorization policy. As explained before, with regard to L_C , this is only possible if there are user-task authorizations for which the existing constraints can actually be applied as well.

(**) It can directly be observed that the policy-related behavior encoded in L_{TA} and L_C is not changed if only the policy-related user-task transitions of the SecANet language are regarded. The composition of both languages $\delta_{\Sigma_T}(L_{TA})$ and L_C that results in the overall policy-behavior is realized by the Kleene closure of the user-task transitions not involved in any constraints. If the transitions of the constraints exactly correspond to the transitions in the language of the user-task assignments, it is empty and results in ϵ . With the help of the shuffle operator, the Kleene closure provides the missing links such that the constraint language and user-task language can be intersected. In particular, based on L_C , words can be built that are also in L_{TA} , whereby the intersection operator limits these words to exactly those words which also occur in L_{TA} . That way, the language that encodes the policy combines the user-task language and the constraint language and fully embeds in the SecANet language, such that its behavior is completely preserved as well, i.e., $\delta_{\Sigma_N}(L_{TA}) \sqsubset L_{\text{SecANet}}$ and $\delta_{\Sigma_N}(L_{TA})(L_C) \sqsubset L_{\text{SecANet}}$. \square

In conclusion, it has been shown that the behavior of all process aspects is embedded in the overall SecANet, which means that the integrity of the overall representation is preserved. In this regard, the words encoding workflow or policy behavior represent a subset of the downward closure of the SecANet language. In turn, the SecANet language represents a subset of the upward closure of the words encoding the initial inputs. Just as, in general, the upward closure offers an over-approximation of system behavior [137], the SecANet behavior can be seen as an over-approximation of the behavior of the initial inputs. The results of this language-related examination can directly be used to interpret the words of different language types. For example, in terms of a terminal language, one can immediately conclude from the SecANet property of complete subword-preservation that words that can be embedded from the language of the initial WF-net are satisfiable words. In the opposite case, if there are SecANet words that are not superwords of words from the language of the initial workflow net, they do not encode complete process executions and, thus, represent obstructions.

3.2.5 Cyclic Behavior and Policy Re-Enactment

The SecANet encoding presented so far is designed in such a way that each user can be assigned only once and each task can be executed only once. The tokens that

enable user-task assignments can be consumed but not produced, hence the overall number of tokens at the end of execution is smaller than the number of tokens in the initial marking. If a workflow execution were to enter a loop in an as-yet-acyclic encoding of a SecANet, the net constructs encoding the policy would lack the tokens to continue to provide its desired functionality. Thus for a comprehensive workflow structure with cyclic behavior, how can the encoding remain functional if its tasks are supposed to be executed multiple times?

First of all, simply restoring the initial tokens would fall short, since there would be a danger of manipulating the original encoding and behavior of a SecANet. This can be illustrated for the places of constraints as well as the places of user-task subnets. It can be observed that constraint places always become empty in the course of a user-task assignment, that is, a firing of a user-task transition. Thus the number of empty constraint places always corresponds to the number of executed user-task assignments to which these constraints apply. Even if all tasks of a workflow were affected by constraints and all tasks were executed, the number of constraint places containing no tokens could not exceed the number of tasks. Hence in cases where the number of constraint places is greater than the number of tasks for which the constraint places were encoded, there are always some marked constraint places left after the tasks are executed. In user-task subnets, there may be cases in which a task is not finally assigned at all, although the user-task assignment has already been executed (cf. Section 3.2.4.5). Therefore, there may be leftover tokens in places of user-task constructs as well. Hence for all places of the policy subnets, it must always be assumed that there are leftover tokens when entering a loop before the initial marking is created again. To restore the initial marking, moreover, one could consider bounding the capacity of the initially marked places to 1, and in that way ensure safeness. However, although this would reduce the possible combinations of places containing leftover tokens (because the initially marked places could be neglected), it would cause other problems. Up to now, the SecANet encoding allows each place to contain an unlimited number of tokens. Such a Petri net is referred to as an infinite-capacity net. A finite-capacity Petri net, in contrast, is one in which there is a maximum number of tokens defined for each place. For such Petri nets, there is an additional firing rule which says that after firing a transition the numbers of tokens in its output places must not exceed their maximum capacities. Since the firing rule then needs to consider the outgoing places, this would change the desired condition-event principle. A condition that is fulfilled would not imply enabledness, which would confuse the interpretation of the behavior of a SecANet. Moreover, such an additional firing rule represents an extension that is comparable, for instance, to the introduction of further tokens in colored Petri nets (which also implies further firing rules or arc weights). Hence to keep the intuitive understanding of the SecANet model and

sustain the possibility of applying rather efficient techniques to it, additional firing rules are avoided, as elaborated earlier. Moreover, each pure finite-capacity net can be transformed into an infinite-capacity net [155]. As depicted in Section 3.2.3.1, the user-task construct represents an example of such a transformation. Therefore, a slightly higher modeling complexity is preferred in order to keep the Petri nets as simple as possible and yet preserve the established properties.

Thus all possible token distributions in the parts of the policy subnets that are affected by a loop have to be considered before the initial markings can be restored. For this, the concept of “policy re-enactment” will subsequently be presented. Such a measure aims to restore the initial functionality of the (affected part of the) policy, that is, to “re-enact” it. To ensure this, all leftover tokens in policy places are supposed to be consumed before the initially marked policy places are marked again. Although this encoding will be more complex than the encoding for acyclic nets, it will preserve the concept of an encoding described in the acyclic case. Thus in order not to compromise the established SecANet properties, the re-enactment constructs must be constructed of safe, plain, and pure infinite-capacity P/T net parts as well. In this process, the idea is to ensure only the desired functionality but not to interfere with the behavior of the SecANet. The focus will thus not be on the looping behavior, the transitions, or the language of the loops, but instead on how the initial marking for the policy parts affected by a loop can be obtained again, hence the possible states (or token distributions) will be crucial. Accordingly, the re-enactment constructions are designed in such a way that they eventually act only for routing purposes, so that all transitions involved in the re-enactment constructions could actually be silenced and yet cyclic SecANet behavior would still be supported.

To some extent, the basic idea of the net constructions discussed below can be compared to so-called “start, run, clear” Petri net constructions. They control the execution of a Petri net and can be used to obtain the previously mentioned normal forms, for example always with the same initial and end markings [167]. The objective of the re-enactment constructions will not be an entire net, but only the parts of the subnets that involve user-task authorizations and constraints. As such, it will focus only on the potential end markings and initial markings in the specific context of the SecANet encoding. (A “run” place controlling each transition of the entire execution is not necessary, and in fact would represent an undesired self-loop.) In all of this, it is key to recall that the process models considered for this approach are well-structured and have clean constructions that avoid “short circuits” between loops. In this regard, as explained earlier, single-entry and single-exit points are assumed for the loops in a net [110] (cf. Section 3.1.2.4 on block-structured models). Because of the many existing techniques, approaches, and tools for identifying cycles in Petri nets, the question here is not how to identify cycles,

but rather which parts of the policy should be reset—and in which way—if a cycle is entered.

In what follows, first the general idea of re-enactment, and how such a “re-enactment block” can be nested will be presented. Then this idea will be generalized and formalized in the course of the “flattening of policy re-enactment” into a SecANet. Since the previous modeling steps of a SecANet already encode certain information, the modeling of re-enactment for constraints and authorizations focuses on the SecANet, and not on the policy states in the original policy-aware WF-net. This allows for building on the knowledge already encoded in the acyclic modeling.

3.2.5.1 Points and Scopes of Policy Re-Enactment

From a control-flow perspective, cycles or loops are self-explanatory. In contrast to acyclic workflow structures, cyclic workflows typically contain workflow tasks that lie on cyclic paths and can therefore occur multiple times. From an organizational point of view, however, this raises the question of what effects cycles in the control flow can have on policy aspects and vice versa. Here, the current policy encoding in a SecANet would allow execution of each task only once, so that it would block tasks that are to be executed again. Therefore, a cyclic path demands that the parts of the policy involved in it be re-enacted in such a way that users can eventually be assigned again after a cycle is entered. For this, the concept of “re-enactment points” will be introduced. This supports “forgetting” previous user assignments and re-assigning users to tasks that lie in the scope of a cyclic path for a specific policy part.

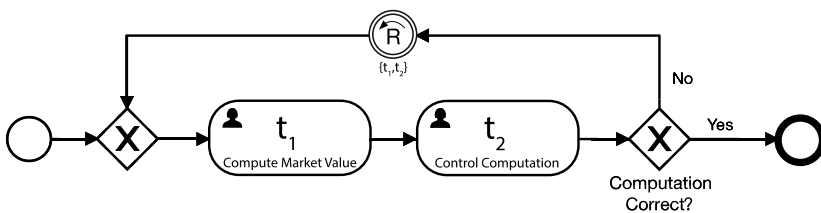


Figure 3.37 DMV with loop and re-enactment point $\widehat{R}_{\{t_1, t_2\}}$

A “re-enactment point” is considered as an event in the workflow execution that triggers the re-enactment of the policy for a specific scope of tasks. It allows fine-grained modeling and control of the position in the workflow at which the

re-enactment is triggered. Figure 3.37 proposes the notation of such an event in the context of a BPMN notation of the DMV process. Thus a re-enactment could even be triggered for acyclic nets in order to force policy enactment to forget user-task assignment decisions that had already been made, for example on certain exclusive paths. In this regard, the proposed policy re-enactment can be related to the notion of “release points” [24], which intends to erase the execution history for certain constraints as well. Release points are defined directly within the SoD or BoD constraints. Re-enactment points, in contrast, focus on the tasks affected by the policy and are not part of the constraint definition. This allows them to easily be considered for the authorization policy as well. Just as with a release point, a re-enactment point explicitly defines its scope, while a loop only implicitly defines the scope to which it applies. Hence for cyclic behavior a re-enactment point can be used to explicitly state the loop-entry point for the entire block of tasks involved in the loop. Indeed, it is assumed that re-enactment points are explicitly stated and defined in the course of the model and policy design, because there are many approaches that can be used to identify the entry and exit point of a loop, or the looping block. These approaches address the identification of (the smallest) cycles, cyclicity, and repetitiveness in Petri nets or WF nets—for example, with the help of the reachability graph, the so-called T-invariant, or more sophisticated techniques [22, 74, 75, 155]. Since the boundedness of a net is decisive in the identification of cycles, for unbounded nets becomes more complex. For free-choice nets, in contrast, cycles can be identified efficiently. Based on the results of the identification of cycles, the re-enactment points involving the scopes of workflow tasks covered by each cycle could then be added right after the entry point of the cycle. Thus re-enactment points provide different possibilities for steering and control of the execution of a security-aware workflow. On the one hand, a re-enactment point could be used to explicitly trigger

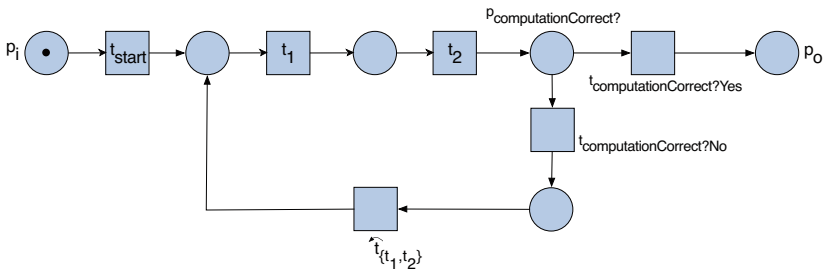


Figure 3.38 DMV Petri net with loop and re-enactment transition $\hat{t}_{\{t_1, t_2\}}$

a policy re-enactment for the scope of a cycle when its loop entry point is passed. On the other hand, a conscious omission of a re-enactment point after loop entry could be used to force taking a path that executes tasks that have not yet been assigned.

Thus re-enactment points are characterized by their position and the scope they define, which determines the tasks that are to be re-enacted. Here, to preserve the basic functionality of a SecANet, it is assumed that the definitions of the re-enactment points are reasonable. For example, if a re-enactment point is entered after loop entry, it should cover all tasks involved in the loop in order to prevent manipulation of workflow behavior. Since the actual position of such a point in the workflow structure has an impact on the execution [39], the position of the re-enactment point is assumed to be reasonable. For example, if a loop is entered, the corresponding re-enactment point is supposed to occur right after loop entry. If a re-enactment point is intended to release a certain scope of tasks, it needs to be triggered right before that scope. Otherwise, re-enactment points at arbitrary positions may negatively impede the behavior of the SecANet, since they quickly become confusing and may cause unintended dependencies between the positions of the re-enactment points and the workflow execution. Thus to ensure a clear structure and preserve the encoding, a re-enactment point for acyclic SecANet (parts) may occur only right before or right after the scope to which it applies.

Re-Enactment Transitions: Analogously to the general concept of a “re-enactment point” in a process, “re-enactment transitions” for Petri nets are introduced. The workflow net in Figure 3.38 depicts the WF-net of the BPMN example in Figure 3.37. It contains the re-enactment transition $\widehat{t}_{\{t_1, t_2\}}$, which defines the re-enactment point and the scope of tasks that are to be re-enacted. In general, \widehat{t}_S denotes a transition that triggers re-enactment of a given scope of tasks $S \subseteq T$.

Definition 3.66 (Re-Enactment Transitions and Scopes). *Given a SecANet N , let \widehat{t}_S be a transition that represents a re-enactment point, and let $S \subseteq T$ be the corresponding scope that defines a set of tasks for which the policy is supposed to be re-enacted. Then the set of all re-enactment transitions in a WF-net is denoted by \widehat{T}_S . The non-empty set of all scopes S of all transitions triggering re-enactment is denoted by a family of sets, $\mathcal{S}_{\widehat{T}} = \{S_1, \dots, S_{|S|}\}$.*

For example, given the three different scopes $S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4\}$, and $S_3 = \{t_1, t_2, t_3, t_4, t_5\}$, the three re-enactment tasks can be denoted by \widehat{t}_{S_1} , \widehat{t}_{S_2} , and \widehat{t}_{S_3} (equivalently, as $\widehat{t}_{\{t_1, t_2\}}$, $\widehat{t}_{\{t_3, t_4\}}$, and $\widehat{t}_{\{t_1, t_2, t_3, t_4, t_5\}}$), hence $S_{\widehat{T}} = \{\{t_1, t_2\}, \{t_3, t_4\}, \{t_1, t_2, t_3, t_4, t_5\}\}$.

Cancellation and Enactment: Based on such a re-enactment transition, an example of a Petri net construction for two tasks t_i and t_j , similar to the example DMV SecANet, is used to sketch the idea of how the policy for a certain scope can be re-enacted. As soon as re-enactment is triggered, tokens are supposed to be consumed and produced in such a way that the initial state of the considered policy encoding is restored. The bright upper part in Figure 3.39 shows the idea of how the re-enactment is implemented. First, an “enter re-enactment” transition produces a token in the “cancellation control” place. In order to avoid self-loops, the cancellation transitions are connected to a further place whose outgoing transition also produces a token in “cancellation control.” It is assumed that there is no obstruction, since that would block the execution of the tasks and no loop could be carried out. Therefore, both user-task assignments are considered to be executed, hence in this example there are no leftover tokens in the places of the user-task encodings. Thus the cancellation needs to consider only the SoD-related encoding here. For every SoD place, a transition that can consume leftover tokens is added (cf. the “cancel” transitions). After all leftover tokens in the regarded places are erased, the “enact” transition can be fired to put one token back into the initially marked SoD and user-task assignment places. To ensure that this “enactment” can be performed only when all corresponding SoD places are empty and to avoid having two tokens in an SoD place, their capacity needs to be explicitly bounded by 1. Based on the previously introduced “complementary-place transformation” [155], complementary places can be used to achieve this. They are labeled with the place they are complementing, followed by “_c1c” (capacity-1 complement). Moreover, arcs are added to the transitions which are connected to the regarded places. Here, the sum of tokens for each pair that consists of an SoD place and its place-complement is always equal to the capacity (which is 1 in this case) before and after executing the regarded transitions. While the purpose of Figure 3.39 is to explain the basic idea of re-enactment, the idea of cancellation and enactment will be formalized in a more fine-grained way. For example, the fulfillment of the condition that all leftover tokens are erased before the “enact” transition can produce tokens in the initially marked places will be partitioned into multiple steps. This higher modularity will allow the cancellation and enactment to be controlled separately and will increase the traceability of the individual steps of the net execution.

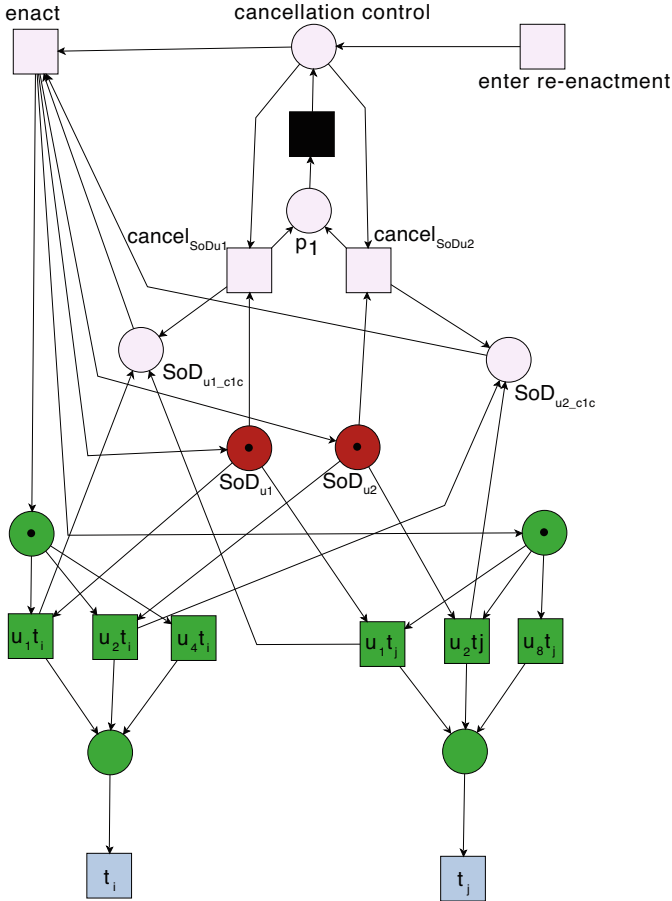


Figure 3.39 Basic idea of policy re-enactment exemplified for SoD constraints

Besides the points of re-enactment and its encoding with Petri net elements, it is key to consider the bigger picture that takes all re-enactment points and scopes defined for a workflow into account. More precisely, the creation of Petri net elements that realize re-enactment (i.e., the creation of “re-enactment constructs”) needs to consider the cancellation and enactment, as well as the fact that scopes may potentially be part of other scopes. Here, straightforward construction of the re-enactment encoding for each of these scopes could result in multiple

constructions of re-enactment encodings that involve the same scope, so that parts of the re-enactment constructs would be built multiple times for the same user-task authorizations or constraints. On the one hand, in order to keep the modeling complexity as low as possible, unnecessary, redundant constructions must be avoided. On the other hand, such multiple constructions could interfere with each other in an unintended way and manipulate the overall re-enactment encoding. For example, based on an SoD place, as in Figure 3.39, if there were multiple cancellation constructs for the cancellation of the same SoD place, the firing of the cancellation transition would then depend on tokens in the cancellation control in both constructs. The cancellation would in turn depend on the triggering of both re-enactment points at the same time, which may not be the case in the control flow. Hence the construction of a re-enactment must be done in a way that preserves the initial net behavior and does not result in such unintended manipulations. Here, creating re-enactment constructs for each user-task authorization and each constraint separately may avoid this. Re-enactment points could then trigger these individual constructs according to their scope of tasks. However, then all net elements controlling each cancellation and enactment would entail a high modeling complexity. Therefore, the re-enactment encoding needs to consider the interdependencies between scopes as well. Analogously to the block structure of the workflow net—in particular, block-structured loops—the scopes of re-enactment relate to blocks. Hence besides demanding an appropriate position of the re-enactment points, it is reasonable to ensure that their scopes are block structured as well. Here, there arises the question of how the creation of the re-enactment construct can be used by multiple re-enactment points such that nested scopes can be re-enacted. Moreover, how must the creation of the re-enactment constructs take place in order to successively allow for nesting of all scopes according to the block structure?

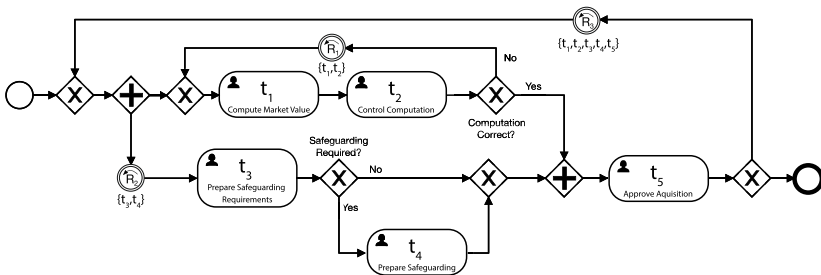


Figure 3.40 CEW re-enactment points

Nesting of Re-Enactment Scopes: Based on the block structure of the workflow, it is assumed that re-enactment scopes are nested only blockwise. Accordingly, Figure 3.40 extends the CEW workflow with re-enactment points and three scopes: $S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4\}$, and $S_3 = \{t_1, t_2, t_3, t_4, t_5\}$. It can be observed that the scopes that are nested are proper subsets of larger scopes; for example, $S_3 = S_1 \cup S_2 \cup \{t_5\}$. Figure 3.41 illustrates the structural idea of how a re-enactment of nested subscopes is embedded into the re-enactment of a scope. Here, the marked place \widehat{p}_S indicates the state in which re-enactment has been triggered. A completed re-enactment for a given scope is indicated by a complete circle in which the symbol of the scope is embedded, that is, a marked place p_{\odot} . Accordingly, t_{\odot} finishes the re-enactment for scope S . Based on this, Figure 3.41 illustrates how to connect the “re-enactment construction” for a given re-enactment transition. For this, the “enter re-enactment” transition is extended by producing a token in each place of the re-enactment constructs for some subscope Y of S . Similarly, each place of the connected subsopes is connected to the “leave re-enactment” transition, hence the re-enactment is not allowed to be finished until after all parts of the scopes are finished. Thus all “re-enactment blocks” that are involved in the transitions of the regarded scope are embedded in parallel. Then placing tokens in the places that trigger re-enactment for their scopes initiates different cancellations and enactment blocks. Moreover, this construction allows for checking whether a block is already nested, namely, if \widehat{t}_S isn't the only incoming transition of the place \widehat{p}_S , that is, $|\bullet\widehat{p}_S| > 1$. Analogously to the idea of cancellation and enactment, Figure 3.41 only the basic idea. In order to allow for higher modularity and clear execution steps, these steps will be formalized in a more fine-grained manner below.

Creation of Re-Enactment Constructs: There is not only the question of how to structure and nest blocks, but also how to conduct the creation of all the Petri net re-enactment elements so as to reflect and preserve the block structure. Analogously to the block structure, scopes can be seen as blocks that nest each other. A scope that nests another scope represents a superscope. A scope that is embedded in a larger scope represents a subscope. Again, depending on which scope is chosen to begin the creation of re-enactment constructs, a scope could be built before its subscope elements are built. The creation of a re-enactment construct for the subsopes either would result in multiple re-enactment constructs from the same policy parts or would require restructuring of the re-enactment constructs already created. To avoid such a cumbersome approach, a systematic procedure for building and connecting the scopes of re-enactment one after the other will be subsequently developed. Here, it is worthwhile to consider the effects of the block structure on the individual

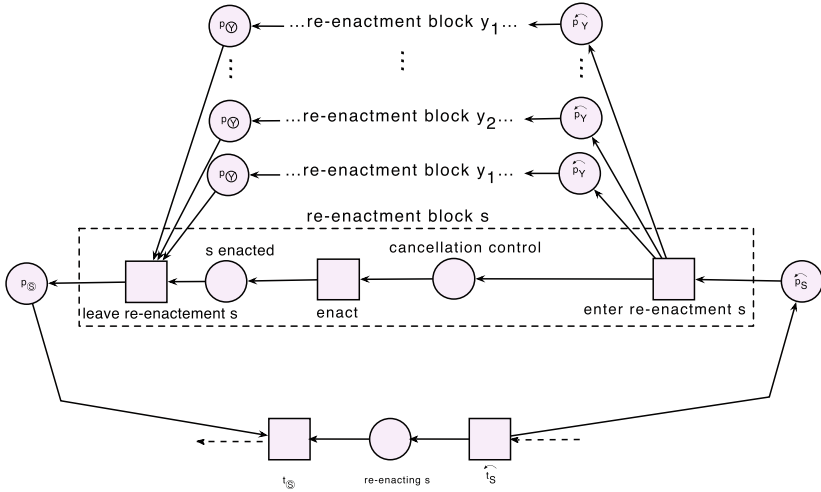


Figure 3.41 Nesting of re-enactment blocks in parallel

scopes—in particular, how they nest each other. The nesting implies that the nested scopes represent proper subsets of a superscope; for example, $S_3 = S_1 \cup S_2 \cup \{t_5\}$ (as indicated before). Obviously, the set of scopes over some set of workflow tasks represents a subset of the power set of these tasks, that is, $S \subset \mathcal{P}(T)$. Here, the block structure of the affected models represents a subset of the standard partial order on the power set of T . For instance, the power set of the set of five tasks with the set of scopes $S \subset \mathcal{P}(T)$ from Figure 3.40 highlighted in bold is

$$\begin{aligned} \mathcal{P}(T) = \{ & \emptyset, \\ & \{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}, \{t_5\}, \\ & \{\mathbf{t_1}, \mathbf{t_2}\}, \{t_1, t_3\}, \{t_1, t_4\}, \{t_1, t_5\}, \{t_2, t_3\}, \\ & \{t_2, t_4\}, \{t_2, t_5\}, \{\mathbf{t_3}, \mathbf{t_4}\}, \{t_3, t_5\}, \{t_4, t_5\}, \\ & \{t_1, t_2, t_3\}, \{t_1, t_2, t_4\}, \{t_1, t_2, t_5\}, \{t_1, t_3, t_4\}, \\ & \{t_1, t_3, t_5\}, \{t_1, t_4, t_5\}, \{t_2, t_3, t_4\}, \{t_2, t_3, t_5\}, \\ & \{t_2, t_4, t_5\}, \{t_3, t_4, t_5\}, \{t_1, t_2, t_3, t_4\}, \{t_1, t_2, t_3, t_5\}, \\ & \{t_1, t_2, t_4, t_5\}, \{t_1, t_3, t_4, t_5\}, \{t_2, t_3, t_4, t_5\}, \\ & \{\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}, \mathbf{t_4}, \mathbf{t_5}\}. \end{aligned}$$

It can be seen directly that only scopes of smaller cardinality, for example $|S_1| = 2$, can be part of scopes of higher cardinality, for example $|S_3| = 5$. Hence if a scope is nested in another scope, this automatically implies that its superscope must have a higher cardinality. For example, if a block of a task t_1 is nested in another scope, this scope must contain t_1 but also at least one additional element. Otherwise, it would constitute just the scope t_1 itself. Consequently, scopes with the same cardinality cannot be subsets of each other, that is, scopes cannot nest other scopes of the same cardinality. Moreover, the nesting excludes the possibility that some smaller scope is directly nested in multiple scopes of higher cardinality. (Equivalently, scopes of the same cardinality cannot nest the same subscope.) Also, a task cannot be directly nested in more than one scope of the next higher cardinality. For example, the scopes $\{t_1, t_2\}$ and $\{t_2, t_3\}$ would have the overlapping task t_2 and would not adhere to the considered block structure. Such overlapping scopes are problematic, since they could allow for re-enactment of a single scope to be triggered multiple times at once from different re-enactment points that involve the same subscope. If that were to happen, multiple tokens would be placed into the re-enactment construct, which would contradict the condition-event principle on which the safe SecANet encoding is built, diffuse the meaning of markings and the firing of transitions, and manipulate the intended behavior of the re-enactment constructs. Hence scopes may not overlap with other scopes. In case some scope contains a task of another scope, it must completely nest the scope in which this task occurs, hence a scope can be directly nested in only a single superscope. Such a superscope can then itself be embedded in a single scope of higher cardinality.

These considerations on the block structure, which manifest in the cardinality of scopes, will be useful in the construction of the policy re-enactment. Accordingly, the creation of re-enactment constructs will be built with increasing cardinality of scopes, that is, from the minimal to the maximal set of the family of sets S . For each cardinality, the corresponding re-enactment constructs are supposed to be created and potentially to nest smaller scopes. After the maximal sets of scopes have been considered, the creation of the re-enactment constructs is completed, hence all re-enactment points will be flattened into the SecANet step by step.

3.2.5.2 Flattening Policy Re-Enactment

Algorithm 3.1 encodes the overall framework of policy re-enactment and controls the creation of the policy re-enactment constructs. Thereby, the order in which the re-enactment is encoded is given by the increasing cardinality of the scopes (see line 7).

Algorithm 3.1 POLICY RE- ENACTMENT FRAMEWORK(SecANet)

Input: $(N_{TA+C}, \widehat{T}_S)$ \triangleright The SecANet N_{TA+C} resulting from Definitions 3.30, 3.31, and 3.32 and its re-enactment points \widehat{T}_S with $S = \{S_1, \dots, S_{|S|}\}$ and $S \subseteq T$.

Output: $N_{TA+C+\widehat{T}_S}$ \triangleright The SecANet $N_{TA+C+\widehat{T}_S}$ encoding policy re-enactment.

1: $N \leftarrow N_{TA+C}$

2: $i \leftarrow 1$, with $i \in \mathbb{N}$. \triangleright i defines the cardinality of the scopes under consideration.

3: **repeat**

4: **for each** $\widehat{t}_S \in \widehat{T}_S$ with $|S| = i$ **do**

5: Create the net elements for \widehat{t}_S (i.e., a re-enactment transition whose scope S has cardinality i), and place them into the SecANet N by successively applying Definitions 3.67, 3.68, 3.69, 3.70, and 3.71.

6: **end for**

7: $i \leftarrow i + 1$

8: **until** $i > |T - \widehat{T}_S|$ \triangleright i may not exceed the number of WF tasks.

9: **return** N

The subsequent definitions will relate to a re-enactment transition of a SecANet N that contains all re-enactment transitions and their corresponding scopes (as specified on line 5). The following is an overview of these definitions.

- Creation and Nesting of Re-Enactment Controls (Definitions 3.67 and 3.68)
 - Cancellation and Enactment (Definition 3.69)
 - User-Task Cancellation and Enactment (Definition 3.70)
 - Constraint Cancellation and Enactment (Definition 3.71)

First, the basic Petri net elements that control the beginning and the end of re-enactment for a certain scope will be created.

Definition 3.67 (Creation of Re-Enactment Controls) *Based on a re-enactment transition \widehat{t}_S and its scope S , the re-enactment for S is controlled by placing the following net elements into the SecANet N , where*

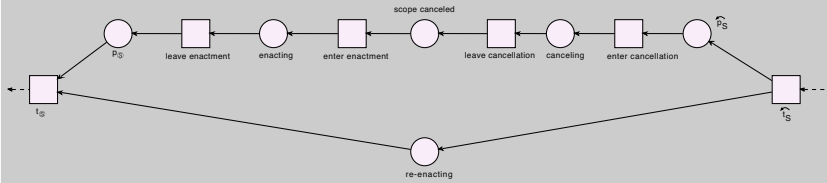
$$N = \langle P \cup P_S, T \cup T_S, F \cup F_S, m_0 \rangle \text{ with}$$

$$P_S = \{\widehat{p}_S, P_{S_{\text{Re-Enacting}}}, P_{S_{\text{Canceling}}}, P_{S_{\text{Canceled}}}, P_{S_{\text{Enacting}}}, P_{S_{\otimes}}\},$$

$$T_S = \{t_{S_{\text{EnterCancellation}}}, t_{S_{\text{LeaveCancellation}}}, t_{S_{\text{EnterEnactment}}}, t_{S_{\text{LeaveEnactment}}}, t_{S_{\otimes}}\},$$

$$F_S = \{(t_{S_{\otimes}}, p_{\text{PostRe-Enact}}) | p_{\text{PostRe-Enact}} \in \widehat{t}_S^\bullet\} \cup \{(\widehat{t}_S, P_{S_{\text{Re-Enacting}}}), (P_{S_{\text{Re-Enacting}}}, t_{S_{\otimes}}), (\widehat{t}_S, \widehat{p}_S), (\widehat{p}_S, t_{S_{\text{EnterCancellation}}}), (t_{S_{\text{EnterCancellation}}}, P_{S_{\text{Canceling}}}), (P_{S_{\text{Canceling}}}, t_{S_{\text{LeaveCancellation}}}), (t_{S_{\text{LeaveCancellation}}}, P_{S_{\text{Canceled}}}), (P_{S_{\text{Canceled}}}, t_{S_{\text{EnterEnactment}}}), (t_{S_{\text{EnterEnactment}}}, P_{S_{\text{Enacting}}}), (P_{S_{\text{Enacting}}}, t_{S_{\text{LeaveEnactment}}}), (t_{S_{\text{LeaveEnactment}}}, P_{S_{\otimes}}), (P_{S_{\otimes}}, t_{S_{\otimes}})\}.$$

Note that $\{(t_{S_{\otimes}}, p_{\text{PostRe-Enact}}) | p_{\text{PostRe-Enact}} \in \widehat{t}_S^\bullet\}$ is supposed to take over the initial outgoing arcs of \widehat{t}_S , hence that now $\widehat{t}_S^\bullet = \{p_{\text{Re-Enacting}}, \widehat{p}_S\}$. The figure below shows the graphical representation of these net elements:¹⁴

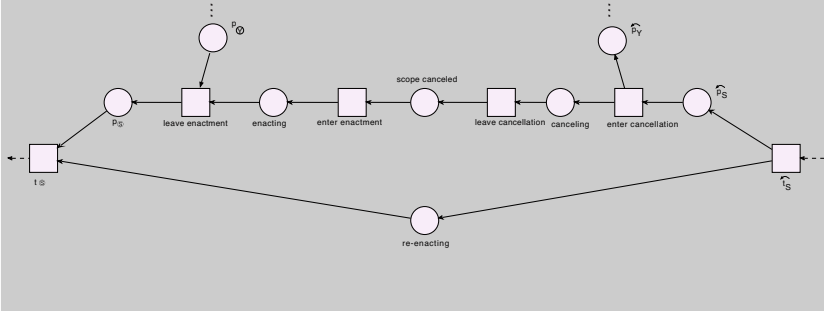


As indicated before, it must be checked whether there are nested subsopes for the scope under consideration. Because of the incremental construction method, which is based on the increasing cardinality of scopes, it must be ensured that the block structure is gradually built into the re-enactment construct already created in N . Here, based on the re-enactment control created for each transition \widehat{t}_S , it must be checked whether its scope S involves other scopes.

Definition 3.68 (Nesting of Re-Enactment Controls). Based on the re-enactment control created for a transition \widehat{t}_S from Definition 3.67, for each place of N of the form $\widehat{p}_Y \in P$ where $Y \in S$ it must be checked whether its scope Y is a subset of the considered scope of S of \widehat{t}_S , that is, $Y \subset S$. Based

¹⁴ For better clarity and readability, the naming of the depicted nodes is simplified compared to their written notation; for example, the scope S is omitted in the naming of the transition.

on the nesting defined in this definition, if $|\bullet \widehat{p}_Y| > 1$ then Y is already embedded in another block. Since scopes may not be nested in more than one block, the construction described in this definition may be built only if $|\bullet \widehat{p}_Y| = 1$. Then the scope Y is not only triggered by its corresponding re-enactment transitions \widehat{t}_Y but also controlled with the help of arcs pointing from the current scope S to \widehat{p}_Y and back, that is, the set of arcs of N is extended by $\{(t_{S\text{EnterCancellation}}, \widehat{p}_Y), (p_{\otimes}, t_{S\text{LeaveEnactment}})\} \cup F$. The figure below shows the graphical representation of this nesting of re-enactment controls:¹⁵



In this way, the basic framework in which the cancellation and enactment of the specific policy parts can take place is built. In the next step, the construction that controls the cancellation and enactment for a specified scope of tasks is explained. It will be controlled by firing the respective “enter” and “leave” transitions for the cancellation and enactment. Accordingly, the subsequent Petri net construct will be the frame which is used to control cancellation and enactment.

Definition 3.69 (Cancellation and Enactment). Let S be a scope of transitions, and let $T_{S\text{cancel}} = \{t_{S\text{EnterCancellation}}, t_{S\text{LeaveCancellation}}\}$ and $T_{S\text{enact}} = \{t_{S\text{EnterEnactment}}, t_{S\text{LeaveEnactment}}\}$ be the sets of transitions in N that trigger cancellation and enactment, respectively. The cancellation

¹⁵ The scopes nested in the scope under consideration are triggered by the arcs containing \widehat{p}_Y . All subscope-related re-enactment constructs that are not already nested in other re-enactment constructs are nested in parallel to the re-enactment construct of the scope under consideration.

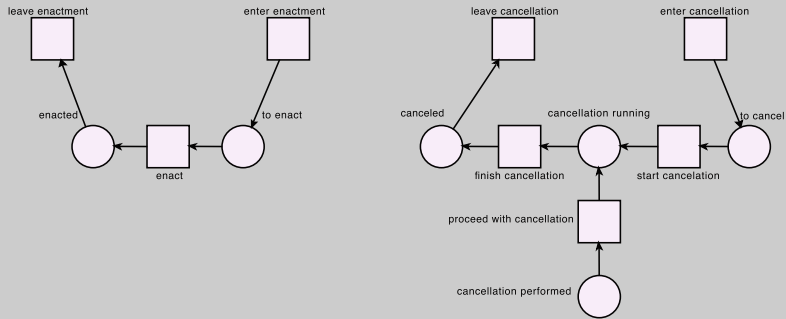
and enactment are controlled by adding the following net elements to the SecANet N , where $N = \langle P \cup P_S, T \cup T_S, F \cup F_S, m_0 \rangle$ with

$$P_S = \{p_{S_{ToCancel}}, p_{S_{CancellationRunning}}, p_{S_{CancellationPerformed}}, p_{S_{Canceled}}, p_{S_{ToEnact}}, p_{S_{Enacted}}\},$$

$$T_S = \{t_{S_{StartCancellation}}, t_{S_{ProceedWithCancellation}}, t_{S_{FinishCancellation}}, t_{S_{Enact}}\},$$

$$F_S = \{\langle t_{S_{EnterCancellation}}, p_{S_{ToCancel}} \rangle, \langle p_{S_{ToCancel}}, t_{S_{StartCancellation}} \rangle, \langle t_{S_{StartCancellation}}, p_{S_{CancellationRunning}} \rangle, \langle t_{S_{CancellationPerformed}}, p_{S_{ProceedWithCancellation}} \rangle, \langle t_{S_{ProceedWithCancellation}}, p_{S_{CancellationRunning}} \rangle, \langle p_{S_{CancellationRunning}}, t_{S_{FinishCancellation}} \rangle, \langle t_{S_{FinishCancellation}}, p_{S_{Canceled}} \rangle, \langle \rangle, \langle p_{S_{Canceled}}, t_{S_{LeaveCancellation}} \rangle, \langle t_{S_{EnterEnactment}}, p_{S_{ToEnact}} \rangle, \langle p_{S_{ToEnact}}, t_{S_{Enact}} \rangle, \langle t_{S_{Enact}}, p_{S_{Enacted}} \rangle, \langle p_{S_{Enacted}}, t_{S_{LeaveEnactment}} \rangle\}.$$

The figure below shows the graphical representation of these net elements:



The “cancellation running” and “cancellation performed” places will be the incoming and outgoing places, respectively, for each transition that will actually allow canceling (i.e., consuming) of leftover tokens, which will be illustrated in Definitions 3.70 and 3.71. Moreover, if all involved places that indicate a completely cleared policy construct are marked, they are supposed to enable the transition that produces a token in a place that indicates that the cancellation is completed. The enactment-related Petri net elements will then need to put only the initial tokens into the places of the corresponding policy net elements.

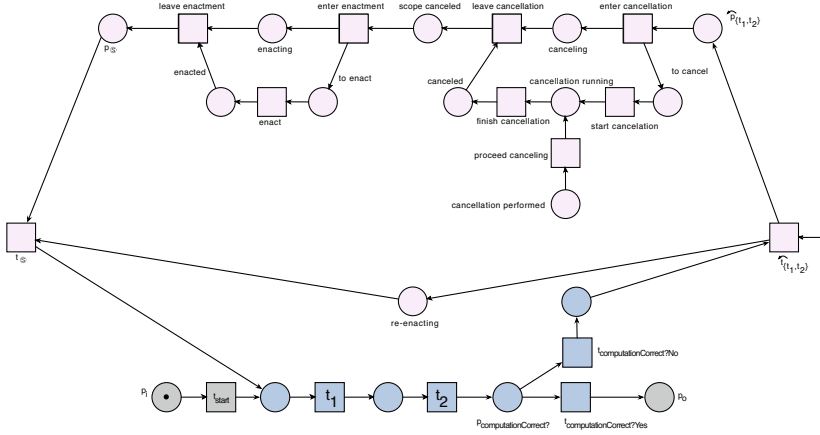


Figure 3.42 DMV example with re-enactment, cancellation, and enactment controls

The example in Figure 3.42 applies the definitions given so far to the re-enactment transition in a DMV SecANet. Since up to now only the control flow of the DMV example is affected by the re-enactment elements, the SecANet constructs that encode user-task authorization and the SoD constraints are omitted for better readability.

The constructed re-enactment-related Petri net elements now constitute the complete framing in which the actual re-enactment is supposed to occur. The next definitions will connect the elements of this frame with the policy-related parts of the SecANet—in particular, with the cancellation and enactment of user-task authorization and constraints, respectively.

Given a user-task authorization, indicated by its initially marked place p_t from a SecANet, the cancellation is as follows: First, a place that indicates whether the task has been executed, that is, an outgoing place p_t of the transition t , is added. Thus it is now possible to cover all three possible states that can occur in the course of a user-task assignment by the three places t_- , t_+ , and p_t . As previously observed, in the course of the language-related initial and final markings, and because of the given sequential order, it is always the case that only one of the three places can be marked. Here, for each such state a cancellation transition is needed. All three cancellation transitions have the same outgoing place, indicating that all places related to the user-task authorization are empty. The following definition represents

the general encoding of how to clear all possible token distributions in a user-task authorization construct and how to enact it. Analogously to the nesting, it will be checked whether this construct already exists, based on the existence of incoming transitions for the place p_- that indicate user-task unassignment. If such incoming transitions exist, this means that the corresponding user-task construct already exists and is part of another scope in which its re-enactment is supposed to be nested.

Definition 3.70 (User-Task Cancellation and Enactment). For each $t \in S$, let $P_{t-+} = \{p_{t-}, p_{t+}\}$ be a set of pairs of the user-task authorization of the SecANet, where $P_{t-+} \subseteq \dot{P}_{TA}$. If $\bullet p_{t-} = \emptyset$, the subsequent elements are added to the SecANet N to cancel the state of the user-task authorization and to enact it. Otherwise (if $|P_{t-}| > 0$), the construct has already been added in a nested scope. Specifically,

$$N = \langle P \cup P_S, T \cup T_S, F \cup F_S, m_0 \rangle \text{ with}$$

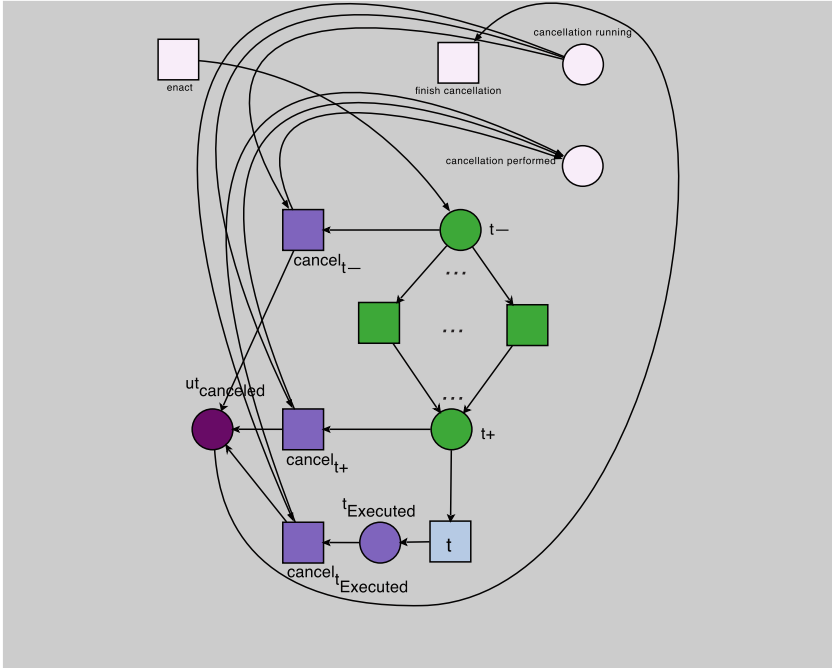
$$P_S = \{p_{t_{\text{Executed}}}, p_{t_{\text{Canceled}}}\},$$

$$T_S = \{t_{\text{Cancel}_{t-}}, t_{\text{Cancel}_{t+}}, t_{\text{Cancel}_{t_{\text{Executed}}}}\},$$

$$F_S = \{ \langle p_{t-}, t_{\text{Cancel}_{t-}} \rangle, \langle t_{\text{Cancel}_{t-}}, p_{t_{\text{Canceled}}} \rangle, \langle p_{t+}, t_{\text{Cancel}_{t+}} \rangle, \langle t_{\text{Cancel}_{t+}}, p_{t_{\text{Canceled}}} \rangle, \langle t, p_{t_{\text{Executed}}} \rangle, \langle p_{t_{\text{Executed}}}, t_{\text{Cancel}_{t_{\text{Executed}}}} \rangle, \langle t_{\text{Cancel}_{t_{\text{Executed}}}}, p_{t_{\text{Canceled}}} \rangle, \langle p_{S_{\text{CancellationRunning}}}, t_{\text{Cancel}_{t-}} \rangle, \langle t_{\text{Cancel}_{t-}}, p_{S_{\text{ProceedWithCancellation}}} \rangle, \langle p_{S_{\text{CancellationRunning}}}, t_{\text{Cancel}_{t+}} \rangle, \langle t_{\text{Cancel}_{t+}}, p_{S_{\text{ProceedWithCancellation}}} \rangle, \langle p_{S_{\text{CancellationRunning}}}, t_{\text{Cancel}_{t_{\text{Executed}}}} \rangle, \langle t_{\text{Cancel}_{t_{\text{Executed}}}}, p_{S_{\text{ProceedWithCancellation}}} \rangle, \langle p_{t_{\text{Canceled}}}, t_{S_{\text{FinishCancellation}}} \rangle, \langle t_{S_{\text{Enact}}}, p_{t-} \rangle \}$$

The figure below shows the graphical representation of these net elements:¹⁶

¹⁶ For better clarity and readability, the naming of the depicted nodes is somewhat simplified compared to their written notation; for example, the scope S is not subscripted.



It is important to note that, based on the re-enactment construction, user-task transitions could occur in a firing sequence in which they are not followed by their respective tasks. For the interpretation of the encoding, therefore, it is important to note that only the last user-task assignment before the corresponding task execution encodes the actual user that executes the task.

Regarding constraints, for a given constraint place C in a SecANet, the cancellation is as follows: First, a complementary place for C is created. Then each outgoing transition from the constraint place C needs to be connected to the new complementary place as its incoming transition. Moreover, a new transition that has the constraint place as its incoming place and the complementary place as the outgoing place is created. Thus if during workflow execution the constraint is taking effect, this state is captured by the complementary place. Otherwise, the cancellation transition may erase leftover tokens during execution of the loop. Since both of these cases result in an empty constraint place C , only one place is used to document this. For example, Figure 3.44 depicts such a cancellation construct for an SoD place. Hence a marked complementary place represents the state in which the correspond-

ing constraint place is empty, which may result from the application of the constraint in the course of a user-task assignment or may indicate cancellation.

Definition 3.71 (Constraint Cancellation and Enactment). *Let $C_{ut_k t_l}$ be a constraint place abstracting from $SoD_{u_j t_k t_l}$ or $BoD_{u_j t_k t_l}$. For each $C_{ut_k t_l}$ where $t_k, t_l \in S$ and $t_k \neq t_l$, the cancellation of the state of the constraint enforcement and its enactment is provided by adding the net elements indicated below. Again, only if $\bullet C_{ut_k t_l} = \emptyset$ are the subsequent elements added to the SecANet N to cancel the state of the constraint and to enact it. (Otherwise, i.e., if $|\bullet C_{ut_k t_l}| > 0$, this indicates that the construct has already been added in a nested scope.) Specifically,*

$$N = \langle P \cup P_S, T \cup T_S, F \cup F_S, m_0 \rangle \text{ with}$$

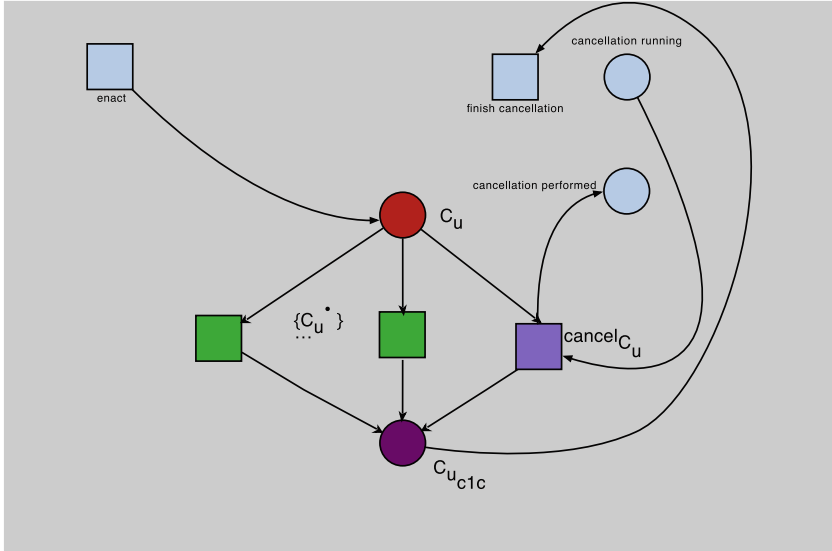
$$P_S = \{p_{C_{ut_k t_l} \text{Complement}}\},$$

$$T_S = \{t_{\text{Cancel}C_{ut_k t_l}}\},$$

$$F_S = \{\langle t_{ut}, p_{C_{ut_k t_l} \text{Complement}} \rangle \mid t_{ut} \in C_{ut_k t_l}^\bullet\} \cup \\ \{\langle C_{ut_k t_l}, t_{\text{Cancel}C_{ut_k t_l}} \rangle, \langle t_{\text{Cancel}C_{ut_k t_l}}, p_{C_{ut_k t_l} \text{Complement}} \rangle, \langle p_{\text{CancellationRunning}}, \\ t_{\text{Cancel}C_{ut_k t_l}} \rangle, \langle t_{\text{Cancel}C_{ut_k t_l}}, p_{\text{ProceedWithCancellation}} \rangle, \langle p_{C_{ut_k t_l} \text{Complement}}, \\ t_{S_{\text{FinishCancellation}}} \rangle, \langle t_{S_{\text{Enact}}}, C_{ut_k t_l} \rangle\}$$

The figure below shows the graphical representation of these net elements:¹⁷

¹⁷ C^\bullet denotes all user-task transitions affected by a constraint. After all the transitions are connected as incoming transitions of $C_{ut_k t_l} \text{Complement}$, the last step is to add the cancellation transition as an outgoing transition of C and an incoming transition of $C_{ut_k t_l} \text{Complement}$.



Based on Algorithm 3.1 and the definitions embedded therein, it is now possible to fully construct the net elements that provide policy re-enactment. Figure 3.43 illustrates the policy re-enactment for the example DMV SecANet. It must be admitted that, for this simple example, the effort for re-enactment seems rather disproportionate. However, this is considered as the “price to pay” to obtain a general and clear encoding for larger nets as well. Here, based on the systematic, incremental construction, the elements of the control constructs increase linearly with the number of re-enactment points and their scopes. Although the overall construct could be simplified (e.g., see Figure 3.39), the aim has been to keep the construction clean and as clearly arranged as possible. Moreover, this general modular construction will allow further usage in other respects (see Section 3.2.6.3). Also, the concept of re-enactment points offers the possibility for refinement. Although re-enactment has applied thus far to tasks in general, it could be refined in terms of policy type or users. In order to prevent multiple access by the same person during particularly critical activities, for example, one could exclude constraints from re-enactment and retain only the SoD constraints that have already been put into effect.

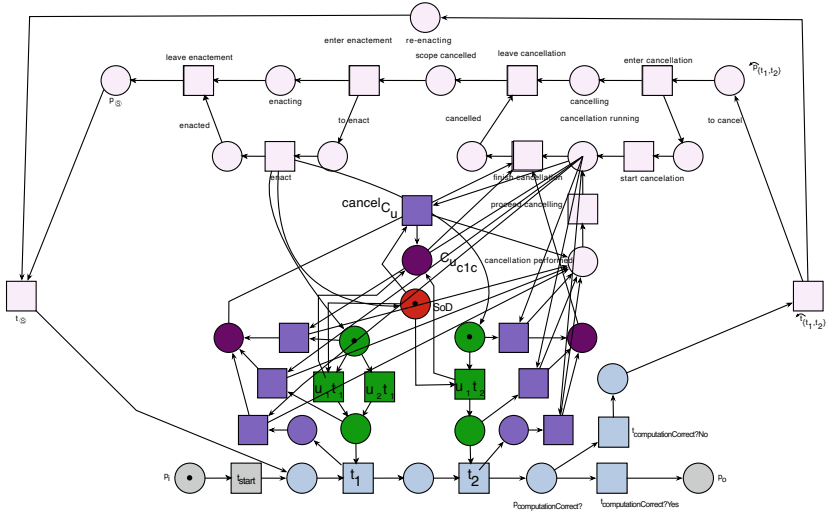


Figure 3.43 DMV loop and re-enactment

Comprehensive SecANet Example of the Collateral Evaluation Process: To show the applicability of the flattening for a larger comprehensive net structure involving cycles, the example of a workflow net for collateral evaluation in Figure 3.20 which is based on the BPMN model that involves release points from Figure 3.19 will be used. The SoD and BoD constraints are noted right in the BPMN

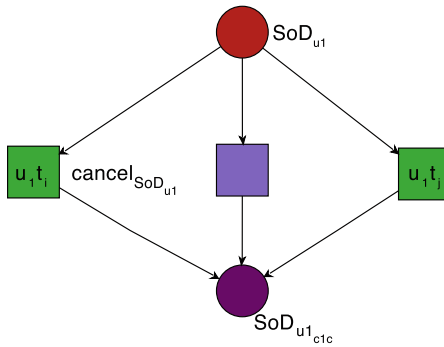


Figure 3.44 SoD cancellation example

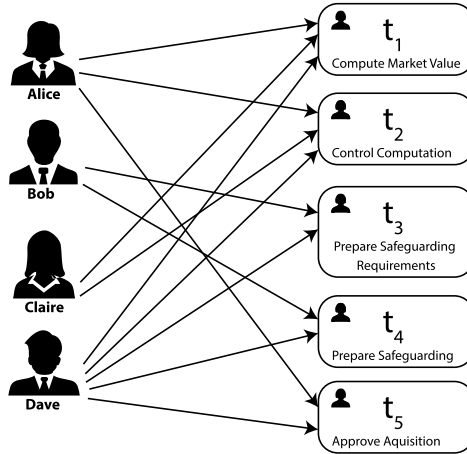


Figure 3.45 User-task assignment

model. The release points (see o_1, o_2, o_3) allow these constraints to be considered in the scope of each loop separately. In order to illustrate that the SecANet encoding is applicable to different models, the intention is to leave the model as is and interpret the release points as re-enactment points for their corresponding scopes, where $S_{o_1} = \{t_1, t_2\}$, $S_{o_2} = \{t_3, t_4\}$, and $S_{o_3} = \{t_1, t_2, t_3, t_4, t_5\}$. Figure 3.45 depicts an example of a corresponding user-task assignment. According to the mapping in Figure 3.18 from Dijkman et al. [78], the BPMN model is first transformed into a P/T net. Then the user-task assignment, the two SoD constraints, and one BoD constraint are flattened into the net according to the theory presented above. Finally, the policy re-enactment constructs for the release points are added. Figure 3.46 depicts the resulting SecANet, with different shades highlighting the individual parts of the net. It is displayed to give an impression of the overall system that results from the flattening. To increase readability, the policy re-enactment considers only simplified re-enactment constructs without user-task cancellation (as, for instance, in Figure 3.39). As previously indicated, if all the tasks are executed, user-task cancellation can be neglected for re-enactment, since the tokens in all user-task assignment places of the form p_{t-} and p_{t+} are consumed in the course of execution. However, this net also contains obstructions, which will be treated in more detail in the subsequent chapters, where approaches that solve obstructions based on the presented flattening are presented.

3.2.6 SecANet Analysis: Satisfiability and Obstructability

Based on Definitions 3.30, 3.31, and 3.32 and Algorithm 3.1, a SecANet can encode a policy-aware WF-net with a comprehensive block structure involving cyclic, exclusive, and parallel paths. Moreover, based on Definitions 3.33 and 3.34, markings that encode obstructed and satisfiable firing sequences are provided. Thus a SecANet can encode security properties and can be interpreted with regard to satisfiability and obstructability. Specifically, the related liveness security property of process completion can be checked by considering all possible execution sequences (or events), which in the end constitute sequences (or traces) that may or may not have that property. As previously illustrated, satisfiable execution sequences may reach markings that contain p_o , that is, the output place of the initial WF-net is marked. If this occurs, it means that there is a full firing sequence (equivalently, that there is a valid plan) such that the workflow is satisfiable. On the other hand, obstructed execution sequences reach terminal markings that do not contain the output place p_o . Hence the liveness security property of process completion can also be checked by examining whether final markings (or states) that either involve p_o or not are reachable.

Hence satisfiability and obstructability are both concerned with reachability in the first place. Satisfiability is concerned with whether there is a marking greater than p_o which is reachable in a SecANet. Obstructability is concerned with whether there is a marking that neither contains p_o nor enables any further transitions. The most basic method for determining satisfiability and reachability would be to play the token game until a desired marking is reached, that is, until the net reaches the obstruction marking or the marking containing p_o . While this is a feasible endeavor for the DMV example, larger net structures demand more systematic approaches. Here, the most straightforward way would be to build the marking graph of the net. The reaching of the end marking would then resolve the workflow as satisfiable, and the reaching of a terminal marking without p_o would reveal an obstructed state. Moreover, as elaborated at the beginning of this chapter, there are many additional techniques which may be used to examine reachability more efficiently. Those techniques will also be highlighted in Chapter 4, which introduces the so-called marking equation and uses integer linear programming techniques to answer questions related to reachability. Next, it is shown how to transform a SecANet in such a way that deadlock detection and liveness analysis can be applied to it, the main objective being to relate the notion of soundness to the SecANet encoding. Thus the analysis of SecANet soundness will subsume the analysis of satisfiability and obstructability.

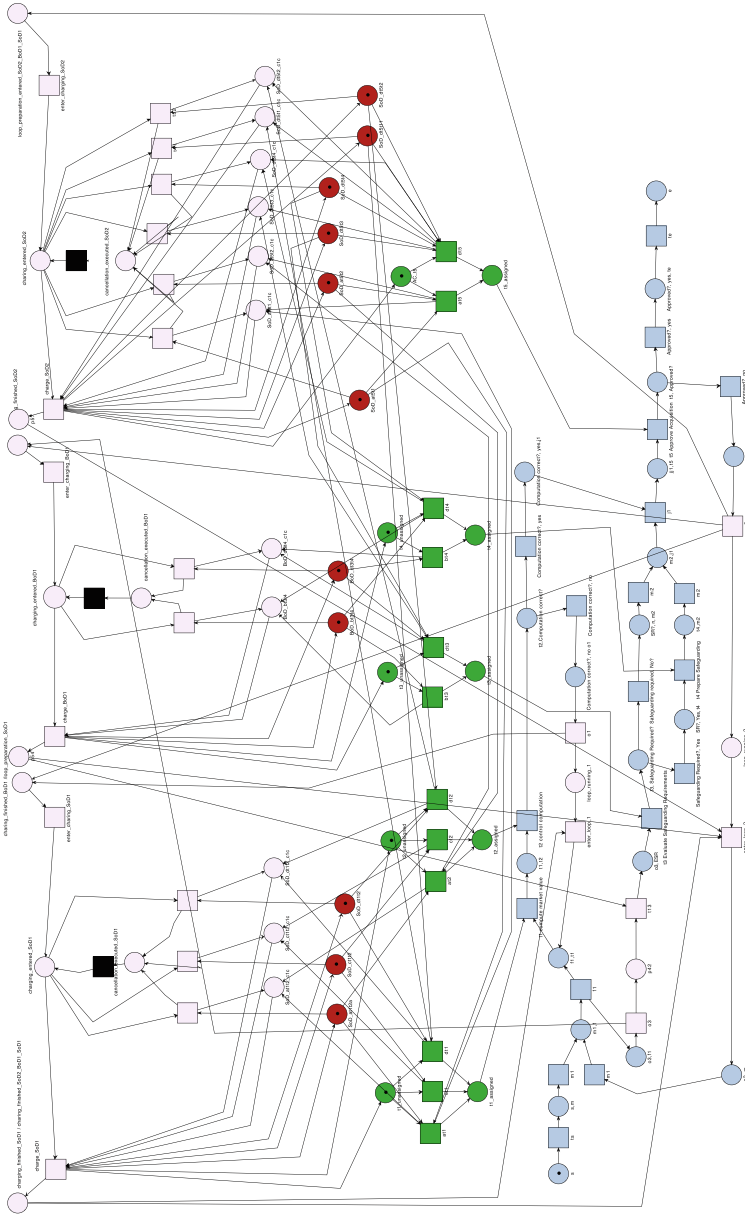


Figure 3.46 The collateral evaluation workflow (cf. Figure 3.20) with the user-task flattening (cf. Figure 3.23a), the SoD or BoD flattening (cf. Figure 3.23b), and re-enactment (bright nodes as in Figure 3.39)

3.2.6.1 SecANet Soundness

A sound SecANet is considered to be satisfiable and obstruction-free. For a consistent evolution of existing terminology and definitions from the literature, the notion of SecANet soundness will be built upon the definitions of soundness of a workflow net. As mentioned in Section 3.1.2.4, a sound workflow represents a workflow that is able to reach the output place from every reachable marking (i.e., it has the “option to complete,” meaning that every task is involved in the processing of a case). Moreover, all of its tasks need to be executable at least once (referred to as the “no dead transitions” condition). Finally, when the output place is marked, no other places are supposed to be marked (denoted as “proper completion”). From the perspective of access control, an obstruction is also a case that cannot be processed to its completion because the policy is blocking further progress. Satisfiability relates to the executability of each task. If a certain workflow task is not executable in any possible execution sequence, the workflow path that involves this workflow task is not satisfiable. Here, this means that the policy does not allow the firing of a user-task assignment that would enable the regarded workflow task. Hence in order to check satisfiability and obstructability, the notion of “soundness” of a workflow will be tailored to soundness of a SecANet. Since, as previously elaborated, it cannot be ensured that the marking in p_o is the only marked place left after a SecANet execution, the “proper completion” property will be neglected for the moment.

The subsequent interpretation of the “option to complete” and the “no dead transitions” property will assume that the workflow for which a SecANet is created is block structured and sound (cf. Section 3.1.2.4). This implies that the workflow net within the SecANet fulfills the “option to complete” and has no dead transitions. Thus it suffices to analyze whether a sound workflow net is still sound in the context of the SecANet encoding surrounding it. If the SecANet encoding cannot fulfill the conditions or soundness, the reason for this has to lie in the net elements created in the course of the flattening that encode the policy.

Similarly to the definition from workflow soundness, the “option to complete” is supposed to check whether it is possible to reach a marking that involves p_o . Thus the “option to complete” condition for a sound SecANet encodes obstruction-freeness for every possible execution sequence (or case). If a case is not completable, it represents an obstructed case, that is, the “option to complete” of the initially sound workflow is no longer fulfilled in the SecANet.

As examined before, obstructions describe individual process executions (or cases) that are not completable. They may occur in satisfiable or unsatisfiable workflows. If all possible cases for a specific path of workflow tasks result in an obstruction (i.e., the path is not completable for any of the possible user-task assignments),

this means that the path is not satisfiable at all. Hence if the “option to complete” is not fulfilled at least once for a certain path, the transitions that are always blocked never become enabled, that is, they are dead. Thus the question of whether a workflow is satisfiable at all can be answered by checking for dead workflow tasks in a SecANet. Put differently, if all WF tasks are simply live (equivalently, quasi-live), the workflow is satisfiable. For workflows with sequential or parallel paths, a dead task then implies that the entire workflow is unsatisfiable. In workflows with exclusive branching, a dead task means that only the path on which this task occurs is unsatisfiable. They may also include satisfiable executions. Regardless of the actual structure of the workflow, therefore, the SecANet workflow will be considered unsatisfiable as soon as there are dead workflow tasks, since the policy does not allow full execution of all possible paths given by the initially sound workflow. It is assumed that the “no dead transitions” property for a SecANet has to hold only for the workflow tasks, not for user-task assignments (which are all simply live based on the initial marking) or for transitions that serve only functional routing purposes. This is because, on the one hand, the original “no dead transitions” condition concerns only the workflow tasks. On the other hand, the question of satisfiability is ultimately also answered by looking at the workflow tasks, namely by determining whether a task is executable or not.

A sound SecANet is thus defined to be obstruction-free and satisfiable if it fulfills the “option to complete” and has “no dead transitions.” These two conditions will

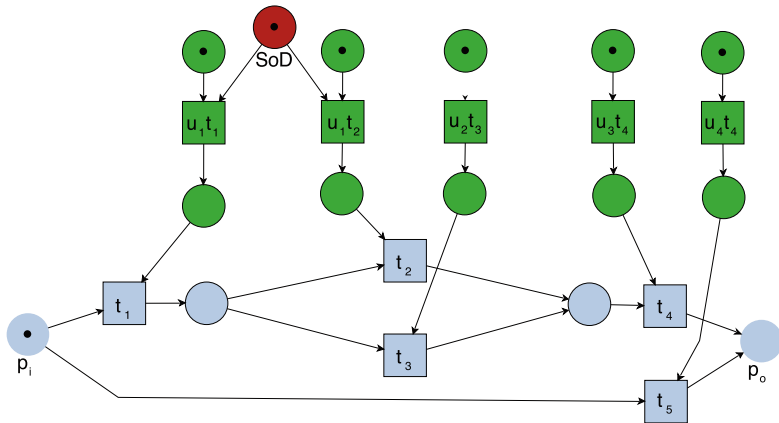


Figure 3.47 SecANet that fulfills the option to complete, with dead transition t_2 and implicit gateways

directly allow observing the connections and implications between satisfiability and obstructability in the context of a specific net. The subsequent definition will adapt the definition of the properties from workflow soundness (cf. Definition 3.72) to the SecANet encoding. Based on the assumption that the underlying workflow net is sound, a SecANet is sound if and only if the following two requirements are satisfied:

Definition 3.72 (Soundness of a SecANet) *A SecANet N with workflow input place i and workflow output place o is sound if the following conditions are met:*

- (i) *Obstruction-Freeness (or Option to Complete): For each case and every marking reached from the initial marking, it is possible to reach a state that marks the end place o ; that is, $\forall m \in R(N, i), m(o) \in R(N, m)$.*
- (ii) *Satisfiability (or No Dead WF Task): It is possible to execute an arbitrary WF task by following the appropriate firing sequence through the WF-net; that is, $\forall t \in T, \exists m \in R(N, i)$ such that $(N, m)[t]$.*

In the example, the previously identified obstruction marking in Figure 3.24 indicates that it is not always possible to complete the DMV SecANet workflow (i.e., the “option to complete” is not fulfilled). However, the DMV SecANet does not contain dead WF transitions, which underlines its previously identified satisfiability. Here, removing u_2 from the user-task permissions for t_1 , for example, would render the last WF-task(t_2) dead because the workflow would not be satisfiable.

In this regard, Figure 3.47 illustrates the potential pitfalls of the interpretation and interconnections between the conditions of SecANet soundness. Supposedly small differences in the interpretation of workflows on the Petri net level can impact the obstructability analysis. At the same time, the Figure underlines the importance of exactly how gateways are modeled as well as how rather coarse-grained modeling may cause a loss of important security-related information. Figure 3.47 depicts a SecANet that has the option to complete the workflow but contains a dead transition, t_2 . The ensuing interpretation that the workflow is obstruction-free but not satisfiable would, however, contradict the identified relationships between satisfiability and obstructability. In this regard, a closer look at the net reveals that it implements forks and joins in a rather reduced, implicit way. For instance, firing transition t_1 or t_5 at the beginning of the net determines which path to take and, at the same time, executes the actual task following this decision. If transition t_1 is obstructed, the other path (the

one that starts with t_5) can be chosen instead. Because of the unconstrained user-task assignment represented by $t_{u_4t_5}$, the latter is a path that obviously cannot obstruct. For this reason, the obstruction does not block the process. However, it still reduces the option of which path to choose and restricts the behavior of the initial workflow. Hence in order to have full control over the net in a fine-grained way and avoid such “hidden” obstructions, a workflow has to be modeled according to the block structure and the common transformation from BPMN into Petri nets (cf. Figure 3.18). Each fork and join is supposed to be resolved as a separate construct that does not involve tasks. Figure 3.48 encodes the workflow with explicit joins and forks according to the common transformation of gateways into Petri nets (e.g., from BPMN). Checking the net in Figure 3.48 for the two soundness conditions reveals the dead transition t_2 too, but the related obstructions at transitions t_1 and t_2 come to light as well. This is thus consistent with the identified implications between the “option to complete” and “no dead transitions” condition and their relations to obstructability and satisfiability. First, the decision to take a certain path has to be made irrespective of any user-task assignment. After this decision, a potential obstruction cannot be circumvented. Thus potential constellations in which certain tasks cannot be performed because of authorization constraints cannot be undermined in such a way that weak spots of the policy can be identified again. Therefore, the workflow involved in a SecANet must explicitly encode forks and joins for parallel and exclusive gateways that do not depend on any user-task assignments of the policy encoding. The user-task encoding in a SecANet must be created only for actual workflow tasks that are supposed to be executed by users. Then an unsatisfiable workflow cannot be obstruction-free, since the explicit modeling of gateways implies that it must be possible to explicitly choose the path that involves the dead transition, and thus to obstruct the execution.

Hence a workflow may have “no dead transitions” and no “option to complete,” so that it is satisfiable but obstructive, or it may have “no dead transitions” and the “option to complete,” in which case it is satisfiable and obstruction-free. A dead transition means that, somewhere on the path before the transition, there is an obstruction that does not allow the transition to be enabled. Hence in a SecANet the option to complete cannot be fulfilled in the face of dead transitions. Either there is an obstruction that directly causes the workflow transition to be dead, or there may be an obstruction at some earlier point in the execution that deadens all succeeding tasks. In regard to the latter, and regardless of whether a user can be assigned to a task or not, the control-flow token needed to enable the workflow task would then be missing for the succeeding tasks. A SecANet that neither has the option to complete nor has no dead transitions is neither satisfiable nor obstructionfree.

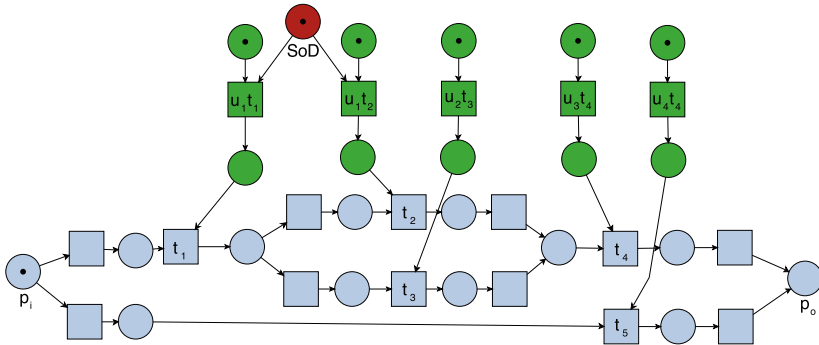


Figure 3.48 SecANet that does not fulfill the option to complete, with dead transition t_2 and explicit gateway modeling

3.2.6.2 SecANet Reversibility and Deadlock Analysis

Both of the soundness conditions can again be represented as typical questions of reachability. “No dead transitions” asks if there is a transition that is not fireable from any reachable marking, while “option to complete” asks whether the final marking is reachable from any reachable marking. Because of the structural complexity of a SecANet, which, as previously explained, belongs at least to the class of asymmetric choice nets, it may be intractable to make a decision on its soundness [7, 136]. In contrast to reachability, as elaborated before, the computation of liveness as well as the identification of deadlocks can potentially be done more efficiently.

In a SecANet deadlocks are used to indicate the two crucial states. A satisfiable full firing sequence reaches a deadlock marking that contains p_o . A SecANet that encodes an obstructed execution reaches a deadlock marking without p_o . Thus the idea is to exclude from deadlock analysis the desirable end marking that encodes process completion and contains p_o . This is done by allowing the end marking to restore the initial marking, so that the net becomes reversible. Thus similarly to the WF-net extension related to the investigation of the strong-connectedness of a WF-net in Definition 3.22, an additional transition that connects the output place with the input place needs to be added. Since the reaching of the output place can no longer represent a deadlock, a deadlock is then supposed to indicate an obstruction. In a sense, the “option to complete” can thereby be “included” in the structure of the net. This inclusion allows searching for deadlocks beyond the desired deadlock of the marked output place. In such an extended net, the identification of deadlocks then becomes synonymous with the search for obstructions, thereby allowing for

the use of typical existing methods for deadlock analysis—for example, checking for the siphon/trap property (cf. Definition 3.15 and Definition 3.16).

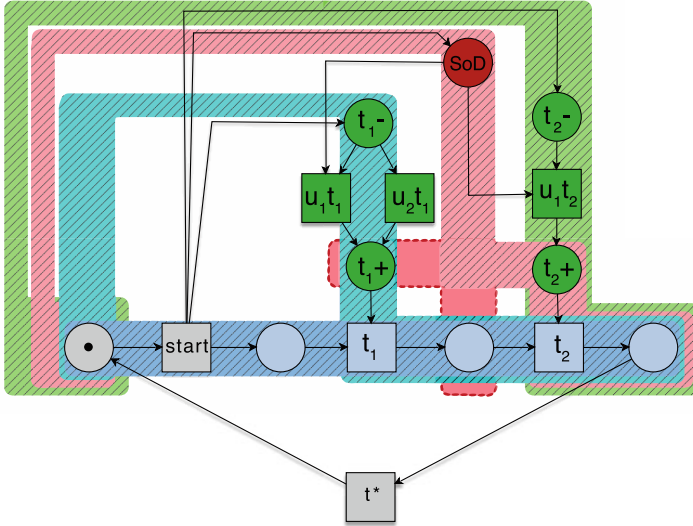


Figure 3.49 Obstructability analysis exemplified with deadlock analysis (siphon/trap)

Because of the simplicity of the DMV SecANet example, it can be used to illustrate the intended extension, which is depicted in Figure 3.49. It is realized by adding a transition t^* whose outgoing transition is the input place of the SecANet. It can now be reset to its initial marking by firing t^* . Moreover, t_{start} is added to enact the policy. Now, the siphon-trap property can be analyzed to identify deadlocks within the extended net, that is, whenever each siphon contains a trap the net is deadlock-free. The siphons and traps for the extended DMV SecANet are indicated with different-shaded bars. Without the extension, the siphons and traps would trivially be the source and sink places, respectively. Hence it can be observed that, concerning the workflow net and its user-task authorization, each siphon contains a trap. The related siphons and traps are completely in cover (which is highlighted by the dashed lines). However, the siphon that involves the SoD place does not contain the trap that contains the SoD place. The SoD-related trap encompasses all elements of the siphon but additionally contains p_o and p_3 . Based on this observation, it can be checked whether the SoD-related siphon can indeed become empty, namely if

there is a marking m where $m(p_0) = 0$ and $m(p_3) = 0$. This is just the case with the obstruction marking. The trap property is fulfilled here too, because either place of the SoD-related trap (p_1 or p_3) is still marked. Moreover, the extended example SecANet could be used to assess the liveness of the extended net, and hence check its soundness as well. Liveness checks typically first check for no dead transitions, and then check whether each transition is reachable from some reachable marking. If there are no dead transitions and the option to complete is fulfilled, each transition can be reached by any transition in the extended net, that is, the entire net is live.

As for the marking that contains the output place, it is important to note that the example net has an advantage in regard to deadlock and liveness analysis: As soon as the output place of the workflow is marked by a token, all other places of the example SecANet are empty. However, this is usually not the case in a SecANet. For example, in case there was another user authorized for both tasks, one of the SoD places would inevitably remain marked at the end of execution. Therefore, for a SecANet in general, before the net would be reinitialized with its initial marking, care must be taken to ensure that all places are empty again. Otherwise, leftover tokens could manipulate the results of the deadlock or liveness analysis. This can be achieved by introducing an additional construction, which uses the re-enactment constructs introduced in Section 3.2.5. If, at the end of execution, eventually only p_o is marked, that is, it is properly completed, this will then allow for an easy transition from this place directly to p_i through a single transition t^* .

3.2.6.3 SecA-WF-Net: Security-Aware Workflow Net

While thus far only the “option to complete” and “no dead transitions” conditions could be related to a SecANet, the modular definition of the re-enactment constructs will subsequently allow for fulfilling the “proper completion” property as well. Since these three conditions exactly constitute the soundness of a WF-net, the SecANet will be extended to a so-called SecA-WF-Net, which represents a general way to extend a SecANet that contains the SecANet encoding as well as a WF-net structure. Moreover, with the help of the additional transition t^* , a SecA-WF-Net can then easily be extended in order to use it for the analysis of deadlocks or liveness.

Figure 3.50 depicts this overall frame of a SecA-WF-Net and how it is supposed to encapsulate a SecANet. It places the SecANet into a further frame that is supposed to enact and cancel the entire policy. The workflow structure demands that there be only one input place and one output place, and that the input place be the only marked place in the initial marking. Therefore, the enactment construction from before is used here to generate the initial tokens of a SecANet based on the initially marked source place p_i . Moreover, in order to allow for “proper completion” the final cancellation construct of the re-enactment modeling above can now be used to

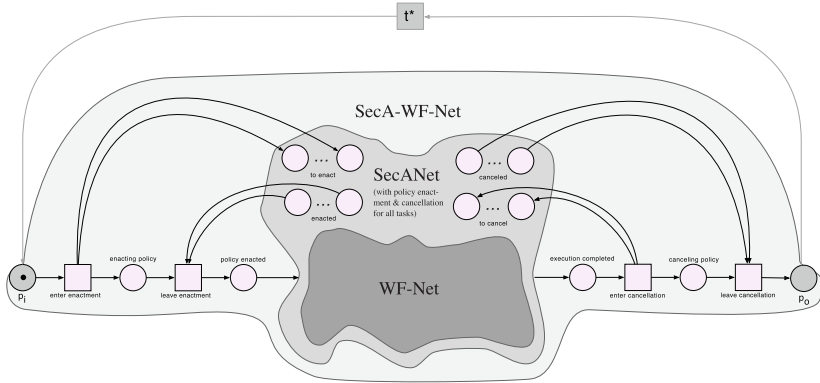


Figure 3.50 SecA-WF-Net encapsulation and extension with t^*

reach the final marking in which only p_o is marked. Figure 3.50 also indicates the optional additional transition t^* that allows for extension of the net in order to use it for deadlock or liveness analysis. This extension makes the net reversible, since the SecANet system can then get initial settings after it is executed, that is, the initial marking m_0 is reachable from every marking of $[m_0]$ (cf. Definition 3.14).

In order to allow for enactment and cancellation of the entire policy, two different cases must be considered. The first is the possibility that there already exist re-enactment constructs, at least for parts of the policy. Similarly to the re-enactment construction above, here again creation of multiple enactments or cancellation constructs must be avoided. Therefore, all places that trigger and document enactment and cancellation need to be connected to the respective “enter/leave enactment” transitions and cancellation transitions, respectively. However, the existence of re-enactment points does not mean that their scopes cover all WF tasks. It could even be the case that the WF-net of the SecANet contains no re-enactment points. Hence for all WF tasks that are not covered by re-enactment constructs, the corresponding construct needs to be created for the respective policy parts. If the SecANet contains no re-enactment points, the construct will need to be created for the entire policy. The modularity of the re-enactment constructions introduced before allows for simply re-using the existing re-enactment definitions for the given sets of enter/leave enactment transitions and cancellation transitions, respectively. Accordingly, the subsequent definition is supposed to generate the workflow structure surrounding a SecANet in order to obtain a SecA-WF-Net.

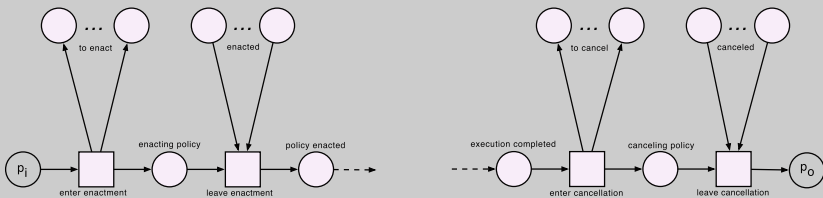
Definition 3.73 (WF-Structure Generation and SecA-WF-Net) Let N be a SecANet that contains a WF-net with an input place p_i and an output place p_o , and let S be its scope involving all WF tasks. In order to control the enactment and cancellation of the policy for the set of all WF tasks $S \subseteq T$ according to the WF structure, the following net elements are added to the SecANet N , where $N = \langle P \cup P_S, T \cup T_S, F \cup F_S, m_0 \rangle$ with

$$P_S = \{p_{EnactingPolicy}, p_{PolicyEnacted}, p_{ExecutionCompleted}, p_{CancelingPolicy}\},$$

$$T_S = \{t_{S_{EnterEnactment}}, t_{S_{LeaveEnactment}}, t_{S_{EnterCancellation}}, t_{S_{LeaveCancellation}}\},$$

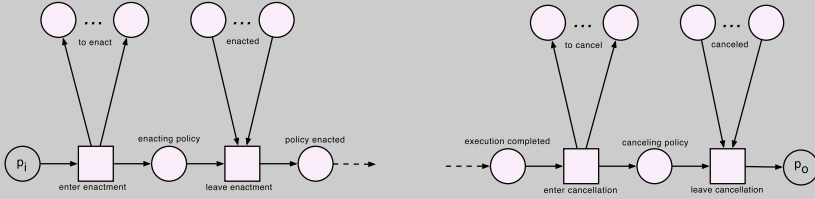
$F_S = \{ \langle p_{PolicyEnacted}, t_{PostInputPlace} \rangle | t_{PostInputPlace} \in p_i^\bullet \} \cup \{ \langle t_{PreOutputPlace}, p_{ExecutionCompleted} \rangle | t_{PreOutputPlace} \in \bullet p_o \} \cup \langle p_i, t_{S_{EnterEnactment}} \rangle, \langle t_{S_{EnterEnactment}}, p_{EnactingPolicy} \rangle, \langle p_{EnactingPolicy}, t_{S_{LeaveEnactment}} \rangle, \langle t_{S_{LeaveEnactment}}, p_{PolicyEnacted} \rangle, \{ \langle p_{ExecutionCompleted}, t_{S_{EnterCancellation}} \rangle, \langle t_{S_{EnterCancellation}}, p_{CancelingPolicy} \rangle, \langle p_{CancelingPolicy}, t_{S_{LeaveCancellation}} \rangle, \langle t_{S_{LeaveCancellation}}, p_o \rangle \}$, and $m_0 = \{p_i\}$ (i.e., the initial markings of all policy encodings are set to 0).

Note that $\{ \langle p_{PolicyEnacted}, t_{PostInputPlace} \rangle | t_{PostInputPlace} \in p_i^\bullet \}$ is supposed to take over the initial outgoing arcs of p_i , and $\{ \langle t_{PreOutputPlace}, p_{ExecutionCompleted} \rangle | t_{PreOutputPlace} \in \bullet p_o \}$ is supposed to take over the initial incoming arcs of p_o , where now $p_i^\bullet = \{t_{S_{EnterEnactment}}\}$ and $\bullet p_o = \{t_{S_{LeaveCancellation}}\}$. The figure below shows the graphical representation of these net elements:



Based on this, all enactment and cancellation places are connected with the enter/leave enactment and cancellation transitions, where the set of arcs of N is $\{ \langle t_{S_{EnterEnactment}}, p_{Y_{ToEnact}} \rangle, \langle p_{Y_{Enacted}}, t_{S_{LeaveEnactment}} \rangle, \langle t_{S_{EnterCancellation}},$

$\langle PY_{ToCancel} \rangle, \langle PY_{Canceled}, t_{SLeaveCancellation} \rangle \setminus \langle PY_{ToEnact}, PY_{Enacted}, PY_{ToCancel}, PY_{Canceled} \in P, Y \subseteq T \rangle \cup F$. The figure below shows the graphical representation of this enactment and cancellation of the different scopes for which there are re-enactment constructs in N :



For the set of WF tasks for which there is no enactment and cancellation place, that is, the scope $\{S - \bigcup_{PY_{Enacted} \in P} Y \mid Y \subseteq T\}$, the re-enactment-related net elements need to be created and added to N by successively applying Definitions 3.69, 3.70, and 3.71, which results in the SecA-WF-Net N .

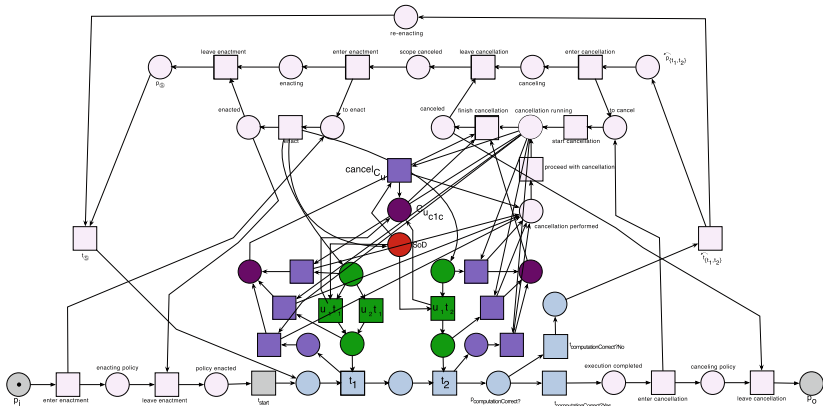


Figure 3.51 DMV with loop and WF structure

The example in Figure 3.51 illustrates the SecA-WF-Net of the DMV SecANet. Again, similarly to the loop and re-enactment constructs, the workflow-structure-generating construction is intended only to provide the structure of a WF-net and,

at the same time, preserve the given functionality and encoding of a SecANet. Hence again, the intention in these constructions is that the related transitions could be set to silent, and yet the basic behavior, meaning, and interpretation of the SecANet would remain. Based on this workflow-structuredness, the soundness of a SecA-WF-Net can now be defined in relation to all three workflow soundness conditions. In addition to SecANet soundness, now the “proper completion” condition can also be included. In a SecA-WF-Net, “proper completion” means the extinction of all tokens in all places except the token in p_o . Accordingly, the soundness of a SecA-WF-Net that contains a sound workflow net can be defined.

Definition 3.74 (Soundness of a SecA-WF-Net) *A SecA-WF-Net N with input place i and output place o is sound if the following conditions are met:*

- (i) *Obstruction-Freeness (or Option to Complete): For each case and any marking reached from the initial marking, it is always possible to reach a state that marks the end place o ; that is, $\forall m \in R(N, i), m(o) \in R(N, m)$;*
- (ii) *Proper Completion: If o is marked, all other places of the SecA-WF-Net are empty for a given case; that is, $\forall m \in R(N, i)$, if $o \in m$ then $m = \{o\}$;*
- (iii) *Satisfiability (or No Dead WF Tasks): It is possible to execute an arbitrary WF task by following the appropriate firing sequence through the WF-net; that is, $\forall t \in T, \exists m \in R(N, i)$ such that $(N, m)[t]$.*

Analogously to SecANet soundness, the soundness of a SecA-WF-Net then implies obstruction-freeness and satisfiability. For the extension of the SecA-WF-Net, the transition t^* is added as the outgoing transition of p_o and as the incoming transition of p_i , hence there are two additional arcs, $\langle p_o, t^* \rangle$ and $\langle t^*, p_i \rangle$ (as indicated in Figure 3.50). The extended SecA-WF-Net can be used to check for deadlock-freeness or liveness of the workflow tasks. Analogously to the findings in [7], a SecA-WF-Net is sound if the extended SecA-WF-Net is live and bounded. Since the boundedness of a SecANet—specifically, its safeness—has been considered in all modeling steps so far, a SecA-WF-Net is assumed to be safe as well. A SecA-WF-Net that is not bounded violates the SecANet encoding and is not interpretable as such. Moreover, it is assumed that the WF-net encapsulated in a SecANet is a sound (and safe) workflow net.

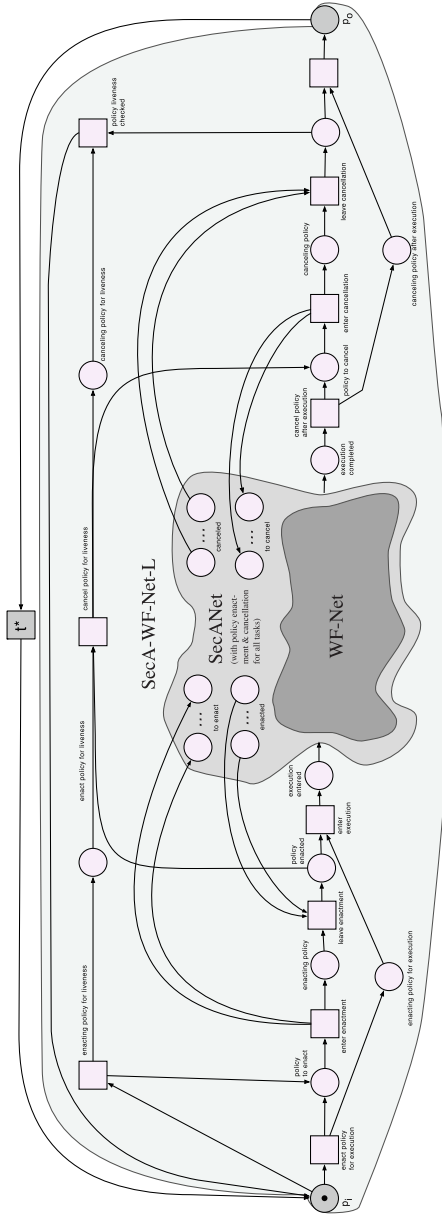


Figure 3.52 SecA-WF-Net that can be used to check for liveness

Hence in order to determine the soundness of a SecA-WF-Net, first and foremost its liveness has to be determined. It is important to note that so far, liveness could be considered only for the actual workflow tasks. Moreover, a SecANet can contain cancellation transitions that never become enabled (i.e., they are dead). For example, the cancellation of a user-task assignment may never require the cancellation of the unassigned place p_{u-} or the assigned place p_{u+} , since the corresponding task is always executed. In contrast to that, liveness of the overall net results from the fact that the net has no dead transitions, that is, every transition is live. This means that every transition of a net can be infinitely enabled through some feasible sequence of firings from any marking in $[m_0]$ (cf. Definition 3.11). Figure 3.52 illustrates how a SecA-WF-Net could be extended to allow checking for the liveness of the entire net. Thus methods of analysis of workflow soundness could be applied to SecANet soundness more easily. The liveness of the extended SecA-WF-Net could then be checked with no restrictions and without having to consider workflow tasks only.

Applicability of Reachability, Deadlock, and Liveness Analysis: A SecA-WF-Net can be analyzed by applying methods from WF-net-related analysis and tools examining WF-net soundness—or, more generally, from reachability analysis. Moreover, the extended SecA-WF-Net allows the use of rather efficient methods that investigate the liveness of its workflow tasks as well as the deadlock-freeness of the entire net. Thus similarly to WF-nets, an extended SecA-WF-Net allows for the use of standard Petri-net-based analysis tools to decide its soundness [7]. For example, off-the-shelf model checkers can be used to analyze the liveness of its workflow tasks.

3.2.7 Experimental Evaluation

This section will demonstrate the applicability of Petri net analysis tools to the SecANet approach and compare the results to a typical existing approach for solving the workflow satisfiability problem (WSP). The SecANet model will be used to show how Petri net tools can be applied to examine obstructability as well. Thus it will be possible to determine SecANet soundness and draw potential benefits from using Petri nets to analyze satisfiability and obstructability.

Since it is difficult to acquire real-world WSP instances [48, 206], the random-instance generator described by Cohen et al. [48] was adapted in order to obtain instances of workflows and their policies. The generator considers a number of tasks t , a number of users u , a number of SoD constraints s , and a random-generator seed

value. Here, since WSP instances usually do not contain exclusive or cyclic paths (cf. Chapter 2), each instance also requires that all tasks be performed in order to complete the workflow execution. Therefore, for the sake of simplicity, the t tasks of each instance are assumed to be performed in sequential order. For each user, the generator creates a uniformly random authorization list $TA(u)$ such that the number of permissions per user, $|TA(u)|$, is selected uniformly from $\{1, \dots, \lceil \frac{t}{2} \rceil\}$ at random. It also generates s distinct not-equals constraints uniformly at random [48]. Then for these instances the SecANet generated according to the definitions in Section 3.2.2. Note that the WSP instance generator does not explicitly consider BoD constraints. Although the SecANet encoding allows the encoding of BoD constraints, for purposes of comparability, BoD constraints will not be considered here. As indicated before, SoD constraints are sufficient to cause obstructed states.

To sketch the runtime behavior of analyzing satisfiability or obstructability, different example nets with an increasing number of tasks, users, and SoD constraints are generated. Figures 3.53 and 3.54 give rough impressions of the smaller instances encoded in the net SecANet representation. A graphical display of larger instances quickly exceeds paper size. In a suitable zoomable digital frame (or on larger paper), a more extensive SecANet is still graphically more intuitive than a purely textual description, since it facilitates understanding of the interdependencies that lead to obstructions.

The experiments were conducted on a MacBook Pro machine, with 8 GB RAM and an Intel Core i7 3 GHz CPU. In order to analyze the WSP instances with a

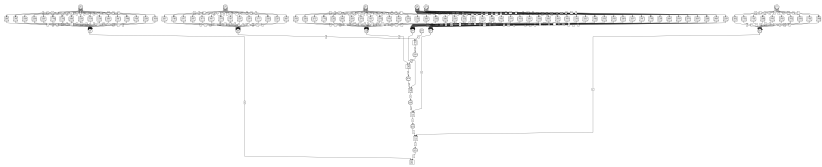


Figure 3.53 Impression of generated SecANet with 6 tasks, 60 users, and 1 SoD constraint. The policy-related elements are arranged horizontally. Below them, the WF tasks flow vertically

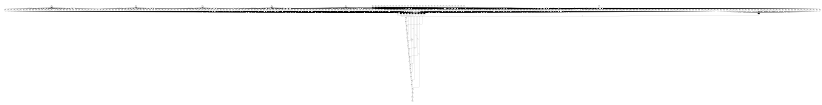


Figure 3.54 Rough impression of generated SecANet with 10 tasks, 100 users, and 4 SoD constraints

common satisfiability analysis technique, they were encoded using pseudo-Boolean (PB) constraints based on Wang et al. [207] and then solved by the PB solver SAT4J [29]. In a SecANet, satisfiability can be examined by checking whether all WF tasks are not dead (cf. Section 3.2.6.1 on SecANet soundness). For this, the Low Level Petri Net Analyzer (LoLa) was used. That is a well-established, state-of-the-art Petri net analysis tool, which constantly participates in the Petri net Model Checking Contest (MCC)¹⁸ and demonstrates its competitiveness with other Petri net verification tools. In order to illustrate the time needed to analyze obstructability, the “option to complete” was considered in the test as well. Table 3.1 shows the results of checking different instances for satisfiability and obstructability¹⁹. The cells with an \checkmark indicate that the PB formula or the respective soundness condition could be satisfied. Cells with an \times highlight unsatisfiable or obstructable instances. In their solutions, both tools offer witness assignments (SAT4J) or witness firing sequences (LoLa) that encode satisfying user-task assignments or cause obstructions. Moreover, since LoLa kills the computation when the computation of the “option to complete” exceeds about 20 minutes, the results for some instances are unknown (the respective results are indicated by a hyphen). It can be seen that the SecANet encoding of the first and fifth WSP instances are sound as well. In terms of computation time, Table 3.1 suggests that the Petri-net-based satisfiability analysis outperforms the analysis of the SAT4J solver by a factor of at least 10. In this regard it should be noted that the SecANet allows examination of the “no dead tasks” property by checking only whether the last task in the execution is not dead. Based on the sequential task order and the considerations regarding the “no dead WF tasks” condition of SecANet soundness, a fireable last WF task renders the entire workflow satisfiable. For the obstructability analysis, the overall rapidly growing time consumption indicates exponential (hence non-polynomial) run-time behavior, which is typical for questions concerning reachability. Checking the “option to complete” condition in a SecANet means a significantly higher level of effort is needed, since from each reachable state it has to be checked whether some marking m that contains the output place, that is, $m \geq p_o$, is reachable.

In conclusion, the tests suggest that the SecANet approach is useful for analyzing satisfiability and that it performs better than the SAT4J alternative. Since there are more elaborate specialized tools for analyzing satisfiability, further approaches could be considered here as well. Moreover, the decreasing runtime for checking the

¹⁸ For further information and tools, see mcc.lip6.fr at the computer-science lab at Sorbonne University, Paris.

¹⁹ The generated process models can be consulted at <https://doi.org/10.6094/UNIFR/228177>. The archive file generated_secanets.zip includes a manual on how to reproduce the results.

Table 3.1 The runtime (in seconds) for analysis of example WSP instances is displayed after the result (i.e., ✓ or ✗). Satisfiability was computed with the SAT4J solver and LoLa, which checked the “no dead transitions” (NDT) condition. For obstructability, LoLa checked the “option to complete” (O2C)

#Tasks/Users/Constraints	SAT4J	NDT (LoLa)	O2C (LoLa)
6/60/1	✓ 0.158	✓ 0.009	✓ 0.008
7/12/21	✗ 0.196	✗ 0.008	✗ 0.090
8/20/11	✓ 0.242	✓ 0.009	✗ 0.213
10/12/22	✓ 0.206	✓ 0.011	✗ 0.011
10/100/4	✓ 0.208	✓ 0.012	✓ 304.0
15/150/10	✓ 0.211	✓ 0.013	–
20/200/19	✓ 0.288	✓ 0.020	–
25/250/30	✓ 0.316	✓ 0.026	–

option to complete for the 10/12/22 instance suggests a more detailed investigation of how a higher number of constraints tends to lower the computational runtime. However, the foremost intention of running the displayed experiment was to show the applicability of Petri net analysis tools to the SecANet approach. To the best knowledge of the author, there is no other approach that solely uses Petri nets to solve satisfiability or obstructability.

3.2.8 Discussion

The SecANet approach provides a holistic basis not only for solving questions concerning satisfiability and obstructability but also for resolving obstructions, which will be examined in subsequent chapters. This chapter has introduced the SecANet encoding, which allows flattening of a security-aware process specification containing the workflow, its corresponding user-task assignments, and its authorization constraints into one Petri net such that obstructions can be captured and analyzed. Therefore, the requirements concerning the obstructability analysis set up in Chapter 2 have been addressed to a large extent. The SecANet encoding integrates common constraints (ROA-4) and a comprehensive workflow structure (ROA-2) and allows for encoding and capture of the state of obstruction (ROA-7). Thus it is possible to do ordered assignments of users, as well as pre-assignments (ROA-1). The SecANet encoding is also able to map costs to each of its elements (ROA-6). Execution sequences of the SecANet can be used to generate satisfiable or obstructed

partial plans (ROA-5). Thereby, the modeling and encoding have been done in such a way that rather efficient techniques—for example, to check the soundness of a net or an extended net—are applicable (ROA-3). Here in particular, the method used in modeling the constraints, the effort to use efficient techniques, and the question of how additional constraints could be taken into account leave room for discussion.

3.2.8.1 Method of Modeling

In general, it can be observed that the modeling of the different elements of the policy integrated onto the control flow manifests itself as “permissive” or “restrictive” modeling. Whereas the modeling of authorization explicitly states what is allowed (permissive), the modeling of constraints implicitly determines what is not allowed (restrictive). In the context of language, analogously to permissive or restrictive modeling, one can also speak of language-extending and language-restricting modeling. For example, the modeling of authorizations adds behavior by adding new letters, while the modeling of constraints restricts existing behavior. Thus the question of whether an added net construct is permissive or restrictive can also be identified by observing its impact on the language.

This distinction between permissive and restrictive modeling also allows for explanation of the differences in the traceability of the policy elements. Whereas the user-task assignment would directly provide reconstruction or retracing of the authorization policy by the user-task transitions, this is possible to only a limited extent for constraints. The proposed method of modeling assumes a set of users and a corresponding user-task assignment such that a constraint can be applied only to user-task assignments that are affected by that particular constraint. A constraint is encoded and retraceable only if there are user-task assignments to which the constraint applies. If there are no user-task assignments affected by a constraint, the SecANet approach does not allow for direct representation of such a constraint in the model. Still, the constraint is implicitly given and does not cause unwanted behavior. This situation is also known as static SoD or static BoD [93]. Hence the visibility and the traceability of constraints in the SecANet model depend on the specific user-task assignments. Analogously, the distinction between “permissive” and “restrictive” applies to the existence or absence of the free-choice property. If a choice between transitions is indeed free, then the choice is explicit, and it is permitted to choose between the options provided. When a choice between transitions is not free, there are further conditions that may restrict the choices. The options can then differ depending on whether these further conditions are fulfilled or not.

3.2.8.2 Time, Space, and Output Memory Requirements

Regarding the computational complexity, in particular, the acyclic SecANet encoding is of interest. The input of the cyclic encoding, in turn, is the output SecANet obtained by the acyclic approach and a set of re-enactment transitions that indicate cycles (cf. Algorithm 3.1). In enhancing the acyclic SecANet with loop-related functionality, the cyclic approach benefits from the net structure already computed by the acyclic encoding. Here, especially the more complex computational compartments of the acyclic approach concerning constraint-related elements will be essential. Moreover, the consideration of acyclic policy-aware workflow nets will allow comparability since related approaches (e.g., regarding the WSP) are typically based on similar input parameters.

For the acyclic SecANet approach that runs on an instance of a policy-aware workflow net (N, U, TA, C) (cf. Definitions 3.29, 3.30, 3.31, and 3.32), the upper bound complexity will be measured in terms of $n = |U|$, $k = |T|$ (i.e., the tasks encoded by transitions in the workflow net N), and $m = |C|$. The set of user-task assignments TA and its corresponding set of assignment list \mathcal{TA} corresponds to k lists each of size at most n (which is, for example, comparable to the WSP instances considered by Cohen et al. [50]).

Time Complexity: Based on the assumption that each task has at least a single user assigned, that is, $|TA| \geq |T|$, a marked place P_{t-} , an unmarked place P_{t+} , and a flow relation $\langle P_{t+}, t \rangle$ are created for each task—which is considered as three computational steps. Analogously, based on each entry in the task-assignment list $TA(t)$, a user-task transition is created together with two arcs that connect P_{t-} and P_{t+} as incoming and outgoing places, respectively, which results in three computing steps for each user-task permission. That way, the complexity to encode the authorization policy is $3 * (|T| + |TA|)$. All possible pairwise combinations of tasks determine the maximal number of constraints, that is, $|C| = \sum_{i=1}^{|T|-1} i = \frac{1}{2}(-1 + |T|) * |T|$. The variable $s \in [0; 1]$ determines the percentage share of SoD constraints of the total number of constraints, such that $|C_{SoD}| = s * |C|$. The proportion of BoD constraints b thus results from $b = 1 - s$, accordingly $|C_{BoD}| = (1 - s) * |C|$. Finding a match between users in the user lists for the two tasks affected by a constraint creates a constraint place and corresponding arcs. In terms of SoD constraints, this means creating the marked SoD place P_{SoD} and the arcs for its two outgoing user-task transitions, thus three steps. It is assumed that the same number of users is assigned to each tasks such that all lists have the same length $\frac{|TA|}{|T|}$. That way, multiplying the number of steps required for comparing each user of each task list with each user of the other task list by the maximal number of

constraints $|C|$ also maximizes complexity for the overall search for matches. For a BoD constraint, a marked constraint place P_{BoD} , a single arc connecting the BoD place with the user-task transition from the user from the list of the one task, and $|\frac{|TA|}{|T|}| - 1$ arcs that connect the BoD place with the user-task transitions from all the other users from the list of the other task are created—so $1 + 1 + \frac{|TA|}{|T|} - 1 = 1 + \frac{|TA|}{|T|}$ steps. If there exist matches for all users in the lists, this means that $\frac{|TA|}{|T|} = |U|$, such that the number of steps needed to find each pair is $\sum_{i=1}^{\frac{|TA|}{|T|}} i = \frac{1}{2} * \frac{|TA|}{|T|} * (\frac{|TA|}{|T|} + 1)$. In case no match is found, the comparison of each entry of one list with each entry of the other list requires $(\frac{|TA|}{|T|})^2$ steps. However, because of $n = |U|$, finding no matches would imply different users assigned to each task (i.e., only $\frac{|U|}{|T|} = \frac{|TA|}{|T|^2}$ users for each task). Then, the comparison of each entry of one list with each entry of the other list only demands $(\frac{|TA|}{|T|^2})^2$ steps. Thus, as the input size increases, the search complexity to find no pairs is less than the complexity required to find matches only. The maximal number of steps for the flattening of all BoD constraints, therefore, results in $\frac{1}{2} * \frac{|TA|}{|T|} * (\frac{|TA|}{|T|} + 1) + (\frac{|TA|}{|T|} * (1 + \frac{|TA|}{|T|})) = \frac{3}{2} * (\frac{|TA|}{|T|})^2 + \frac{3}{2} * \frac{|TA|}{|T|}$, and $\frac{1}{2} * \frac{|TA|}{|T|} * (\frac{|TA|}{|T|} + 1) + (\frac{|TA|}{|T|} * 3) = \frac{1}{2} * (\frac{|TA|}{|T|})^2 + \frac{7}{2} * \frac{|TA|}{|T|}$ for SoD constraints, respectively. The runtime complexity of the SecANet encoding for the overall policy (i.e., the authorization policy and the constraints) results in:

$$(1 - s) * |C| * \left(\frac{3}{2} * \left(\frac{|TA|}{|T|} \right)^2 + \frac{3}{2} * \frac{|TA|}{|T|} \right) \\ + s * |C| * \left(\frac{1}{2} * \left(\frac{|TA|}{|T|} \right)^2 + \frac{7}{2} * \frac{|TA|}{|T|} \right) + 3 * (|T| + |TA|)$$

Although this formula allows considering different $\frac{|TA|}{|T|}$ -ratios, the worst-case assumes that the maximal number of steps is obtained by $\frac{|TA|}{|T|} = |U|$, which results in:

$$(1 - s) * |C| * \left(\frac{3}{2} * |U|^2 + \frac{3}{2} * |U| \right) + s * |C| * \left(\frac{1}{2} * |U|^2 + \frac{7}{2} * |U| \right) + 3 * (|T| + |T| * |U|) \\ \Leftrightarrow (1 - s) * m * \left(\frac{3}{2} * n^2 + \frac{3}{2} * n \right) + s * m * \left(\frac{1}{2} * n^2 + \frac{7}{2} * n \right) + 3 * (k + k * n)$$

That way, $|U|$ determines both the number of users and the number of users assigned to each task. Moreover, assuming $\frac{|TA|}{|T|} = |U|$ takes care of the case that $|TA|$ must not be smaller than $|T|$. Because encoding BoD constraints requires more

steps, the complexity is maximal if $s = 0$ (or $|C| = |C_{BoD}|$), which results in $m * (\frac{3}{2} * n^2 + \frac{3}{2} * n) + 3 * k * (1 + n)$. Hence, the acyclic SecANet encoding is in the class of efficient algorithms that run in quadratic time $O(m * n^2)$. Thus, it belongs to the class of FPT-algorithms $O(f(k) * n^c)$ (cf. Chapter 2). Based on the worst-case assumption that the maximal number of constraints is $m = f(k) = (\frac{1}{2} * (k - 1) * k)$, this results in $(\frac{1}{2} * (k - 1) * k) * (\frac{3}{2} * n^2 + \frac{3}{2} * n) + 3 * k * (1 + n)$ which is in $O(n^2 * k * (k - 1) + (k^2 * n))$. From practice, it can be assumed that the number of tasks usually is an order of magnitude smaller than the number of users [50]. Even the case that $n = k$ results in $\frac{3}{4} * n^4 + \frac{9}{4} * n^2 + 3 * n$ and still is in quartic complexity in the number of users (i.e., $O(n^4)$).

Space Complexity: For all steps, a constant number of iteration variables can be assumed. The authorization-related elements can directly be created by iterating over the task-assignment lists in \mathcal{TA} , which gives the list of users assigned to each task. Moreover, based on each constraint in C involving a pair of tasks, the corresponding lists of users can be obtained again based on the given input. Hence, besides a fixed number of iteration variables, no additional input-dependent space is required to iterate over the lists for two tasks. All generated elements are directly written to the output (which is not read again). Hence, the acyclic encoding is in constant space complexity $O(1)$.

Input/Output Ratio: If one compares, for example, the XES log format mentioned in Chapter 2 with the older MXML log format, the MXML format requires about four times the memory for identical log content. Similarly, based on the memory required for the policy-related part in a policy-aware workflow net, which constitutes the inputs of WSP problem instances, a comparison is possible to the output memory used by the SecANet. However, such a comparison must take into account that the SecANet brings a significant added value compared to the plain input contained in a policy-aware workflow net. First, in a SecANet, there is a place pair $|P_{-+}| = 2$ for each task, thus $|\dot{P}_{TA}| = 2 * |TA|$. Moreover, the number of user-task assignments $|TA|$ in the policy determines the user-task transitions $|\dot{T}_{TA}|$ in a SecANet. Additionally, for each user-task transition, two arcs connect it to the places indicating unassignment and assignment, and there is an arc that connects the corresponding task, which sums to $\dot{F}_{TA} = |TA| * 2 + |T|$. Hence, all Petri net elements related to $|TA|$ result in:

$$|\dot{P}_{TA}| + |\dot{T}_{TA}| + |\dot{F}_{TA}| = 2 * |TA| + |TA| + |TA| * 2 + |T| = 5 * |TA| + |T|$$

Similarly to the maximal runtime complexity, the maximal number of elements is assumed if all users are assigned to all tasks ($\frac{|TA|}{|T|} = |U|$). Then, for large $n = |U|$, the input/output ratio for the authorization-related input $|TA| + |T| = |U| * |T| + |T|$ converges to the factor:

$$\lim_{n \rightarrow \infty} \frac{k + 5 * n * k}{k + n * k} = 5$$

Analogous to the runtime complexity, the worst-case assumption for the output is that all users assigned to the two tasks involved in a constraint are also affected by the constraint. This results in the number of constraint places $|\dot{P}_C| = |C| * \frac{|TA|}{|T|}$ for both cases $\dot{P}_C = \dot{P}_{SoD}$ and $\dot{P}_C = \dot{P}_{BoD}$. The difference here lies in the number of arcs, namely $|\dot{F}_{SoD}| = 2 * |\dot{P}_{SoD}|$ and $|\dot{F}_{BoD}| = \frac{|TA|}{|T|} * |\dot{P}_{BoD}|$, respectively. Hence, the worst-case output memory required for all policy-related elements of the SecANet results in:

$$\left(\left(\frac{|TA|}{|T|} + 1 \right) * (|C_{BoD}| * \frac{|TA|}{|T|}) \right) + \left(3 * |C_{SoD}| * \frac{|TA|}{|T|} \right) + \left(5 * |TA| + |T| \right)$$

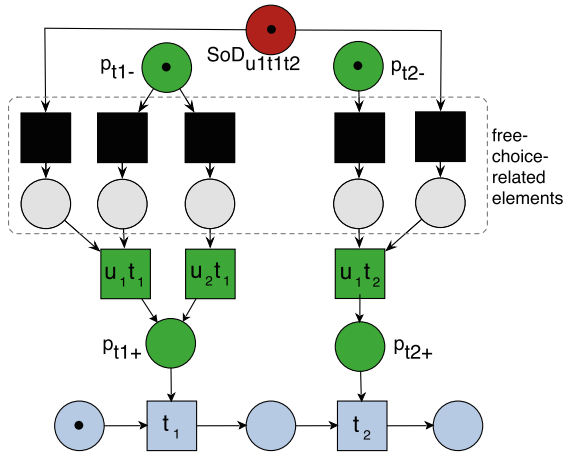


Figure 3.55 DMV free-choice SecANet with initial marking

Here, the worst-case assumption again is that every user is assigned to every task:

$$(1 - s) * |C| * (|U|^2 + |U|) + 3 * s * |C| * |U| + 5 * |U| * |T| + |T|$$

$$\Leftrightarrow (1 - s) * m * (n^2 + n) + s * m * 3 * n + 5 * k * n + k$$

Hence, while the output of authorization-related elements is approximately five times the input size only, the encoding of constraints significantly increases the input size $m = |C|$ by factor $\frac{m*(n^2+n)}{m} = n^2 + n$ (only BoD) and $\frac{m*3*n}{m} = 3 * n$ (only SoD), respectively. Still, in the face of both the pessimistic worst-case assumptions and the added value of the SecANet output—in particular, its graph-based policy visualization, its execution semantics, and its applicability to Petri net-related analysis—the increased output memory requirements can be regarded as moderate.

3.2.8.3 More Efficient Analysis

Although the SecANet has important properties for fostering an efficient computation—for example, the overall net still represents a safe P/T net—the exponential growth of the runtime in obstructability analysis indicated by the example in Table 3.1 underlines the necessity to strive for more efficient techniques. Here, for a more efficient analysis of the obstructability of a SecANet, the repeatedly discussed endeavor of achieving the free-choice (FC) property could be pursued further. The rationale is that the satisfiability and obstructability of a live, safe, and reversible free-choice SecANet may be efficiently solvable as well. The previously explained transformations for arbitrary Petri nets into free-choice nets [124, 187, 188] can be used for that purpose. However, making a net “free choice” would be accompanied by a further increase in the structural complexity of the net. More precisely, assuming a SecANet that contains a free-choice WF-net, the free-choice transformation has to be done for all other net constructs, that is, the user-task assignments, constraints, and re-enactment-related constructions would have to be transformed accordingly.

Because of the structural properties of the SecANet modeling and the assumed block structure, the non-free-choice (NFC) elements of a net that have been considered here are indeed not so “arbitrary.” They already show such a structural limitation that they can be transformed into a free-choice net by only a few modifications. Figure 3.55 depicts a method by which the example DMV can be transformed into a free-choice net. For each user-task assignment, a single transition and a single place are added. Moreover, for each user-task assignment affected by the SoD constraint,

a place and transition are inserted as well, hence each choice must be made explicitly. For example, not only does the SoD place restrict the execution of user tasks, but it has to be explicitly decided whether the SoD place is to be put into action as well as for which user-task assignment it should take effect. The free-choice-related elements, therefore, allow a more detailed breaking down of the decisions taken during the process. Currently, the transitions are silent and do not add behavior to the net. However, one could also consider labeling them—for example, in order to explicitly encode the choices made in the execution sequences as well.

Hence although the increase in the structural complexity of the example DMV which is caused by the transformation into a free-choice net remains moderate, the transformation extends the state space and creates intermediate transitions. Both the new states and the new transitions demand interpretation. In this regard, the marking in Figure 3.56 relates to the obstruction marking of the DMV SecANet. Figures 3.57 and 3.58 display new variants of terminal markings that seem to obstruct as well.

On the one hand, the results of a free-choice SecANet can be interpreted with regard to the initial SecANet. Here, the markings in Figures 3.56 and 3.57 could be interpreted as “false-positive” obstruction markings, since they have no obvious relation to any marking of the underlying SecANet. In fact, in Figure 3.58 it looks as though the SoD constraint is fired “the wrong way,” and that it is even more restrictive. Such false-positive obstructions could be filtered out by backward firing the leftover tokens in the FC places (directly above the user-task transitions) of

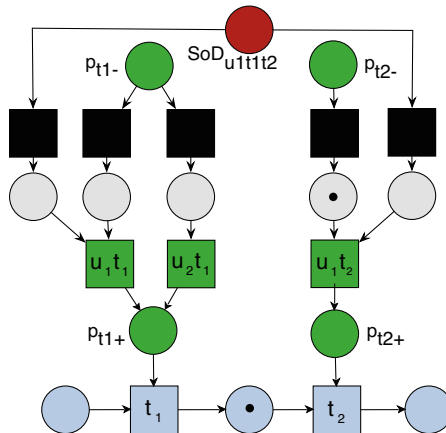


Figure 3.56 DMV free-choice SecANet with obstruction marking (true positive)

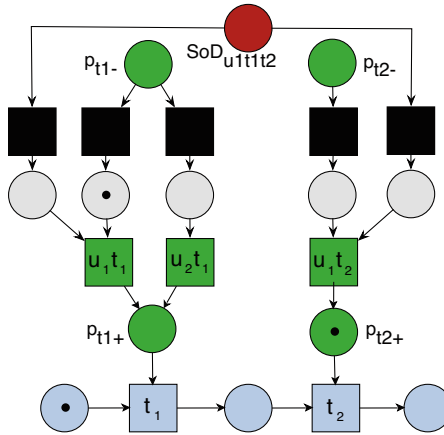


Figure 3.57 DMV free-choice SecANet with new variant of a potential obstruction marking (false positive)

the terminal marking. Thus the firing is first reversed such, so that the corresponding user-task assignment or constraint place is marked again. Then if some firing sequences allow escaping the deadlock, the marking is not a “real” obstruction. For example, the deadlock in Figure 3.58 could be resolved by backward firing the SoD-related token in the free-choice place and then firing the right-hand SoD transition of the free-choice-related nodes. Thus $t_{u_1 t_2}$ would be enabled in such a way that the deadlock would be resolvable. If SoD places do not affect a workflow execution, it would not matter if the left-hand or right-hand FC-SoD-transition is fired. Based on this backward-firing method, all example deadlock markings can be avoided, except for the marking in Figure 3.56, which is the one that represents the SecANet obstruction.

On the other hand, these new markings could also be re-interpreted with regard to the initial SecANet. For example, the free-choice-related Petri net elements could be interpreted as a sort of assignment reservation, anticipation, pre-assignment, or pre-decision, and as such could encode further states. Moreover, these elements could be related to costs. In an SoD constraint, for example, costs could be assigned to the FC elements to determine which of the user-task assignment should preferably be enabled.

Hence a free-choice SecANet would require some filtering of the results or at least some rules of interpretation, but it seems as though the effort needed for such a transformation would be manageable. Moreover, there seem to be only false-positive

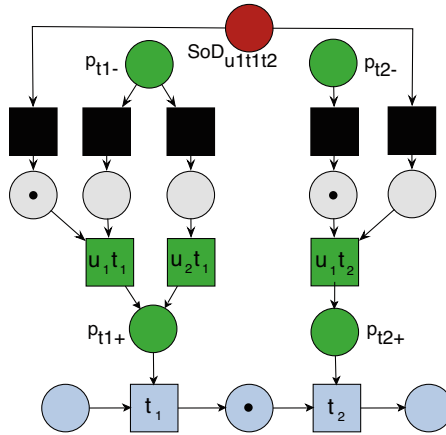


Figure 3.58 DMV free-choice SecANet with further variant of a potential obstruction marking (false positive)

obstructions and no false-negative ones. Thus in case of doubt, the result of an analysis of a free-choice SecANet may be too restrictive, but at least it would not overlook obstructions. Regardless of the chosen interpretation, by such a “free-choice preprocessing” the analysis of SecANet soundness may be done more efficiently. Therefore, it would be interesting to compare the potential increase in efficiency with the increase in net elements (i.e., the structural complexity) caused by the free-choice transformation and how the computation of the initial NFC SecANet compares to it. Here, the free-choice transformation of the re-enactment constructions for a cyclic SecANet or a SecA-WF-Net seems to be structurally much more complex.

Besides the effort to make a SecANet free choice, SecANet-specific analysis could be refined and certainly optimized. Based on the well-defined SecANet construction rules elaborated in this chapter, the SecANet offers the possibility for further structural analysis, property-preserving reductions, or simplification of the nets. For example, in order to reduce search space, only places or transitions which are of interest in regard to the occurrence of an obstruction could be considered. Then, for example, the question of whether every transition related to re-enactment is live could be ignored. Furthermore, the identification of typical obstruction patterns could facilitate the search for obstructions.

3.2.8.4 Common and Further Constraints

The applicability of the SecANet approach has been illustrated for common constraints, namely user-task assignments and SoD/BoD constraints. Thus far, in order to cause obstructions, SoD constraints have been of greatest interest. The neglect of BoD constraints is not unusual in WSP research. There, BoD constraints are eliminated during preprocessing, before solving a WSP instance [48]. On the one hand, the relation of BoD constraints to obstructability depends on how consistently a BoD is actually defined and whether it is considered in the authorization policy as well. Since a BoD constraint binds two tasks to the same user, if a certain user is not authorized for both tasks and executes one of them, the other task is not trivially executable. For example, Figure 3.26b displays an encoding of a BoD constraint, which may indeed cause such an obstruction, namely, if $t_{u_2t_1}$ is executed. Because of the missing assignment of user u_2 to t_2 , this part of the BoD constraint cannot be trivially fulfilled. Here, one could indeed consider eliminating the assignment encoded by $t_{u_2t_1}$. On the other hand, even if such contradictions between BoD constraints and the authorizations are avoided, the BoD constraints can still foster obstructions, since they basically restrict possible user-task assignments. This can in turn restrict the user-task assignments that satisfy the SoD constraints. As an example, Figure 3.59 shows a variation of the DMV SecANet that can obstruct because of the interplay between the SoD and BoD constraints. In that case, the workflow offers the additional possibility that the person who will later control the market value is first asked to compute an independent market value without bias. This could be done on a random basis—or, for example, if the calculated market value exceeds a certain threshold. Figure 3.60 shows the corresponding obstruction marking, which occurs after $t_{u_1t_1}$, t_1 , $t_{u_1t_3}$, and t_3 are fired. Here, on the one hand a BoD constraint obstructs the execution of $t_{u_3t_4}$, and on the other hand an SoD constraint obstructs $t_{u_1t_4}$.

In general, the constraints considered so far in the SecANet modeling represent entailment constraints that are sufficient to obstruct the process. The basic assumption for SecANet modeling is that the scope of the constraint entails two tasks. However, if one or both sides of a binary constraint encompass sets of tasks, this can be encoded with several choice places where each place models a separate conflict between two user-task assignments. Thus the entailment constraints that encompass sets of tasks can be broken down into several conflicting user-task pairs. The flattening does not restrict these entailment constraints in terms of the cardinalities of the involved sets of tasks. However, the focus is on the basic constraint on two singletons, which allows for straightforward application and explanation of the approach. However, beyond SoD or BoD constraints, there are further constraints (cf. Section 2.1.2.4), such as counting constraints, that could contribute to the occur-

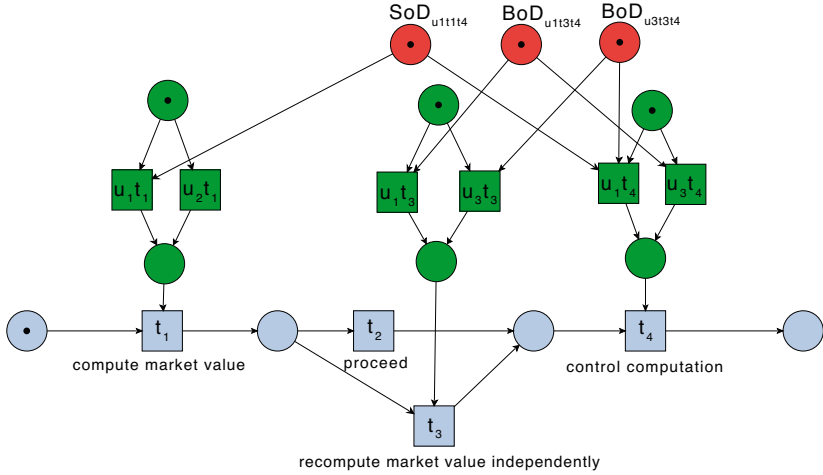


Figure 3.59 Variation of the DMV SecANet with SoD and BoD constraints causing an obstruction

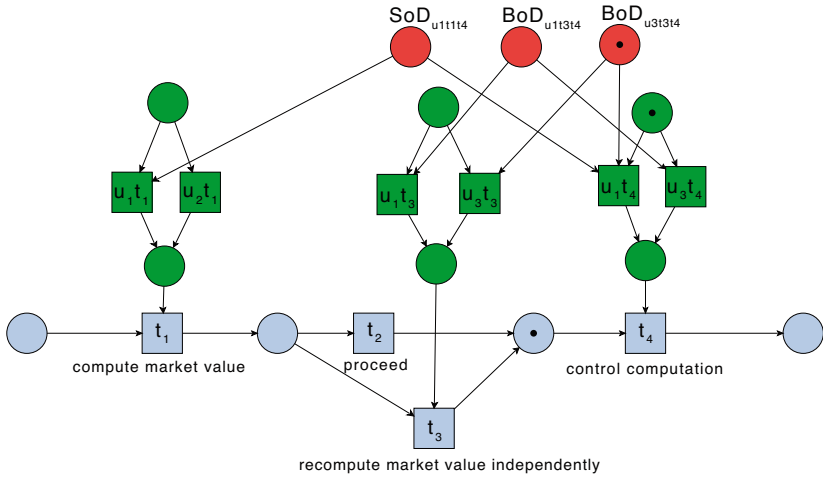


Figure 3.60 Obstruction marking in the variation of the DMV SecANet

rence of obstructions but are not yet covered by the approach. Moreover, role-based access control models could be integrated as well.

In general, the policy encoded in the SecANet can also be seen as a conflict between the allocation of resources, where the resource is the token that allows assignment. The notion of resource allocation becomes even more apparent in the modeling of the availability of resources or users. In this regard, one could think of how a task-specific subnet could be modeled that encodes resilience and then incorporates it into the given net. In order to address such extensibility, in the final section of this chapter an extension of the SecANet approach will be presented, namely SecANet+. That can be seen as the epilogue of this chapter and suggests a general extensible framework for modeling of security-aware Petri nets that incorporates the policy elements considered so far as well as further constraints or restrictions that may impede the process execution.

3.3 SecANet+: Extension of the SecANet Approach

SecANet+ generalizes the SecANet approach with a deliberate focus on extensibility and modularity. Still, the preceding description of the SecANet approach is necessary to understand the idea, meaning, and purpose of the SecANet+ approach in context. The SecANet approach integrates all net elements directly into the workflow net. Then the decomposition of the SecANet allows for the determination of the language of its subnets and the composition of its overall language (cf. Section 3.2.4.5). SecANet+, in contrast, aims to build the individual (sub)nets first and then combine these nets. Therefore, the properties and languages can be directly determined for each net separately and for the combined nets, so that there is no need for an elaborate decomposition and recomposition. Again, the use of Petri nets will be rewarding, since the previously introduced net synchronization operator (subsequently indicated by $sy\sigma$) will play a central role. Although the composition of languages will need to consider the prefix language, the nets and languages resulting from the SecANet and SecANet+ are the same for the same input and modeling method. Because of its modular setup, the SecANet+ approach builds the basis for further extensions—for example, for further constraints or to model further behavior that in some way constrains or restricts the workflow execution.

The overall framework of the SecANet+ approach is depicted in Figure 3.61. As with SecANet, the intuition behind SecANet+ is to capture every input in a process-oriented way—in particular, by including further inputs that in some way restrict the execution of tasks, such as counting not only the constraints but also user absences. The SecANet+ approach is twofold:

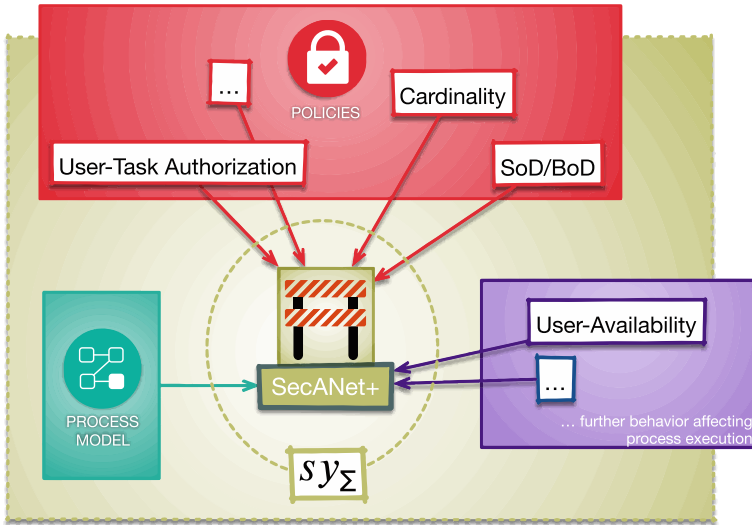


Figure 3.61 The SecANet+ approach

1. **Creating nets for each input:** The arrows in Figure 3.61 indicate these separate modeling steps for each of the considered inputs. Here, it is key to consider how the different inputs that contribute to the overall process in some way relate to each other. For example, a user-task authorization and the workflow may overlap with their tasks, or an SoD constraint and a user-task authorization may overlap in their user-task assignments. Those overlapping areas contain the “common denominator” of the transitions that build the interfaces that will be used to synchronize the individual nets. This explicit transition-oriented modeling also provides the basis for further extensions. Here, the examples in Figure 3.61 indicate only the broad range of inputs that may restrict the process execution in some way. User-activity assignments can, for instance, also be related to the availability of users, which could be used to glue together models that encode resilience. In order to create the nets of further inputs, searching for the intersections of transitions must be done in more than just a formal (theoretical) sense. Some thought must be put into the modeling and behavior of the inputs. Their execution semantics and states need to consider the other inputs as well as the overall process.
2. **Combining the nets.** The “common denominator” builds the interface that combines the transitions of each modeled net. Therefore, in this step the resulting

nets are successively combined into one representation by using the synchronization net operator (indicated by $s\gamma_{\Sigma}$ in Figure 3.61). As previously explained, the synchronization operator can synchronize two nets with each other by using the activity or transition labelings that are in both nets (i.e., their “common denominator”).

By this modular, twofold approach, the SecANet+ provides higher modularity and flexibility in choosing the constraints that affect process execution in some way as well as in combining them into a single representation. Elements of the encoded policy can be stored and kept available. For example, the policy nets and workflow nets could be created and maintained independently of each other in a kind of construction kit which could be combined (or synchronized) only when required. This could save repeated modeling effort, since, for example, the same policy element can be applied to different workflow constellations or only slight adjustments in transition labeling would be required to re-use a modeled net, for example, because of changes in activity names. Moreover, changes that affect only parts of the policy may be done in a better way, without recomputing the entire encoding. Thus it could be possible to remove unused parts of an already existing policy.

In order to explain the SecANet+ approach, this section will first show how to create the building blocks that represent the different process-specific inputs and then later show how to compose them. Since much of this is similar to the SecANet approach, only the key points will be examined more closely, and they will be explained only by way of example, in order to show the applicability. For this reason, first the creation of the policy-related nets that consider the authorization policy and SoD constraints that are comparable with the SecANet approach are described. Then the combination of the resulting nets is described and exemplified with the DMV SecANet. As a final example of an extension of the approach, user availability will be modeled in a way that encodes resilience.

3.3.1 Create and Combine Policy Nets for Workflows

Figure 3.62 depicts the application of the SecANet+ approach to the considered security-aware process specification, namely a WF-net, a user-task authorization, and SoD constraints. What the processes, the authorization, and the constraints all have in common is their consideration of the activities in some way. More precisely, the “common denominator” of the inputs considered so far consists of the process tasks or the user-activity assignments, as depicted in Figure 3.62. Here, modeling of the individual inputs is indicated by separate rectangles. Then these individual nets

are combined by synchronizing the net on their common transitions (i.e., $\Sigma_{input_1} \cap \Sigma_{input_2}$). Note that, in contrast to the decomposition of the SecANet, the terminal markings resulting from the overall composition of the different nets cannot be determined prior to the synchronization. If, for instance, the only final marking which was assumed for a net that models user-task authorization is \emptyset , this would then exclude other terminal markings that would result from synchronization. Therefore, the final marking is assumed to contain all reachable markings. Accordingly, the final marking determines the prefix language of the net (cf. Definition 3.41).

3.3.1.1 Creating Policy Nets

The generalized encoding of acyclic nets is described here, and it will be illustrated by the example DMV process in Figure 3.21. As mentioned in the first step of the approach, it must first be known which of the nets are to be combined later. Hence the first step of the approach is to model the policy with Petri nets. Thus the policy modeling presented in Section 3.2.2 can now be adapted to the creation of individual processes, independent of the process model, so each Petri net can be regarded separately (cf. Figure 3.62). In order to avoid confusion with previous definitions

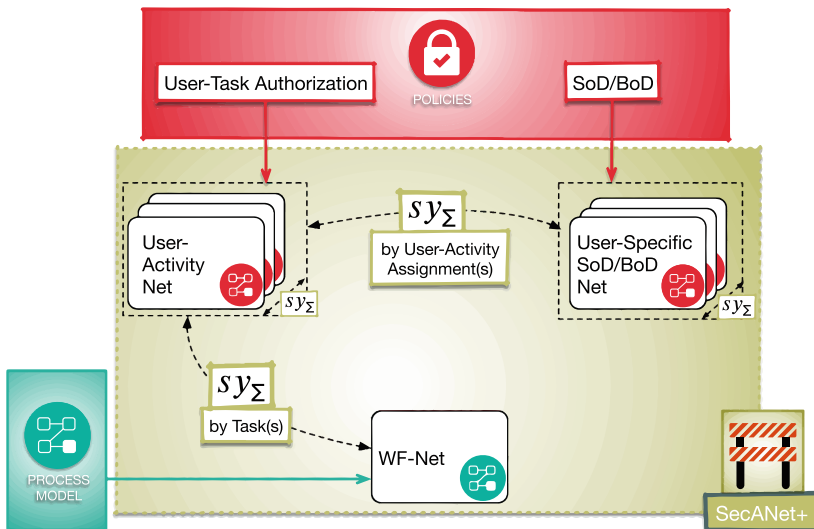


Figure 3.62 SecANet+ for security-aware process specification, with user-task authorization and SoD

of the SecANet encoding, “tasks” will now be termed “activities.”. Therefore, in the SecANet+ approach, the user-task authorization TA will be replaced with user-activity authorization UA , where the set of activities A is used analogously to the set of tasks T . In order to have common transitions in the two nets, a user-activity Petri net models the activity for which it describes the given authorizations.

User-Activity Petri Net: Since the actual user-activity assignment and the subsequent access to the activity shall be modeled, the activity needs to be involved as well. Equivalently to the SecANet, “-” stands for the “unassigned” state, and “+” stands for the “assigned” state.

Definition 3.75 (User-Activity Petri Net). *Let $a \in A$ be an activity such that there exists a user $u \in U$ where (u, a) is an element of the user-activity relation $UA \subseteq U \times A$, that is, user u is mapped to activity a . Hence the set of all users authorized for activity a can be denoted by*

$$U_a = \{u | u \in U, (u, a) \in UA\} = \{u_1^a, u_2^a, \dots, u_n^a | n = |U_a|\}.$$

Based on this, the language-generating Petri net for activity a , $N_a^{UA} = \langle P, T, \mathcal{F}, m_0, [m_0], T, id_T \rangle$, is defined with

$$P_{a-+} = \{p_a^-, p_a^+\},$$

$$T = \{t_a\} \cup \{t_{ua} | u \in U_a\},$$

$$\mathcal{F} = \{ \langle p_a^-, t \rangle, \langle t, p_a^+ \rangle | t \in T - \{t_a\} \} \cup \{ \langle p_a^+, t_a \rangle \}, \text{ and}$$

$m_o = \langle 1, 0 \rangle$. The figure below shows the graphical representation of such a net N_a^{UA} :

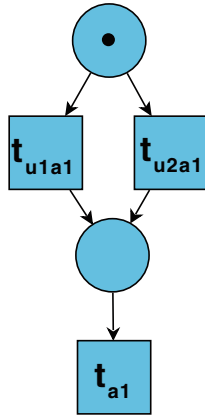
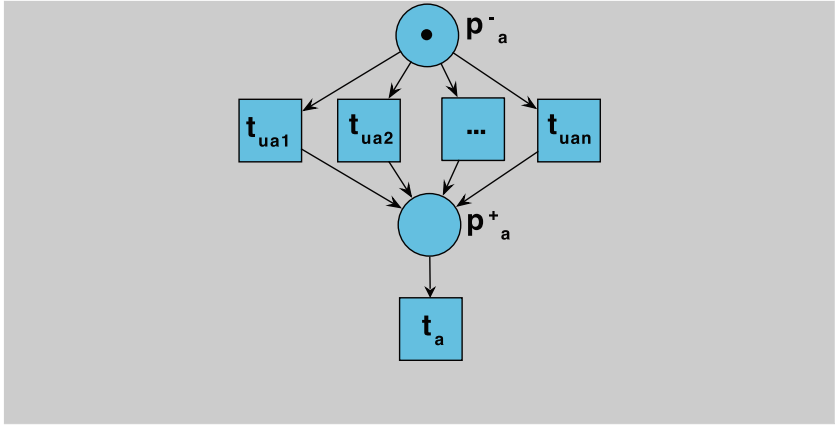


Figure 3.63 Example of user-activity net $N_{a_1}^{UA}$

Figure 3.63 shows an example of the encoding of the user-activity assignment for the activity “compute market value” of the DMV process. Given that user u_1 and user u_2 are authorized for activity a_1 and user u_1 is authorized for a_2 , this is formalized with the activity set $A = \{a_1, a_2\}$, the user set $U = \{u_1, u_2\}$, and the user-activity assignment $UA = \{(u_1, a_1), (u_2, a_1), (u_1, a_2)\}$. Then for a_1 the user-activity Petri net $N_{a_1}^{UA} = \langle P, T, \mathcal{F}, m_0 \rangle$ consists of the components

$$\begin{aligned}
P &= \{P_{a_1}^-, P_{a_1}^+\}, \\
T &= \{t_{a_1}, t_{u_1a_1}, t_{u_2a_1}\}, \\
\mathcal{F} &= \{\langle P_a^+, t_{a_1} \rangle, \langle P_a^-, t_{u_1a_1} \rangle, \langle P_a^-, t_{u_2a_1} \rangle, \langle t_{u_1a_1}, P_a^+ \rangle, \langle t_{u_2a_1}, P_a^+ \rangle\}, \\
m_o &= \langle 1, 0 \rangle.
\end{aligned}$$

The graphical representation of the Petri net $N_{a_1}^{UA}$ is depicted in Figure 3.63. The properties of such a net, as well as its language, are directly observable. User-activity Petri nets are safe (i.e., 1-bounded) and free choice. Compared to the language of a user-task subnet in Definition 3.51, the language of a user-activity net is easier to determine, namely $L(N_a^{UA}) = \text{Pre} \{t_{u_1a}, t_{u_2a}, \dots, t_{u_n a}\} \cdot \{t_a\}$. For the example, the set of possible words of the language of the net is $L(N_{a_1}^{UA}) = \text{Pre} \{t_{u_1a_1} t_{a_1}, t_{u_2a_1} t_{a_1}\}$.

Modeling of SoD Constraints: Analogously to the user-activity assignments, the SoD constraints can be defined as an elementary choice construct, given the type of relation (\neq) and the involved activities. In order to allow the combination with the given authorizations, as indicated in Figure 3.62, what needs to be considered are the user-activity authorizations which will later build the common denominator that will combine them with the user-activity nets.

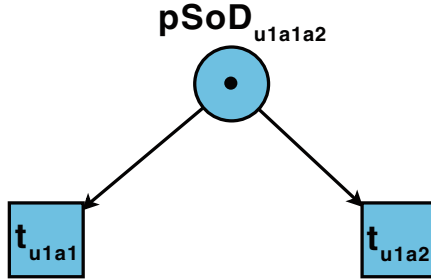


Figure 3.64 Example of a user-specific SoD Petri net $N_u^{(a_1, a_2, \neq)}$

Definition 3.76 (User-Specific SoD Petri Net). Let $u \in U$ be a user, and let $a, b \in A$ be two activities such that there exists an SoD constraint $c_{SoD} \in C$ of the form (a, b, \neq) , that is, activities a and b may not be executed by the same user, and the user-activity relations $(u, a), (u, b) \in UA$. The language-generating user-specific SoD Petri net is defined as $N_u^{(a, b, \neq)} = \langle P, T, \mathcal{F}, m_0, [m_0], T, id_T \rangle$ with

$P = \{p_{uab}^{sod}\},$
 $T = \{t_{ua}, t_{ub}\},$
 $\mathcal{F} = \{\langle p_{uab}^{sod}, t_{ua} \rangle, \langle p_{uab}^{sod}, t_{ub} \rangle\},$ and
 $m_o = \langle 1 \rangle.$ The figure below shows the general graphical representation of such a net $N_u^{(a,b,\neq)}$:

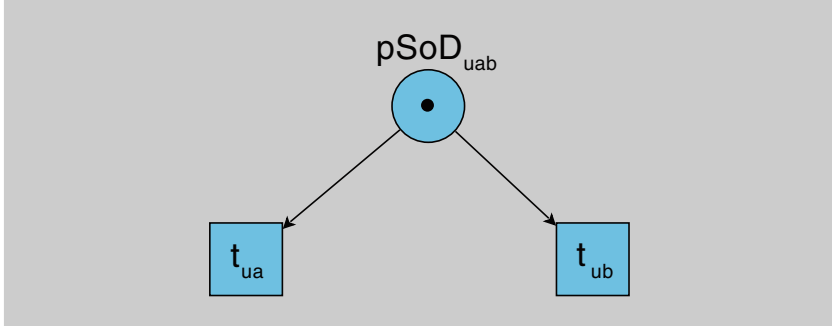


Figure 3.64 is the graphical representation of the SoD net $N_{u_1}^{(a_1, a_2, \neq)}$ of the DMV example and consists of the components

$P = \{p_{u_1 a_1 a_2}^{sod}\},$
 $T = \{t_{u_1 a_1}, t_{u_1 a_2}\},$
 $\mathcal{F} = \{\langle p_{u_1 a_1 a_2}^{sod}, t_{u_1 a_1} \rangle, \langle p_{u_1 a_1 a_2}^{sod}, t_{u_1 a_2} \rangle\},$ and
 $m_o = \langle 1 \rangle.$

It can directly be seen that SoD Petri nets are safe and free choice. The language of such a user-specific SoD net can be denoted by $L(N_u^{(a,b,\neq)}) = \text{Pre}\{t_{ua}, t_{ub}\}.$ Hence the language of the example net is $L(N_{u_1}^{(a_1, a_2, \neq)}) = \text{Pre}\{t_{u_1 a_1}, t_{u_1 a_2}\}.$

3.3.1.2 Combining Policy-Related Nets and the Workflow

After these components are created, they are combined by using the net synchronization operator (cf. Definition 3.45). For the sake of clarity, it is recommended that the respective net types be synchronized first with themselves and then with the other net types, analogously to the order of composition in Sections 3.2.4.5 and 3.2.4.4. Moreover, in order to consider the full behavior of the nets, their prefix languages must be determined. Therefore, all reachable markings represent the final markings. Since the final markings are again transformable into normal form and all nets are free, the language of the combined nets can again be determined with the help of the restriction operator—or, if the synchronization alphabet is empty, by the shuffle operator. For the user-activity assignments and the SoD constraints, the combination

of the individual nets ultimately results in the same net as in the SecANet approach. An example of this will now be used to demonstrate the applicability of the approach.

Similarly to the composition in Section 3.2.4.4, the synchronization alphabet for the synchronization of the user-activity nets is empty, since each user-activity net is created separately. Figure 3.65 gives the net of the combined example activity nets, namely $N_A^{UA} = N_{a_1}^{UA} \text{ sy } \emptyset N_{a_2}^{UA}$. Moreover, analogously to Section 3.2.4.4, its language can be directly deduced, specifically $L(N_A^{UA}) = \text{Pre } L_{N_{a_1}^{UA}} \sqcup L_{N_{a_2}^{UA}} = \text{Pre } \{t_{u_1a_1} \ t_{a_1}, t_{u_2a_1} \ t_{a_1}\} \sqcup \text{Pre } \{t_{u_1a_2} \ t_{a_2}\}$. This modular combination allows for direct observation of the properties of the combined net constructs. For example, it can be seen that when the user-activity nets are combined, the resulting nets are safe and free choice. This contrasts with synchronization of user-specific SoD nets, as in Section 3.2.4.4, where the restriction operator \textcircled{R} needs to be applied to determine their language. Since in the example there is only one user-specific SoD net, the language of all user-specific SoD nets is $L(N^{SoD}) = \text{Pre } \{t_{u_1a_1}, t_{u_2a_1}\}$. Figure 3.66 shows the graphical net that results from the synchronization of all net elements for the example net. The synchronized activities are highlighted by “sy.”

Language Composition: The separate modeling allows for direct examination of the language of the generated nets. Since it is not known beforehand which terminal markings will ultimately result from the synchronization of the nets, the prefix language has to be assumed for the individual nets here. Otherwise, words that might result from later combinations would be excluded from the beginning. Thus the prefix language of each individual net first includes all possible words. In the

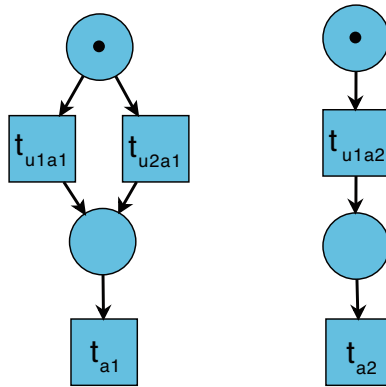


Figure 3.65 Combination of the two example user-activity Petri nets N_A^{UA}

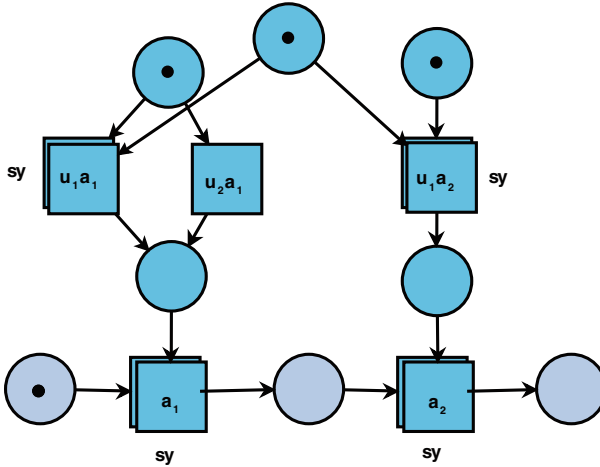


Figure 3.66 Example DMV SecANet $N_{wf}^{UA sy SoD}$ after combining all nets

course of the synchronization, every prefix or word that becomes impossible will be “cut out” step by step. The terminal language of the composed net can then be derived by considering the full prefixes only. Here, the last letter of a word indicates that a terminal marking has been reached. Such a full word sequence of a prefix language ends when there are no further letters that could be appended, that is, when no further markings are reachable. If the words that encode the terminal firing sequences of the combined net contain all the letters of the full prefix words of the prefix language of the WF-net, they represent satisfiable words; otherwise, they represent obstructed words.

To illustrate this, the language of the DMV SecANet+ will be determined. The prefix language of the workflow net N_{wf} that encodes the DMV control flow is $L(N_{wf}) = \text{Pre}(\{t_{a_1} t_{a_2}\})$. The synchronization of the prefix language is advantageous, since the intersection operator may easily consider prefixes and not only full words. For example, because of the prefix notation, the word t_{a_1} is already included as a prefix in $L_{wf} = \text{Pre}(\{t_{a_1} t_{a_2}\})$. Thus the prefix notation can be used to determine the (terminal) language of the DMV SecANet more elegantly. First, the languages of the two policy-related nets can be combined.

$$\begin{aligned}
L_A^{UA} \textcircled{R} L_{SoD} &= ((L_A^{UA} \sqcup (\Sigma_{SoD} - \Sigma_A^{UA})^*) \cap (L_{SoD} \sqcup (\Sigma_A^{UA} - \Sigma_{SoD})^*)) \\
&= ((\text{Pre} (\{t_{u_2a_1} \cup t_{u_1a_1}\} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}) \sqcup (\Sigma_{SoD} - \Sigma_A^{UA})^*) \\
&\quad \cap (\text{Pre} (t_{u_1a_1} \cup t_{u_1a_2}) \sqcup (\Sigma_A^{UA} - \Sigma_{SoD})^*)) \\
&= ((\text{Pre} (\{t_{u_2a_1} \cup t_{u_1a_1}\} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}) \sqcup (\emptyset)^*) \\
&\quad \cap (\text{Pre} (t_{u_1a_1} \cup t_{u_1a_2}) \sqcup (\{t_{u_2a_1}, t_{a_1}, t_{a_2}\})^*)) \\
&= (\text{Pre} (\{t_{u_2a_1}, t_{u_1a_1}\} t_{a_1} \sqcup t_{u_1a_2} t_{a_2})) \\
&\quad \cap (\text{Pre} (\{t_{u_1a_1}, t_{u_1a_2}\}) \sqcup (\{t_{u_2a_1}, t_{a_1}, t_{a_2}\})^*)) \\
&= ((\text{Pre} (\{t_{u_2a_1} t_{a_1}, t_{u_1a_1} t_{a_1}\} \sqcup t_{u_1a_2} t_{a_2})) \\
&\quad \cap (\text{Pre} (\{t_{u_1a_1}, t_{u_1a_2}\}) \sqcup (\{t_{u_2a_1}, t_{a_1}, t_{a_2}\})^*)) \\
&= (\text{Pre} (\{t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}, t_{u_1a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}\})) \\
&\quad \cap (\text{Pre} (\{t_{u_1a_1}, t_{u_1a_2}\}) \sqcup (\{t_{u_2a_1}, t_{a_1}, t_{a_2}\})^*)) \\
&= \text{Pre} (\{t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}, t_{u_1a_1} t_{a_1}\}).
\end{aligned}$$

Analogously, the synchronization of the prefix languages of the policy net and the WF-net is as follows:

$$\begin{aligned}
L(N_{wf}) \textcircled{R} L(N^{UA \text{ sy } SoD}) &= L(N_{wf}) \text{sy}_{\Sigma_{wf} \cap \Sigma^{UA \text{ sy } SoD}} L(N^{UA \text{ sy } SoD}) \\
&= L(N_{wf}) \text{sy}_{\{t_{a_1}, t_{a_2}\}} L(N^{UA \text{ sy } SoD}) \\
&= ((L_{wf} \sqcup (\Sigma^{UA \text{ sy } SoD} - \Sigma_{wf})^*) \\
&\quad \cap (L^{UA \text{ sy } SoD} \sqcup (\Sigma_{wf} - \Sigma^{UA \text{ sy } SoD})^*)) \\
&= (\text{Pre} (\{t_{a_1} t_{a_2}\} \sqcup (\{t_{u_1a_1}, t_{u_2a_1}, t_{u_1a_2}\})^*) \\
&\quad \cap (\text{Pre} (\{t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}, t_{u_1a_1} t_{a_1}\} \sqcup (\emptyset)^*)) \\
&= ((\text{Pre} (\{t_{a_1} t_{a_2}\}) \sqcup (\{t_{u_1a_1}, t_{u_2a_1}, t_{u_1a_2}\})^*) \\
&\quad \cap \text{Pre} (\{t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}, t_{u_1a_1} t_{a_1}\}) \\
&= \text{Pre} (\{t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2} t_{a_2}, t_{u_1a_1} t_{a_1}\}).
\end{aligned}$$

Note that, similar to the example decomposition and composition DMV SecANet, the full prefix words $(t_{u_2a_1} t_{a_1} \sqcup t_{u_1a_2}) t_{a_2}$ and $t_{u_1a_1} t_{a_1}$ encode the satisfiable and obstructed words, respectively.

3.3.2 Resilience Extension: Modeling User Absence

The elements of the SecANet+ approach could now also be further refined, so that, for example, it is ensured that there are no leftover tokens in the places of the newly combined nets after execution. Also, cycles could be encoded in isolated nets and combined in such a way that canceling and enacting the policy is possible. However, in order to illustrate the extensibility of the SecANet+ approach beyond the net constructions given so far, we sketch a net that could be used to encode user absence. As indicated in Chapter 2, user availability can be used to determine the resilience, and it plays an important role in the WSP context as well.

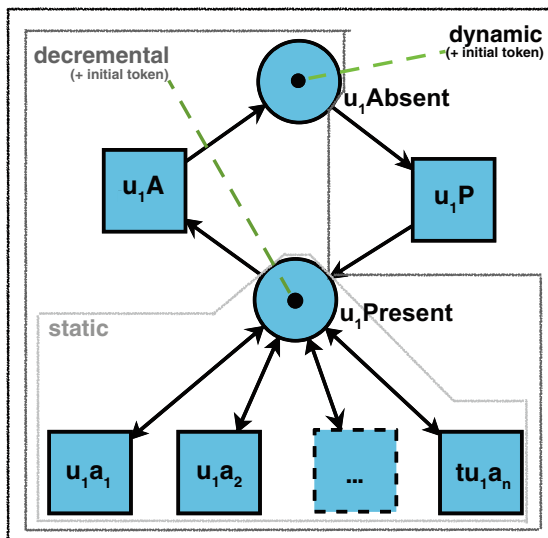


Figure 3.67 Compact representation of user-availability net (A = Absence, P = presence) (with self-loop)

Figure 3.67 gives the graphical representation of an idea for a user-availability net that is supposed to take all three resilience levels introduced by Wang and Li [207] into account and the related initial markings.

- First, for static resilience, the idea is to create a place that indicates user presence (e.g., $p_{u1Present}$) that is bidirectionally connected to all user-activity transitions

of the same user. If this place does not have a token, which is then a precondition for the user-activity transition to be executable, it means that this user is not available. The self-loop could easily be avoided by adding a further place and transition (analogously to the “cancellation control” in Definition 3.69). The self-loop is neglected, because here only the basic idea of modeling is to be conveyed. Hence static resilience is represented in the state of the corresponding place, that is, the presence or absence of a token in the corresponding place directly encodes whether a user is present or absent.

- Decremental resilience means that a given user who was initially authorized and available to execute an activity becomes unavailable for that activity. Here, a transition that indicates a change in user availability needs to be introduced that disallows all related transitions of that user to be assigned to a given activity. For this, a further net transition (e.g., t_{u1A}) that consumes the token of the place that indicates user presence is introduced and produces a token in a new place that indicates user absence (e.g., $p_{u1Absent}$). If this transition is fired, the user-activity transition is no longer fireable. Thus the firing of the “absence transition” encodes the fact that the user becomes absent for the rest of the execution.
- Finally, dynamic resilience allows making the users come back. For this, a newly introduced “presence transition” (e.g., t_{u1P}) is produced at the place that indicates user-absence conditions. This transition can produce the token in the place that indicates a return to availability (e.g., $u1Present$).

Thus the net in Figure 3.67 would explicitly allow encoding of a user being available, being unavailable), or having come back. The transitions encoding user presence or absence would be represented in the language of the net as well. For instance, in order to encode a typical notion concerning resilience, namely k -resilience, where k is the number of absent users, the “absence transition” would have to be triggered for two users in the corresponding user-availability nets for u_1 and u_2 , for example, t_{u1A} and t_{u2A} . Based on the user-activity transitions, user-availability nets, as in Figure 3.67, could easily be integrated into a SecANet. They could then be combined with each user-activity assignment and the overall process by using the net synchronization operator.

In conclusion, SecANet+ allows not only taking constraints resulting from regulatory requirements or corporate governance into account but also considering restrictions resulting from the context or environment of a business, for example, user unavailability. When it comes to resilience, user availability also demands authorization, since not having access control would mean that any person would be allowed to execute the workflow tasks. Then the presence of a single user would trivially always suffice to execute the workflow.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





OLive-M: A SecANet Use Case for Model-Based Obstruction Solving

4

If it is not possible to avoid an obstruction, and given that early process termination is not an option and that the business goal encoded by the process is still to be achieved, a method for completing an obstructed execution must be provided. This chapter introduces an approach called OLive-M to show that, on the basis of a SecANet, it is not only possible to capture obstructions but also to show how it can be applied to provide ways to resolve them. Thereby, to restore the liveness of an obstructed workflow execution caused by access control enforcement, a security-sensitive partial plan that completes the execution must be found. This approach will act within the general framework for requirements set up in Chapter 2; namely, on the basis of the captured state of an obstruction (GR-1), the obstruction is supposed to be resolved in a security-sensitive way based on indicators (GR-2), which are captured as costs. In the course of the approach, it will be depicted how all inputs that occur during the process lifecycle (i.e., model, policy, and log) are integrable (GR-3). This chapter presents a use case that demonstrates how, on the basis of a SecANet encoding, Petri net-related analysis techniques can be applied or “tweaked” to resolve obstructions.

Figure 4.1 shows the OLive-M approach. Here, an obstructed execution is supposed to live again based on a SecANet model. The requirements for such model- or specification-based solutions have been subsumed in “Requirements for specification-based completability (RSC)” in Chapter 2. To allow for obstruction-resolving enforcement (RSC-1), partial plans that resolve obstructions must be generated. Therefore, the idea is to first “revive” the obstructed marking to escape the deadlock situation and obtain those plans. More specifically, to make the obstructed marking live again, tokens are supposed to be added to the places to enable transitions. Based on a marking that enables transitions; i.e., a live marking, a sequence of transitions that can complete the obstructed execution is to be found. Thereby, security-sensitive overrides (RSC-2) may take violations of the policy into account.

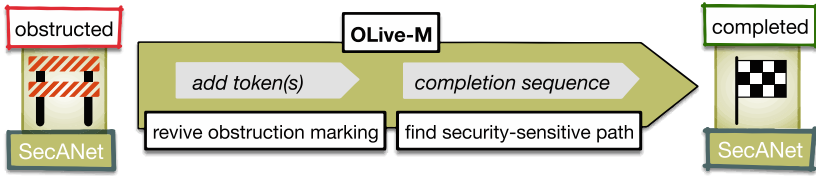


Figure 4.1 Overall OLive-M approach to resolve obstructions

The costs assigned to the considered places and transitions of the SecANet can be used to assess security-sensitivity. A minimal cost is assumed to provide the most security-sensitive solution. In a nutshell, the OLive-M approach involves two aspects: On one hand, adding tokens and computing the completion sequence from there, and on the other hand, determining the token addition and sequence with minimal cost. Such a completion sequence then represents a plan suitable for delegating the remaining tasks to users automatically (RSC-3). The obstruction is encoded using a SecANet, which can fully capture all specification-based information of relevance for the obstruction; i.e., the workflow, authorizations, constraints, users, and costs. Therefore, the solution can also be based on all relevant information to allow for an obstruction-aware completion (RSC-4).

A straightforward way to resolve an obstructed marking according to the described OLive-M approach would be to try out where an added token or tokens would lead to a live marking. Then a reachability graph could determine whether there is a reachable marking that contains the desired final marking. Thereby, the resulting costs of the added tokens and the transitions involved in the sequence of completion could be determined. However, for more extensive nets, because this method may face the problem of a state-space explosion, the “small” example SecANet can be resolved manually. Figure 4.2 shows a DMV SecANet with annotated costs and how the OLive-M approach can be applied. The highlighted costs are required to revive the obstructed marking and to complete the workflow. For example, on the basis of the obstruction marking $\{p_2, p_{t_2-}\}$, adding a token to the SoD place as depicted in the net would allow escaping the obstruction marking because the resulting marking would then enable the transition $t_{u_{1t_2}}$. However, adding a token to the SoD place would violate the SoD constraint, which is associated with a cost of $c = 5.0$. After firing $t_{u_{1t_2}}$ ($c = 1.0$) and t_2 ($c = 1.0$), this would result in a total cost of 7.0. This is obviously the only option to resolve the obstructed marking and reach the final marking $\{p_o\}$ that makes sense here. For larger nets with a comprehensive workflow structure, multiple SoD places, more transitions, and differing costs, it can be assumed that more than one solution may exist. Here (one

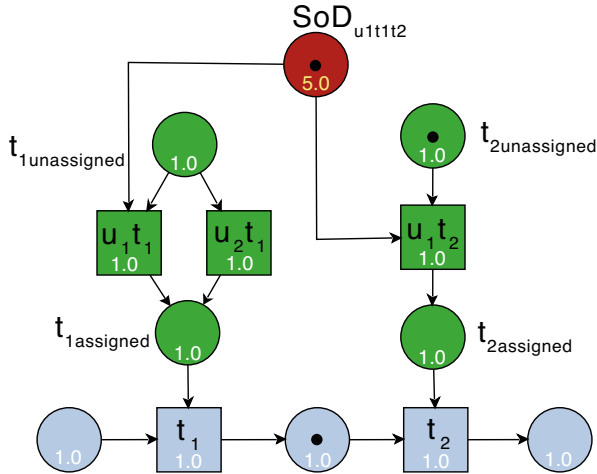


Figure 4.2 DMV SecANet with annotated costs and an added token in p_{SoD} to escape the obstructed state

of) the least costly; i.e., the most security-sensitive solution, is then to be chosen. The examples, moreover, stress that interpreting the states of a SecANet is crucial here again. For example, adding a token to a constraint place that has already been in effect represents a violation. Also, adding a token in p_{t_2+} instead would contradict the overall interpretation of a SecANet. More specifically, an added token in p_{t_2+} would enable t_2 . Firing t_2 would skip the user-task assignment $t_{u_1 t_2}$ and directly lead to the terminal marking $\{p_{t_2-}, p_o\}$. Hence, the place would indicate a user assignment even though no user has been assigned. The reductio ad absurdum of adding a token in p_o and claiming process completion would result in the marking $\{p_{t_2-}, p_2, p_o\}$. It can be seen that in both cases, the leftover tokens in the terminal markings apart from p_o indicate an improper process completion. Therefore, the final marking that must be reached to complete the process must be defined in a way that excludes such unwanted solutions. In the example, the final marking $\{p_o\}$ would take care of this. Based on these considerations, the following implications will be the keys to achieving the Olive-M approach.

- **Reaching proper final markings:** Solutions that resolve obstructions are supposed to be reasonable; i.e., the completion sequence must complete the obstructed execution from where the obstruction has happened. Hence, all unassigned WF tasks necessary to complete the workflow are supposed to be assigned

to a user. Adding tokens to places that would allow skipping pending user-task assignments or tasks is not allowed. The definition of the final marking, which encodes process completion, is supposed to exclude such simple undesired solutions. Based on the further observation that obstructions may occur due to tokens missing in either SoD or BoD places, the addition of tokens must take these constrained places into consideration. Hence, given an obstructed marking, the question is which tokens must be added to reach a *proper* final marking that encodes the completion of the process.

- **Security-sensitive optimization:** The decision of which tokens to add to escape an obstruction, together with the decision of which path to choose that allows the execution to complete, must be optimized. Such an optimal security-sensitive solution represents an optimization problem, where the objective function is supposed to be a minimization that considers both the violations required to escape from the obstruction and the length of the task sequences required to complete the workflow. To solve such constrained optimization problems, local search, heuristics, or genetic approaches, and in particular, integer linear programming (ILP) are some of the most common methods.

This chapter first introduces the structural theory of Petri nets and then the context of ILP. That way, the setting of the approach can be described as a system of linear equations. Based on the theory that resolves such a system and finds a security-sensitive optimum, an ILP model that optimizes a marking equation and the costs associated with the SecANet will realize the OLive-M approach as a way to deal with obstructions. In contrast to computing a marking graph, this realization represents a “light technique” to resolve obstructions. The implementation of the approach will use experiments and discuss the results. Finally, in the course of the overall discussion of the approach, it will indicate how the OLive-M approach can be used to analyze satisfiability or resilience as well, which will show the versatility of the approach.

4.1 Methods and Realization of the OLive-M Approach

This section elaborates on the selection of Petri net analysis techniques to realize the OLive-M approach. These techniques should be capable of both determining how a certain final marking can be reached and of optimizing the solution. To illustrate their application, the selected techniques will be directly adapted and applied to the OLive-M context. The subsequent notation and terminology related to linear equalities or inequalities and (integer) linear programming (LP) are based on Schrijver [184].

4.1.1 Structural Theory of Petri nets

A Petri net system consists of both a net structure; i.e., the state variables (places) and their transformers (transitions), and a marking that represents a distributed overall state on the structure. The dynamics of the system or behavior are given by the evolution rules for the marking. On the basis of these two components, net-based models can be analyzed at two different levels: structural and behavioral. Structural reasoning allows the derivation of some “fast” conclusions about the possible behavior of the modeled system. Purely behavioral reasoning can be more conclusive but may require costly computations or may not even be feasible [186]. The three groups into which Petri net analysis is typically divided [155] are (1) the reachability (or also coverability) graph method, which explores the behavior, (2) the matrix-equation approach, and typically also (3) the reduction or decomposition techniques concern the net structure. The computation of the reachability graph involves essentially the enumeration of all reachable markings and applies to all classes of Petri net classes; however, it is limited to “small” nets due to the growing complexity of the state space. Particularly in light of this state-space explosion problem in behavioral analysis, structural methods are more appropriate for deducing whether a specific marking can be reached. Therefore, structural reasoning can be regarded as an abstraction of behavioral reasoning. For instance, instead of studying whether a given system, e.g., a net structure with an initial marking, has a finite state space, the system can be investigated to determine whether the state space is finite for every possible initial marking. Moreover, it can be investigated to determine whether there exists an initial marking that guarantees infinite activity rather than deciding this for a given marking [186]. Analogously, the main question of this chapter is not what markings could be reached, but whether a proper final marking is reachable from an obstructed state. Although structural reduction techniques and matrix equations are powerful, they often can be applied only to special subclasses of Petri nets or special situations [155]. Because the encoding of the SecANet is kept as simple as possible, and a SecANet represents an ordinary, plain, pure, and safe Petri net, it can be easily represented in a matrix as well.

Based on these considerations, the structural theory of Petri nets will subsequently be introduced to be used in the OLive-M approach. Here in particular, the addition of tokens, and the aim to search for firing sequences that lead to a proper final marking, point to the use of the so-called marking equation. This system of linear equations will then be adapted such that the OLive-M approach can be used.

First, basic matrix notations that allow the relation of the behavior and structure of a Petri net are given.

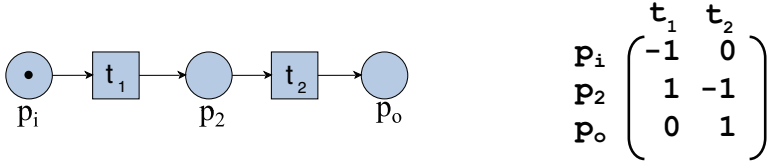


Figure 4.3 Incidence matrix of DMV workflow net

Definition 4.1 (Token conservation equations). Let $N = \langle P, T, \mathcal{F}, m_0 \rangle$ be a Petri net. Given a feasible sequence $m_0 \xrightarrow{\sigma} m$, the number of tokens for a place p in m is equal to the tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the output transitions of p in σ :

$$m(p) = m_0(p) + \sum_{t \in {}^*p} |\sigma|_t \mathcal{F}(t, p) - \sum_{t \in p^*} |\sigma|_t \mathcal{F}(p, t)$$

Definition 4.2 (Incidence matrix of a Petri net). The matrix $\mathbf{N}(p, t)$ which is defined by $\mathcal{F}(t, p) - \mathcal{F}(p, t)$ is called the incidence matrix of N .

Figure 4.3 gives an example of the incidence matrix of the DMV workflow net. Figure 4.4 shows the incidence matrix of the structurally more complex SecANet.

Definition 4.3 (Parikh vector). Let σ be a feasible sequence of transitions of N . $|\sigma|_a$ represents the number of occurrences of a in σ . The Parikh vector is a function $\widehat{\sigma}: T^* \rightarrow \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{t_1}, \dots, |\sigma|_{t_m})$. For simplicity, $|\sigma|_{t_i}$ will also be represented as $\widehat{\sigma}(t_i)$. The support of a Parikh vector $\widehat{\sigma}$, denoted by $\text{supp}(\widehat{\sigma})$ is the set $\{t_i \mid \widehat{\sigma}(t_i) > 0\}$.

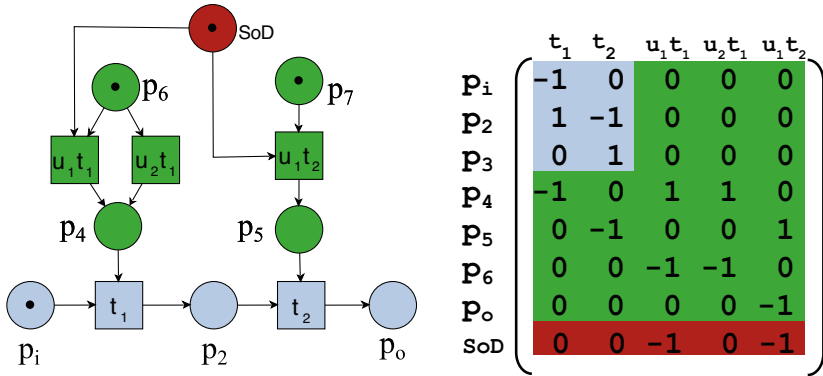


Figure 4.4 Incidence matrix of DMV SecANet

Definition 4.4 (Linear Equality). A linear equality is given by a row vector $a \in \mathbb{R}^n$, a vector of variables x , and a real value b . It is represented by

$$a^\top \cdot x = b$$

and it is feasible if there exists some assignment $k \in \mathbb{R}^n$ to x satisfying $a^\top \cdot k = b$.

Definition 4.5 (System of Linear Equalities). A system of linear equalities is a set of linear equalities. It is feasible if there exists a vector that satisfies all equalities of the set. If it is finite, it has a matrix-based representation

$$A \cdot x = b,$$

where the vectors a of the linear equalities are the rows of the matrix A , and the numbers b are the components of the vector b .

Based on these definitions, the marking equations for all the places in the net can be written in the following matrix form (see Fig. 4.5c for an example): $m = m_0 + N \cdot \hat{\sigma}$, where $N \in \mathbb{Z}^{P \times T}$ is the incidence matrix of the net: $N(p, t) = \mathcal{F}(t, p) - \mathcal{F}(p, t)$.

Definition 4.6 (Marking Equation). *If a marking m is reachable from m_0 , there exists a sequence σ such that $m_0 \xrightarrow{\sigma} m$ and the following system of equations has at least the solution $X = \widehat{\sigma}$:*

$$m = m_0 + N \cdot X \quad (4.1)$$

If equation (4.1) is not feasible, m is not reachable from m_0 . The inverse does not hold in general: there are markings satisfying equation (4.1) which are not reachable. Those markings are said to be *spurious* [186]. Figures 4.5a-c show an example of a net with spurious markings. The Parikh vector $\widehat{\sigma} = (2, 1, 0, 0, 1, 0)$ and the marking $m = (0, 0, 1, 1, 0)$ are a solution to the marking equation, as shown in Figure 4.5c. However, m is not reachable by any feasible sequence. Figure 4.5b depicts the graph containing the reachable markings and the spurious markings (shadowed). The numbers inside the states represent the tokens at each place (p_1, \dots, p_5). This graph is called the *potential reachability graph*. The initial marking is represented by the state $(1, 0, 0, 0, 0)$. The marking $(0, 0, 1, 1, 0)$ is reachable from the initial state only by visiting a negative marking through the sequence $t_1 t_2 t_5 t_1$, as shown in Fig. 4.5b. Therefore, equation (4.1) provides only a sufficient condition for the reachability of a marking and the replayability for a solution of equation (4.1).

For well-structured Petri nets, for example when nets are *free-choice* [155], *live*, *bounded* and *reversible*, equation (4.1) together with a collection of sets of places (*traps*) of the system completely characterizes reachability [76]. For the rest of cases, the problem of the spurious solutions can be palliated by the use of trap invariants [89], or by the addition of some special places named *cutting implicit places* [186] to the original Petri net that remove spurious solutions from the original marking equation. Concerning matrix equations, it is assumed that a Petri net is pure or is made pure by adding a “dummy pair” of a transition and a place, as discussed previously, to avoid self-loops [155].

To realize the OLive-M approach, the marking equation is applied in the following way. First, on the basis of the obstructed marking, tokens must be added to make the marking live again. In a second step, the Parikh vector then can be computed to give the Parikh representation of the completion sequence. The marking equation is adapted accordingly:

Definition 4.7 (OLive-M State Equation). Let $N_{SecANet}$ be an obstructed SecANet, with a corresponding obstruction marking m_{\otimes} . Then, if there exists at least one transition $t \in T_{SecANet}$ that can be enabled by the addition of a marking $m \in P_{SecANet}$ to m_{\otimes} ; i.e., the resulting marking $m_{live} \geq^* t$, the following system of equations has at least the solution $\Delta = m$:

$$m_{live} = m_{\otimes} + \Delta \tag{4.2}$$

If a marking m_{end} is reachable from m_{live} , there exists a sequence σ such that $m_{live} \xrightarrow{\sigma} m_{end}$ and the following system of equations has at least the solution $X = \hat{\sigma}$:

$$m_{end} = m_{live} + \mathbf{N} \cdot X \tag{4.3}$$

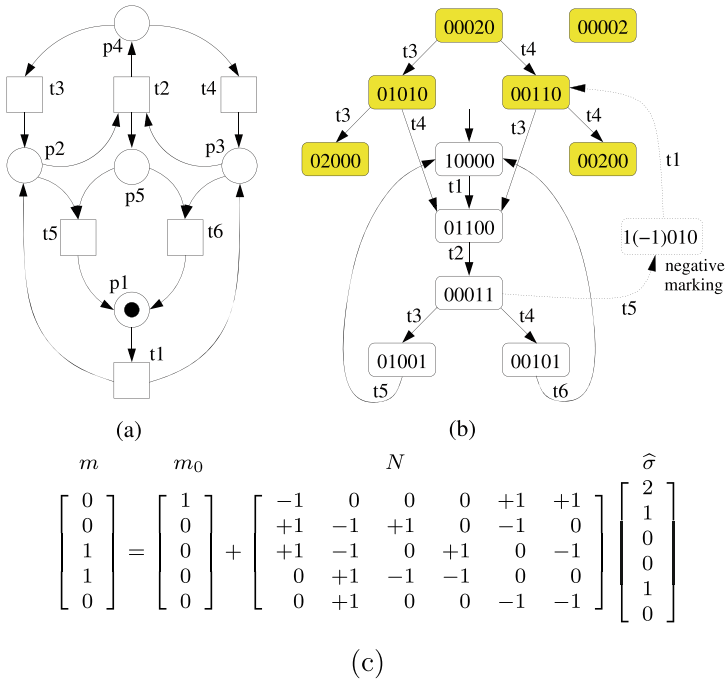


Figure 4.5 (a) Petri net. (b) Potential reachability graph. (c) Marking equation

$$\underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\mathbf{m}_{\text{end}}} = \underbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}}_{\mathbf{m}_{\otimes}} + \underbrace{\begin{pmatrix} p_i \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_o \\ p_{\text{SoD}} \end{pmatrix}}_{\Delta} + \underbrace{\begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 \end{pmatrix}}_{\mathbf{N}} \underbrace{\begin{pmatrix} t_1 \\ t_2 \\ t_{u_1 t_1} \\ t_{u_2 t_1} \\ t_{u_1 t_2} \end{pmatrix}}_{\mathbf{X}}$$

Figure 4.6 OLive-M state equations applied on an obstructed DMV SecANet. Each place-related variable of Δ (e.g., p_i) denotes the number of tokens to be added to the corresponding place. Each transition-related variable of X (e.g., t_1) denotes the number of occurrences of the corresponding transition in the resulting Parikh vector

Example Solution of the OLive-M State Equation: There are various methods to solve systems of linear equations. Probably the most well-known algorithm in linear algebra is the Gaussian elimination method, which is a polynomial-time method for solving a system of linear equations [114, 184]. Solving the system of linear equations in Figure 4.6 that applies the OLive-M marking equation to the example SecANet results in the solution

$$\begin{aligned}
 t_1 &= p_i \\
 t_2 &= p_i + p_2 + 1 \\
 t_{u_1 t_1} &= -p_i - p_2 + p_5 + p_{\text{SoD}} - 1 \\
 t_{u_2 t_1} &= 2p_i + p_2 - p_4 - p_5 - p_{\text{SoD}} + 1 \\
 t_{u_1 t_2} &= p_i + p_2 - p_5 + 1 \\
 p_3 &= -p_i - p_2 \\
 p_6 &= p_i - p_4 \\
 p_o &= p_i + p_2 - p_5 .
 \end{aligned}$$

Based on the observation that one can restrict the addition of tokens to constraints, one could boil the possible solutions down considerably by setting all place variables that do not encode a constraint to zero. In the example, this results in the solution

$$\begin{aligned}
 t_1 &= 0 \\
 t_2 &= 1 \\
 t_{u_1 t_1} &= p_{SoD} - 1 \\
 t_{u_2 t_1} &= -p_{SoD} + 1 \\
 t_{u_1 t_2} &= 1 \\
 p_o &= 0.
 \end{aligned}$$

Because the values of X and Δ are supposed to be natural numbers, here there is only one solution, namely $p_{SoD} = 1$; i.e., to escape the obstruction marking, p_{SoD} is additionally marked with a token. Based on this, the elements of the corresponding Parikh vector X can be determined. Figure 4.7 shows this variable assignment that solves the OLive-M state equation applied to the example DMV SecANet. Figure 4.8 shows the same solution graphically, which obviously corresponds to the solution found manually for Figure 4.2 above.

$$\begin{array}{l}
 \mathbf{P}_1 \\
 \mathbf{P}_2 \\
 \mathbf{P}_o \\
 \mathbf{P}_4 \\
 \mathbf{P}_5 \\
 \mathbf{P}_6 \\
 \mathbf{P}_7 \\
 \text{SoD}
 \end{array}
 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
 =
 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
 +
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
 +
 \begin{pmatrix} t_1 & t_2 & u_1 t_1 & u_2 t_1 & u_1 t_2 \\
 -1 & 0 & 0 & 0 & 0 \\
 1 & -1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 \\
 -1 & 0 & 1 & 1 & 0 \\
 0 & -1 & 0 & 0 & 1 \\
 0 & 0 & -1 & -1 & 0 \\
 0 & 0 & 0 & 0 & -1 \\
 0 & 0 & -1 & 0 & -1 \end{pmatrix}
 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$\underbrace{\hspace{10em}}_{\mathbf{m}_{end}}$

$\underbrace{\hspace{10em}}_{\mathbf{m}_{live}}$

$\underbrace{\hspace{10em}}_{\mathbf{N}}$

$\underbrace{\hspace{10em}}_{\mathbf{X}}$

Figure 4.7 OLive-M state equation $m_{live} = m_{\otimes} + \Delta$ (token(s) to add) and $m_{end} = m_{live} + \mathbf{N} \cdot \mathbf{X}$ (Parikh vector of completion sequence) to resolve the obstruction in DMV SecANet

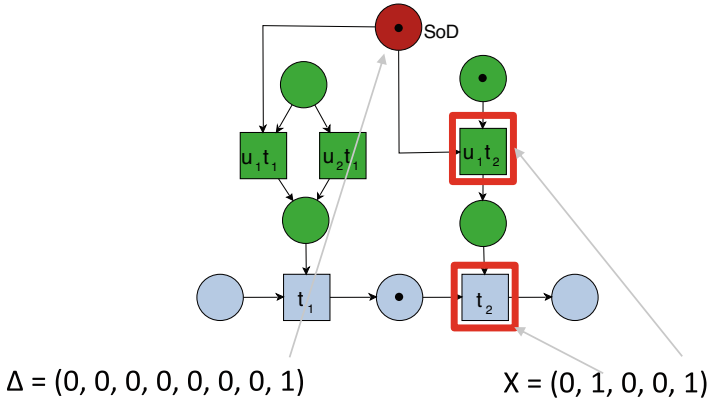


Figure 4.8 Δ (added tokens) and X (Parikh vector of completion sequence) to escape the obstructed state

Replayability of Solutions: Based on the fact that the solutions of the marking equation can be spurious, it may not be possible to reach the final marking with a proposed solution; i.e., there is no respective firing sequence. Therefore, the replayability of each solution must be checked. This means that because the solutions provided by the X vector do not provide the real ordering of transition executions, all possible linearizations must be explored to assess whether a solution obtained denotes a real sequence (in some of its possible linearizations). Due to the simplicity of the example, the order in which the transitions encoded in X must be executed is obvious, namely, $t_{u_1t_2}t_2$.

4.1.2 Linear Programming

The solution of the simple example in Section 4.1.1 suggests that the solution of the system of linear equations encoding the OLive-M state equation does not propose a specific solution for Δ and X . The number of variables in the vectors X and Δ exceeds the number of linear equations, which indicates that a solution itself probably contains variables again. Hence, rather, the solution results in further linear equations, which builds the constraints of an optimization problem that is supposed to then find specific variable assignments of natural numbers. Here, with regard to security-sensitive optimization, LP seems particularly appropriate. It not only allows the solution of systems of linear equalities but also considers linear inequalities, also

called constraints. Therefore, it is possible to take an objective function into account that can be used to find an optimal solution for the constraints given. That way, the decision on which tokens to add and which transitions to fire to achieve process completion will be embedded into a constrained optimization problem.

Definition 4.8 (Linear inequality). A linear inequality or constraint is given by a row vector $a \in \mathbb{R}^n$, a vector of variables x , and a real value b . It is represented by

$$a^\top \cdot x \leq b$$

and it is feasible if there exists some assignment $k \in \mathbb{R}^n$ to x satisfying $a^\top \cdot k \leq b$.

Definition 4.9 (System of Linear Inequalities). A system of linear inequalities is a set of linear inequalities. It is feasible if there exists a vector that satisfies all inequalities of the set. If it is finite, it has the matrix-based representation

$$\mathbf{A} \cdot x \leq b ,$$

where the vectors a of the linear inequalities are the rows of the matrix \mathbf{A} , and the numbers b are the components of the vector b .

Definition 4.10 (Linear Programming Problem). A LP problem is a system $\mathbf{A} \cdot x \leq b$ of linear inequalities, and optionally a linear function $c^\top \cdot x$ called the objective function. A solution of the problem is a vector of rational numbers that satisfies the constraints. A solution is optimal if it minimizes the value of the objective function (over the set of all solutions). An LP is feasible if it has a solution.

Definition 4.11 (Integer Linear Programming Problem). *An ILP is an LP where the set of solutions is given by vectors of integers only; i.e., it is feasible only if there exists some assignment $k \in \mathbb{Z}^n$ to x satisfying $A \cdot k \leq b$.*

The complexity of solving a linear problem depends on the domain under consideration. Specifically, it is known [184] that:

- Each LP over \mathbb{R} can be solved in polynomial time.
- The solubility of ILP is NP-complete.

Based on these definitions, LP is not only suitable to resolve the OLive-M state equation for an obstructed SecANet, it may be used directly to minimize the costs associated with the nodes of a SecANet to find a security-sensitive solution. Because the algebraic representation of Petri nets is based on integers, and the envisaged approach to add tokens and find a completion sequence implies integers (the tokens in Δ are supposed to be natural numbers, as well as the transitions in X , which can only be fired entirely or not at all), the subsequent approach will directly apply an ILP model accordingly.

OLive-M Realization by the Marking Equation and ILP: Given an obstruction marking m_{\otimes} and a final marking m_{end} , the marking equation $m = m_0 + \mathbf{N} \cdot X$ may provide the Parikh vector X , indicating which transitions must be fired to reach the final marking. To enable the transitions proposed by X , first, a live marking must be reached by adding tokens from variable Δ to the obstructed marking. With the help of a cost function $cost(X, \Delta)$ considering sequence length, the number of tokens, and user-defined costs (e.g., for security violations), an optimized solution with minimal cost for X and Δ is supposed to be proposed. The ILP model below shows the approach for using the marking equation:

ILP model for completing an obstruction state m_{\otimes}

Min $cost(X, \Delta)$ subject to:

$$m_{live} = m_{\otimes} + \Delta$$

$$m_{end} = m_{live} + \mathbf{N} \cdot X$$

$$X, \Delta \geq \mathbf{0} \quad X \in \mathbb{N}^{|T|} \quad \Delta \in \mathbb{N}^{|P|}$$

After an obstructed marking m_{\otimes} has been reached, the necessary tokens will be added to the deadlocked model to take the current obstructed marking to a final state. The ILP model above has two sets of variables, according to the equations (4.2) and (4.3): Δ is the addition of tokens to m_{\otimes} that takes to an unobstructed marking m_{live} , and X is the Parikh vector that will take from m_{live} to m_{end} . A solution to the ILP model will then jointly decide the necessary number of tokens and the consequent firings to be made to reach m_{end} . Remarkably, the cost function is a minimization that considers both the length of the sequence completing the workflow (through the Parikh vector X) and the number of tokens required to escape from the obstruction marking (the variables Δ), thus globally optimizing these two decisions. This thesis considers the cost as a user-defined function because, on the basis of different indicators, different costs could be assigned depending on the context; e.g., if the shortest path is preferred independent of the violations done, then one can set the Δ variables' cost to 0 (or substantially less than the X variables). On the other hand, if the number of violations should be reduced, the opposite cost can be set. Also, the cost for variables in the X vector may differ if, e.g., the firing of certain activities should be incentivized or avoided. The same holds for the Δ variables.

For instance, for the Petri net in Figure 3.24, analogously to the example solution before, the given ILP model (assigning unitary costs to both X and Δ) will find the solution $\Delta = (0, 0, 0, 1, 0, 0, 0, 0)$ (i.e., putting a token in the *SoD* place) and $X = (0, 1, 0, 0, 1)$ (with X according to the sequence $t_1, t_2, t_{u_1t_1}, t_{u_2t_1}, t_{u_1t_2}$). Assuming that the costs associated with transitions are normally > 0 , minimizing the costs of cyclic nets means that loops are avoided as far as possible. This then incentivizes that the process execution successfully completed as immediately as possible. The assignment on Δ and X variables defines the violations to make in order to complete the workflow. Ways to assess the impact and meaning of these violations in terms of security-sensitivity for the authorization, constraints, and users will be the subject of the discussion of the approach in Section 4.3.

4.2 OLive-M Experiments for Model-Based Obstruction Solving

The implementation of the realization of the OLive-M presented above was applied to the CEW SecANet shown in Figure 3.46.

4.2.1 Implementation

The implementation of the Olive-M obstruction solution bases on a Python implementation by Taymouri [198, 197] and uses Gurobi [100] as the ILP solver. The experiments were run on a MacBook Pro with 8 GB RAM and an Intel Core i7 3 GHz CPU. Based on an SecANet input file in PNML standard format the the optimal solution is returned as a list that contains an encoding of the Δ and X vectors.

To assign costs in the implementation, the Petri net type definition (PNTD) of place/transition nets (P/T nets) was extended with the optional addition of costs to places or transitions, resulting in P/T cost Petri nets (P/TCost-nets)¹. This PNTD redefines the value of P/T-nets with costs for places and transitions and inherits the marking and annotations from the official PNML P/T-net definition.

4.2.2 Remarks on Costs and Results

Note that no matter how costly, there may be places to which a token must be added to reach the final marking constraint (e.g., the SoD place in the DMV SecANet example). If there is a choice among multiple transitions or places (e.g., multiple SoD places to violate) to reach the final marking, the different costs assigned are crucial. The implementation can handle both unitary and differing costs. The optimal solution is obtained when the constraint regarding the proper final marking is satisfied. In general, the ILP solutions provided in the experiments are not unique. It is possible to have many optimal solutions, and all of them are correct.

4.2.3 Experiment Preparation

To allow the final marking to be determined only by the output place, further cancellation transitions were introduced at the end of the process. They consume all the remaining tokens in the places that were put on top of the initial workflow before the flattening; i.e., (un)assignment, SoD/BoD, and their corresponding capacity-1-complement places. Therefore, similar to the introduction of the re-enactment constructions to deal with loops, or, at the end of the SecA-WF-Net encoding in Chapter 3, a transition “reach_end” was added just before the end place p_o . Firing this transition produces a token into a place to which all cancellation transitions are connected. That way, this construction allows the cancellation of remaining tokens

¹ <https://github.com/iig-uni-freiburg/SEPIA/blob/ptcnet/res/pntd/ptcnet.pntd>

if required. In this way, the obstruction marking could be changed without the necessity to consider changes in the final marking; in particular, changes in the marking for the places resulting from the flattening.

4.2.4 Experiment Setup and Solution

The tool was able to provide a solution based on the net in Figure 3.46 with unitary costs assigned (cost = 1.0 for each place and transition) and given an arbitrary obstruction marking and a final marking to reach. To demonstrate this, an interesting obstruction marking could be reached after executing the following sequence:

$$\sigma_{\otimes} = \langle t_s, t_{m_1}, t_{f_1}, t_{dt_1}, t_{1\text{compute market value}}, t_{at_2}, t_{2\text{control computation}}, t_{\text{Computation correct?}_{\text{yes}}}, t_{o_3}, t_{bt_3}, t_{3\text{Evaluate Safeguarding Requirements}}, t_{\text{Safeguarding Required?}_{\text{yes}}}, t_{bt_4}, t_{4\text{Prepare Safeguarding}}, t_{m_2}, t_{j_1} \rangle$$

The obstruction marking resulting from this sequence is shown below, listing only places that contain $m(p) > 0$. For reasons of clarity, the marked places (with $m(p) = 1$) are categorized:

Workflow places: P_{j_1, t_5}

$$SoDplaces : SoD_{ct_1t_2}, SoD_{at_5t_1}, SoD_{dt_5t_3}, SoD_{dt_5t_4}, SoD_{dt_5t_2}$$

$$C1Cplaces : SoD_{at_1t_2c1c}, SoD_{dt_1t_2c1c}, BoD_{bt_3t_4c1c}, BoD_{dt_3t_4c1c}, SoD_{at_5t_2c1c}, SoD_{dt_5t_1c1c}$$

The solution provided required that a token be put into $SoD_{at_5t_2}$ on the basis of Δ and provided the Parikh vector X containing the following transitions²:

$$t_{\text{Approved?}_{\text{yes}}}, t_{\text{reach end}}, t_{at_5}, t_{\text{Approve Acquisition}}, t_e$$

However, running the tool again on the same obstruction marking and net provided a different optimal solution, namely adding one token into $SoD_{dt_5t_1}$ and firing a sequence containing the following transitions:

$$t_{\text{Approved?}_{\text{yes}}}, t_{\text{reach end}}, t_{dt_5}, t_{\text{Approve Acquisition}}, t_e$$

² The cancellation transitions are omitted here for the sake of clarity.

Hence, the approach obtained multiple correct solutions. The replayability for both of the Parikh vectors on the net was successfully checked with an extension to the same tool.

4.2.5 Experiments with More Extensive Nets

The same net was then concatenated step by step (again with equal costs) up to six times (see Figure 4.10 for an impression of the sixth concatenation) and encoded the obstruction marking above into it. For all cases, the solution proposed the addition of a token into either $SoD_{at_{5t_2}}$ or $SoD_{dt_{5t_1}}$.

More specifically, the obstruction marking was always encoded in the first of the concatenated nets. Therefore, in particular, the solutions assigning values to the X vector to reach the final marking increased.

Table 4.1 shows the statistics of the ILP that was solved³. Gurobi also computed the corresponding costs. Note here that transitions in the computed solution can happen multiple times because of cycles in the net. However, as indicated before, the occurrence of loops is minimized in the course of the optimization. Figure 4.9 shows a perfect linear relation between the execution time required and the size of the problem.

Table 4.1 ILP statistics

Places	Transitions	Variables	ILP constraints	Runtime	Size of $supp(\Delta)$	$supp(X)$	Total cost
69	82	151	220	0.304	1	17	30
138	165	303	441	0.578	1	51	74
207	248	455	662	0.810	1	85	118
276	331	607	883	1.081	1	119	162
345	414	759	1,104	1.392	1	153	206
414	497	911	1,325	1.712	1	187	250

³ The related process models can be consulted at <https://doi.org/10.6094/UNIFR/228177>. The archive file olivem.zip includes a manual that explains how to reproduce the results.

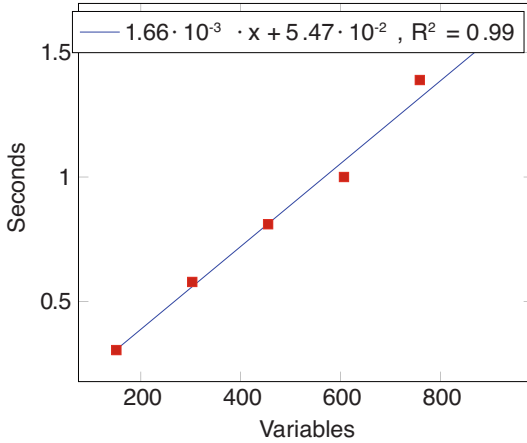


Figure 4.9 Experiment runtimes

4.2.6 Replayability

Because the solutions provided by the X vector do not provide a real ordering of transition executions, all possible linearizations were explored to assess where a solution obtained denoted a real sequence. This checking could be done for only the first four experiments because for the rest the exploration of possible solutions was very large. It is remarkable that for these four models, the solutions obtained represent a real sequence; i.e., there was a replayable linearization in the model.

4.2.7 Obstruction Position

To determine whether the position of the obstruction in the net affected runtime, an obstruction was incorporated into the last of the six concatenated nets (cf. Figure 4.10). The regarded net with 911 variables took 0.725 seconds to be solved, although the solution contained only 18 variables instead of 188. Hence, there were no substantial runtime differences from the same net with the same size but different positions of the obstruction marking.

4.2.8 Differing Costs

In a further experiment, various costs were assigned to $SoD_{dt_{5t_1}}$ (cost = 3.0) and $SoD_{at_{5t_2}}$ (cost = 5.0) to the net from the first experiment (with 151 Variables). Consequently, then the second solution containing $SoD_{dt_{5t_1}}$ became the only solution, resulting in a total cost of 32.0. Hence, the proposed solution was the most security-sensitive with the lowest cost.

4.3 Discussion and Potentials of the OLive-M Approach

Based on the conclusions and requirements from Chapters 1 and 2, this chapter addresses the obstructions that must be handled at runtime to restore the liveness of an obstructed process execution. Based on the given representation of the problem, possible solutions were identified. Finally, the solution, namely OLive-M, to finding a security-sensitive solution to complete the obstructed execution was deduced. By encoding the obstruction with a corresponding marking, the marking equation was used to provide a minimized Parikh vector to reach a completed marking, which if it is fired from the obstruction state, violates the firing rules. That way, on the basis of the model of a workflow and its authorizations and constraints, an obstructed state was solved not by changing the semantics of the model [25], but by finding the best path in the given model with minimal violation. Thereby, the language of a SecANet is temporarily extended by the words resulting from the obstructed sequences and the appended sequence that completes the obstructed execution. This solution allows the enforcement of an obstruction-resolving (RSC-1) in a security-sensitive way (RSC-2). In addition to the minimization of the solution, a certain threshold of security sensitivity could moreover exclude proposed solutions, because depending on the severity of the violations, it may be less risky to abort the process than to resolve the obstruction for an unjustifiable cost. The plans obtained to resolve obstructions could be realized automatically by delegation (RSC-3). For example, they could be used to recommend who should do which tasks in a break-glass situation, or as an assisted delegation, showing to the delegator potential best delegates (with the least violations). In all of this, the SecANet allowed searching for solutions that were fully aware of the obstruction (RSC-4). Although these requirements have thus been addressed to a certain extent, subsequent discussions of different aspects will point out the limitations but also the further potential of the presented approach.

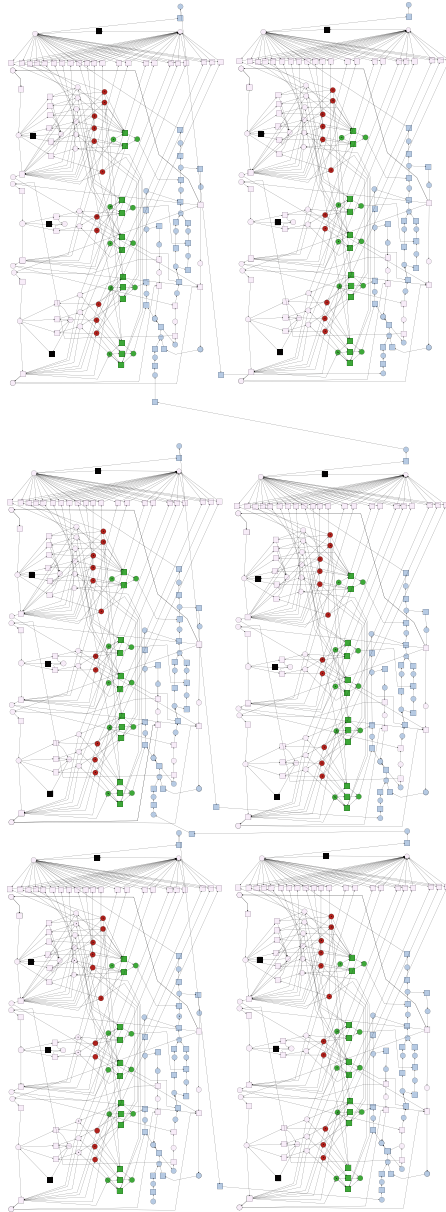


Figure 4.10 Impression of six concatenated nets

4.3.1 Limitations in Solutions

Based on the objective function that considers the costs assigned to the places and transitions and the calculated total cost, solutions can be identified as optimal; i.e., minimal. However, based on the fact that the solutions of the ILP can be spurious regarding their marking, and hence spurious in reaching the final marking, it may be possible for a solution to be minimal but at the same time not able to reach the final marking; i.e., there would be no respective firing sequence based on the given Parikh vector X . Hence, the solutions have the limitation that they do not necessarily represent the minimal violations required to reach the proper final marking. Here, results could be investigated further to trade efficiency for precision; i.e., by incorporating ordering constraints in the marking equation. Moreover, because the existence of a firing sequence based on a Parikh vector is guaranteed only for weakly sound-bounded free-choice models, and not for more general Petri net classes, the SecANet free-choice transformation identified in Chapter 3 could be further pursued here.

4.3.2 Efficiency

OLive-M represents a “light” technique to resolve obstructions as structural methods are more efficient than reachability analyses. In particular, the matrix equation itself can be solved in polynomial time. Computing the ILP and replay may be inefficient for larger instances. However, smaller-sized ILP instances can be solved efficiently in practice. Because a SecANet represents a plain, safe, and pure ordinary Petri net, the structural theory can be easily applied, with the data reasoning remaining mild. Although a SecANet is, e.g. not free-choice, the experimental results speak well for the SecANet (see Figure 4.9). Here, one could further customize the approach with regard to the SecANet application by further refinements or simplifications. If one assumes that only constraints may be violated by adding tokens one would not have to consider all other places for the addition of tokens such that the variables in the matrix equation, the number of possible solutions, and the complexity would be reduced. However, a comparison between Figures 4.3 and 4.4 indicates how the structural complexity of the SecANet approach is a tradeoff for efficiency, which indicates the greater amount of space that is used for the SecANet approach. As shown above (cf. Chapter 3), a free-choice transformation would further increase the structural complexity of the SecANet. With regard to the replay, just replaying all possible linearizations of the solutions could, in the worst case, result in exploring

all reachable markings. In contrast, given the marking equation and a free-choice net, the solution cannot be spurious and can be found in polynomial time.

4.3.3 Replayability

To determine replayability, the experiment basically tried to replay all possible linearizations of the Parikh vector on the SecANet. This, however, was a very simple and exhaustive way to check replayability, as indicated by the fact that only the replayability of the first four nets could be checked. Here, instead of checking whether every possible linearization of the Parikh vector is replayable, a more efficient way to check replayability would be to take the model into account. For example, given a Parikh vector that includes the transitions $t_{u_1 t_2}$ and t_2 , if the model always starts with $t_{u_1 t_2}$, there would be no need to try linearizations that start with t_2 . That way, an exploration algorithm could try to fire all transitions in the Parikh vector and thereby change the marking. In case there are several options of which transitions to fire based on the model, the algorithm could backtrack to what extent branches could be further replayed. More sophisticated techniques in this respect can be found in, for example, the works of Taymouri and Carmona [199]. Their replayability checks operate in the context of the computation of model trace alignments for conformance checking (cf. Chapter 2 on Process Mining), but could certainly be adapted to the SecANet context as well.

4.3.4 Emergency-SecANet

In contrast to the basic DMV SecANet example, the solutions of the CEW SecANet show that multiple solutions with various violations of constraints are possible. However, there may also be only one constraint, as in the DMV SecANet that can be violated such that the solution *must* involve the violation of that particular SoD constraint at no matter what cost. However, in a critical situation; for example, an emergency, there may still be further possibilities in the context of the processes that are not captured in the normal process specification. Therefore, to provide for more solutions to escape obstruction, one may consider not only violating constraints but also escalating user privileges or even introducing new users in the widened context of potential process participants. To address such exceptional situations, a so-called “emergency SecANet” could be defined. Such a net could then be used only in case an obstruction were entered. It would allow authorizing a user to a task that was not authorized in the initial policy and thus permitting further user-

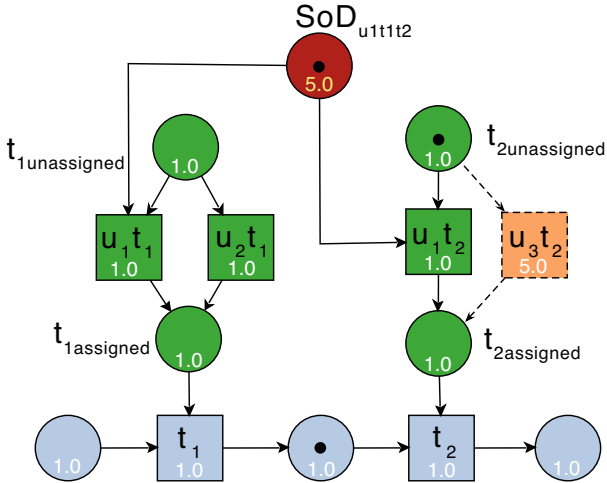


Figure 4.11 “Emergency-SecANet” with costly additional user-task assignment $t_{u_3t_2}$ that could be used to resolve the obstruction too

task assignments. Figure 4.11 shows such an additional user-task transition for the second tasks. Adding such a user could then be connected to a certain cost for, for example, the associated risk of privilege escalation. Alternatively, adding further user-task transitions could, for example, encode superuser rights. Further optional elements could be built into the SecANet in that way as well; e.g., additional constraint places. Thereby, the obstruction marking could easily be transferred to the emergency SecANet containing further net elements. Therefore, it would be important to allow only the additional constraints to contain a token, such that no existing constraints were enacted again. As a solution to resolve an obstruction in such an emergency SecANet could then foster empowering users who were not authorized before, instead of taking a too-high violation of an SoD constraint into account. In this way, a SecANet could be used to integrate break-glass scenarios and directly assess the best security-sensitive solution. This could also be developed to the point where a so-called “compensation task” would be introduced that even reduces costs; e.g., a task to review the affected case could have a negative cost assigned.

In conclusion, the main purpose of the OLIVE-M approach is to demonstrate a use case where, on the basis of a SecANet, Petri net-based techniques can be used to find solutions to complete an obstructed state. Based on the considerations, the

marking equation and ILP method were chosen to propose solutions to complete the workflow. Beyond the limitations and possibilities of the OLive-M approach, the matrix representation may also be used not only for solving obstructions but to help detect obstructions. More specifically, on the basis of the structural patterns in the SecANet encoding, one could also investigate whether there were certain structural patterns in the matrix representation in the SecANet—for example, in the incidence matrix—that could reveal obstructions beforehand, and that way simplify their detection. However, on the basis of the plethora of techniques that exist for or relate to Petri nets, the SecANet encoding certainly builds the basis for further approaches that would resolve obstructions differently. Here one could, for example, allow the reversal of markings, and by that search for reachable markings from the initial marking that come the closest to the obstruction marking and still allow for process completion. Because these reachable markings would belong to full firing sequences that would end in the desired final marking, this could save checking replayability. Or, one could search for firing sequences that are closest to the obstructed sequence and in that way search for the least distant or best aligning alternative path that is exceptionally allowed to resolve the obstructed state. Thereby, both approaches could incorporate costs as well. In fact, the latter can be compared to the approach idea in the next chapter, which will be considered with regard to the log-based resolving of obstructions. As identified in Chapter 2, indeed, based on the general framework for requirements (GR) embedded in all SecANet-related solutions and also the policy and the model, the log may be considered as well to allow the consideration of all inputs (GR-3). It could be applied to the OLive-M approach as well by deriving indicators, such as resource behavior indicators. In the sense of the aspired indicator-based security, these indicators could then also be incorporated in the costs assigned to the SecANet elements, and thus enhance the SecANet model.

4.3.5 Extending the OLive-M Principle to Analyze Satisfiability and Resilience

The two parts of the OLive-M principle, namely adding tokens and completing the execution in an optimal way, may have further applications. In the context of this work, and on the basis of the SecANet encoding, it will subsequently be shown how the OLive-M realization can be used to analyze satisfiability and resilience. Here, the graphical nature of the SecANet particularly would allow the direct identification of weak spots in an unsatisfiable WSP instance. Moreover, the marking equation could be used to propose which users to add to make a workflow executable. Therefore,

the notion of cost can be seen in a broader context because not only security-sensitive costing may be required, but also the costs of having employees available and assigned to a certain process.

4.3.5.1 Analyzing Satisfiability

Similar to the capture and detection of obstructions, an unsatisfiable workflow raises the question as to where exactly its weak point is; for example, to identify how authorization policies or the user base must be changed to allow for a valid execution plan. Therefore, the principle of the OLive-M approach can be adapted to the context of WSP solving in a way that allows (even graphically) the identification of such weak spots in workflows and their authorization constraints by the use of Petri nets.

More specifically, on the basis of a start marking m_{start} and a final marking m_{end} , the marking equation $m = m_0 + \mathbf{N} \cdot X$ can be used to find the Parikh vector X , indicating which transitions must be fired to reach the final marking. If the policy specification cannot provide a valid plan; i.e., no firing sequence can reach the final marking, a live marking must be reached by adding tokens from variable Δ to the start marking. Analogously to the ILP model above, with the help of a cost function $cost(X, \Delta)$ considering sequence length, the number of tokens, and user-defined costs (e.g., for adding staff), an optimized solution with minimal cost for X and Δ is proposed. The ILP model below sketches the approach:

ILP model to find a firing sequence from m_{start}

Min $cost(X, \Delta)$ subject to:

$$m_{live} = m_{start} + \Delta$$

$$m_{end} = m_{live} + \mathbf{N} \cdot X$$

$$X, \Delta \geq \mathbf{0} \quad X \in \mathbb{N}^{|T|} \quad \Delta \in \mathbb{N}^{|P|}$$

Hence, if there were a valid plan or firing sequence, no further tokens need be added; i.e., $\Delta = \mathbf{0}$, and X would contain only the transitions that must be contained in the firing sequence. In contrast, if Δ is not empty (or $\mathbf{0}$), this means that the workflow is not satisfiable. Δ then already indicates the weak spot of the policy; namely, for example, which user is missing at which task or which constraint is the foremost cause of the unsatisfiability. That way Δ provides an insight as to where the “bottleneck” occurs in the security-aware process. Here, a more in-depth evaluation could be further pursued that considers the efficiency of the ILP solution compared to that

of the usual WSP techniques; e.g., based on SAT-Solvers. Eventually, such a satisfiability analysis technique providing a (graphical) representation of authorizations and workflow in one model could help policy designers to pinpoint deficits in the policy specification causing the unsatisfiability.

4.3.5.2 Analyzing Resilience

Another line of research could be devoted to estimating resilience by using the structural theory of Petri nets. That way, a minimal amount of users required to successfully complete a workflow could be estimated, which would provide insight into the resilience of the workflow. Based on the presented idea to extend the approach with user-availability nets (cf. Section 3.3.2 on SecANet+), a SecANet could involve the encoding of user availability. Here, finding for instance the minimal amount of users to execute a workflow could be done by using the marking equation as well. More specifically, the “ILP model to find a firing sequence from m_{start} ” (see Section 4.3.5.1) for satisfiability analysis could be used because, in the resilience setting, based on the initial marking, the final marking is supposed to be reached as well. The difference would lie in the SecANet, which is extended by the user-availability encoding. Analogously to the ILP model for satisfiability, adding tokens to the places that indicate users’ presence, Δ is then supposed to indicate which users to add by the tokens in the places encoding user presence to make the process executable. Such an ILP model could then be used to compute various resilience levels introduced at the end of Chapter 3. The Δ -based approach would address primarily the static version of resilience. However, one could also consider the X vector for decremental or dynamic resilience and associate the costs of the respective transitions. This would cause user absence or user presence with positive or negative costs such that, for example, there would be incentives to make sure that users only must be there as long as they are required to run the process (so that making users absent would have a negative cost). That way, the structural theory could be used to actually propose where tokens; i.e., users, must be added (Δ) and when users could become present or absent (X) to make a workflow resilient. Thereby, depending on the respective costs or risks that adding or removing personnel would mean in reality, adding users and changing user availability could be promoted or prevented.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





OLive-L: A SecANet Use Case for Log-Based Obstruction Solving

5

In a metaphorical sense, event logs are a “force of the past” [159] that can be used to shape the present and predict future events. Similarly, in the context of this thesis, past process executions are intended to eliminate a present obstruction and predict the sequence of events for how the process execution can be completed (cf. the Log-based Completeness Requirements in Chapter 2.3). In addition to the model-based approaches, this chapter assumes there is historical information for detecting and resolving obstructions. The consideration of logs addresses the general requirements of completeness to consider all inputs (model, policy, log) throughout all approaches (GR-3). Similar to the obstructed path sequences of process models, obstructed traces represent obstructed states (GR-1). As identified in Chapter 2.3, various log-based techniques exist to derive a range of indicators to consider when assessing the security-sensitivity of solutions (GR-2).

Analogously to the model-based OLive-M approach, this chapter presents OLive-L to make obstructed sequences live again based on the log. The log-based completeness requirements derived in Chapter 2.3, representing the potentials for how logs can be used in the context of obstructions, are incorporated.

Leveraging this approach, this chapter presents a further use case of the SecANet encoding to demonstrate its applicability to log-based techniques. Although the SecANet approach forms the basis, OLive-L works independently because, depending on the information system used to execute a process, the process model or policy is not necessarily provided or enforceable, and only logs are available. However, the approach also leverages SecANet modeling, which is relevant in additional aspects, such as generating or classifying logs.

Figure 5.1 illustrates the OLive-L approach. To detect and separate obstructed and successful traces (RLC-1), the *traces* are first *partitioned* accordingly. After this preprocessing, the approach then proposes a partial trace of events, i.e., a segment of the successful trace, to find paths to complete the obstructed trace (RLC-3). Such

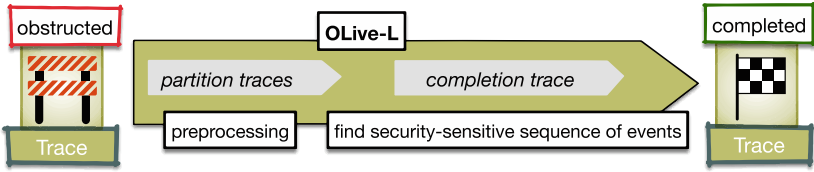


Figure 5.1 The Olive-L approach for resolving obstruction based on logs

a *completion trace* could violate a safety property, such as an SoD requirement, but eventually allows the liveness to be “enforced” (in case a PAIS is provided that steers the execution accordingly). The proposed solution is again assessed with regard to its security-sensitivity. Indicators that quantify the cost of the obtained completion traces (RLC-2) are identified based on the log, enabling a measure of security-sensitivity for selecting the most security-sensitive candidate.

Figure 5.2 illustrates the Olive-L approach with example traces of an arbitrary model that are categorized as “successful” or “obstructed.” Based on this partitioning of cases, the intention is to find the closest matches for the obstructed trace to the successfully executed traces. These nearest matches then propose the partial trace to complete the execution, and the example presented here results in two

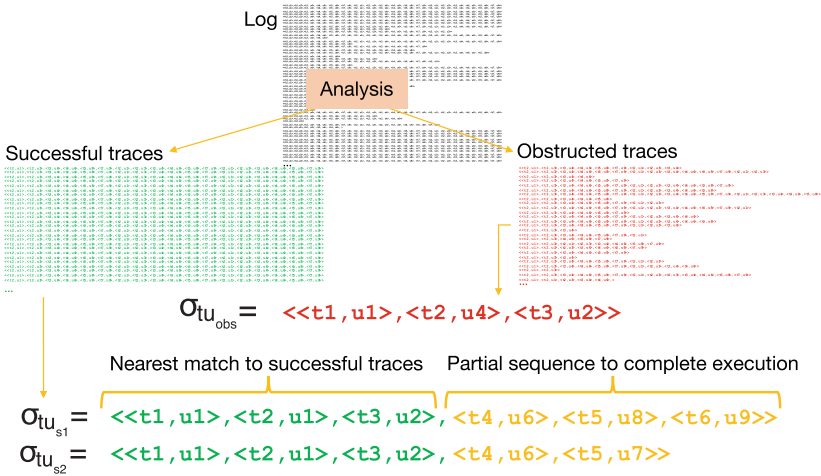


Figure 5.2 Log-based approach with example traces

candidates. An objective function tries to find, for example, the shortest completion trace, which could be justified based on the assumption that a shorter exceptional completion trace involving the minimal number of executed events may be less risky and more security-sensitive. If constraint definitions exist, then an objective function minimizes the number of violations a solution candidate implies.

Similarly to the OLive-M approach, this chapter demonstrates the applicability of OLive-L by using logs for resolving obstructed workflow executions. A subsequent illustration presents a realization of the OLive-L approach that exemplifies how process mining and machine learning methods are applied in this context. The OLive-L approach primarily addresses the obstruction resolving requirement (RLC-3), as well as depends on the partitioning of traces and the assignment of costs. Therefore, the other requirements of RLC-1 and RLC-2 are incorporated as essential building blocks. Chapter 2 previously examined various possibilities of using logs for the partitioning of logs (RLC-1) and how they can determine indicators and costs (RLC-2). This chapter complements these methods for partitioning by taking the SecANet into account to separate the traces.

Next, possible methods for exploring the similarity of traces are examined, resulting in identifying a common and applicable approach to provide solution candidates of successful traces that represent the closest match to the considered obstruction. Then, to finally select a candidate trace from which the completion trace is obtained, the ways to determine the security-sensitivity of candidates, i.e., the different costs they may imply, are illustrated. An implementation of the OLive-L approach offers experimental results based on a log synthesized from the CEW SecANet. Finally, a discussion considers further developments and extensions, such as the possibilities to assess security-sensitivity, and sketches how the logs may be used for the model-based approach.

5.1 Methods and Realization of the OLive-L Approach

This section identifies the methods to realize an implementation of the OLive-L approach, which builds upon the approaches and methods indicated in Chapter 2 that suggested the promising use or adaption of OLive-L. The required formalism is introduced, beginning with a trace replay performed on a SecANet considered for identifying the successful or obstructed traces. The partitioning of logs represents a fundamental initial step to provide further reason on the traces of the logs. For resolving an obstructed trace, the logs can also recommend or predict actions based on the behavior they reveal, such as how to complete the processes to achieve a positive outcome. Therefore, this section will then consider log-based techniques from

process mining, in particular those from predictive monitoring suitable for determining nearest matches to select and demonstrate an appropriate method to resolve obstructions. The partitioning of logs further enables deriving meaningful measures for identifying the most security-sensitive candidate trace, where such indicators and measures are considered under the notion of “security-sensitive costing.”

5.1.1 Trace Replay

To partition traces, a log may already provide enough information to directly categorize its traces as obstructed or successful, for example, by considering the attribute-based filtering (e.g., “pi_abort” in XES) or endpoints filter. Alternatively, the log may be classified by corresponding conformance checking artifacts, such as rule checking, replay, or alignments. Subsequently, based on the SecANet encoding, a straightforward way of conformance checking, the trace replay, is prepared and illustrated. If a fully replayed trace reaches the final marking, then it is considered successful. If it first reaches a terminal marking, then it is considered obstructed.

5.1.1.1 Trace-Related Notations

The basic definition related to traces and event logs is presented in the following.

Definition 5.1 (Trace and Event Log). *Given an alphabet of events, $T = \{t_1, \dots, t_n\}$, a trace is a word $\sigma \in T^*$ that represents a finite sequence of events. An event log $L \in \mathcal{B}(T^*)$ is a multiset of traces.*

In the current context, an event consists of the executed task. The user who executes the task is denoted as $\langle \text{task}, \text{user} \rangle$. Equivalently, a corresponding trace that contains events of such form is denoted as σ_{tu} . Accordingly, the obstructed trace in Figure 5.2 is given as $\sigma_{tu\otimes} = \langle \langle t_1, u_1 \rangle, \langle t_2, u_4 \rangle, \langle t_3, u_2 \rangle \rangle$, with the successful trace of $\sigma_{tuS} = \langle \langle t_1, u_1 \rangle, \langle t_2, u_1 \rangle, \langle t_3, u_2 \rangle, \langle t_4, u_6 \rangle, \langle t_5, u_8 \rangle, \langle t_6, u_9 \rangle \rangle$. Analogously to these language-related formalisms, the trace sequences beginning after the i -th position of the trace is indicated by σ_{tuSi} . Then, the partial trace to complete the workflow is $\sigma_{tuS3} = \langle \langle t_4, u_6 \rangle, \langle t_5, u_8 \rangle, \langle t_6, u_9 \rangle \rangle$.

Similar to full firing sequences, full traces indicate a sequence of events that is fully replayable on a WF-net by taking the net from the initial to the end markings. The sequence definitions from van der Aalst [3] are adapted and extended to this

notion, as well as the use of traces, so that partial and possibly incomplete traces can be defined and potentially concatenated with a completion trace.

Definition 5.2 (Concatenation of traces, set of log events). A trace σ appended with element t' is denoted as $\sigma \oplus t' = \langle t_1, \dots, t_n, t' \rangle$. Similarly, $\sigma_1 \sigma_2$ appends the trace σ_2 to σ_1 , resulting in a trace of length $|\sigma_1| + |\sigma_2|$. This can be simplified as $\sigma t'$ or $\sigma_1 \oplus \sigma_2$, respectively. For any log $L = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$, $L \oplus = \sigma_1 \oplus \sigma_2 \oplus \dots \oplus \sigma_n$ concatenates all traces into a single trace of length $|\sigma_1| + |\sigma_2| + \dots + |\sigma_n|$. Hence, $\text{supp}(\widehat{L \oplus})$ gives the set of all events that occur in all traces contained in the log.

Appending the completion trace $\sigma_{tuS|\sigma_{tu\otimes}}$ to the obstructed trace $\sigma_{tu\otimes}$ can be denoted as $\sigma_{tu\otimes}\sigma_{tuS|\sigma_{tu\otimes}} = \langle t_1, u_1 \rangle, \langle t_2, u_4 \rangle, \langle t_3, u_2 \rangle, \langle t_4, u_6 \rangle, \langle t_5, u_8 \rangle, \langle t_6, u_9 \rangle$.

5.1.1.2 Replay-Based Partitioning of Traces

To replay the traces on the SecANet, the events of the trace may need enriching. For example, an event that encodes which user executed a task is represented as a distinct user-task transition and a corresponding task. In turn, the traces must consist of events containing the name of the executed task t_i and the user u_j who executed it. For the case that the traces are not easy to map and replay on the flattened model, it is first shown how the replay can be prepared. For this, events of the form $\langle t_i, u_j \rangle$ are mapped to the transitions of the model:

Definition 5.3 (Replay preparation). For each event $\langle t_i, u_j \rangle$ of the traces σ_{tu} occurring in the log L_{tu} , the corresponding transitions of the flattened SecANet N , i.e., the corresponding user-task transition $t_{u_j t_i}$ that assigns the user to its task and the transition t_i indicating the task afterwards, are mapped to each other. By doing this, each event $\langle t_i, u_j \rangle$ of the trace σ_{tu} is transformed to the sequence $\langle t_{u_j t_i}, t_i \rangle$. The resulting trace is notated as σ_{utt} , indicating the order of the transformed events. Analogously, the log is denoted as L_{utt} .

The log replay algorithm is used to replay the resulting traces σ_{utt} [175]. The transformation from the BPMN model into a P/T-Net [78] and the conducted flattening

introduces transitions that are not visible in the log (e.g., the forks or joins that route the control flow).

Such invisible tasks are considered lazy. In other words, they might fire in order to enable succeeding visible tasks, i.e., the tasks from the BPMN model (t_i) or user-task transitions ($t_{u_j t_i}$), but never directly in the course of log replay because they do not have an associated log event [175]. If a trace σ_{utt} is replayable by applying the *log replay* algorithm and reaches the final marking (with only one token remaining in the end position of the WF-net), then the trace is considered *successful*. Thus, the corresponding original trace σ_{tu} is added to the set of successful traces L_{tus} . If the trace σ_{utt} is replayable and does not reach the desired final marking but some other terminal marking, then it represents an *obstruction* with its corresponding original trace σ_{tu} classified as obstructed $\sigma_{tu\otimes}$. Traces not fully replayable, such as those resulting from the aforementioned incompleteness or noise (cf. Chapter 2.3), are neglected.

5.1.2 Nearest Match

Based on the performed classification of logs, given an obstructed trace (cf. Figure 5.2) that contains the executed tasks and its executor (e.g., $\sigma_{tu\otimes} = \langle \langle t1, u1 \rangle, \langle t2, u4 \rangle, \langle t3, u2 \rangle \rangle$), the nearest match to the successful traces must be found to identify a partial sequence to complete the execution. To obtain the matches of traces that are in some way the “closest” to the obstructed trace, this section introduces the so-called k-nearest neighbor (kNN) search as a method to realize the OLive-M approach.

5.1.2.1 k-Nearest Neighbor

Identifying the nearest match and proposing an addition of events have strong similarities to the imputation of missing values for cleaning and imputing raw data. If the traces related to the OLive-M approach were considered as data points, then an obstructed data point could be imputed with the user-task events from the points that encode successful executions.

A popular imputation approach to correct missing values is based on the k-nearest neighbor search. For each instance that contains one or more missing values, the k-nearest neighbors are calculated, and gaps are imputed based on the existing values of the selected neighbors. The most commonly used similarity function to obtain the k-nearest neighbors for missing values imputation is a variation of the Euclidean distance that accounts for those samples containing missing values [27]. Along with being a typical method in machine learning (ML), predictive monitoring that

leverage ML approaches often use kNN in addition to support vector machines, artificial neural networks, decision trees, clustering methods, or regression trees [144, 202].

The kNN algorithm is based on the Parikh vector representation, introduced previously, allowing for an easy transition of the results to Petri net-related matrix equations and Parikh vectors, which facilitate combining the elements and solutions of the log- and model-based approaches. Therefore, kNN is suitable to search for similarities between the data points used to realize the approach presented here.

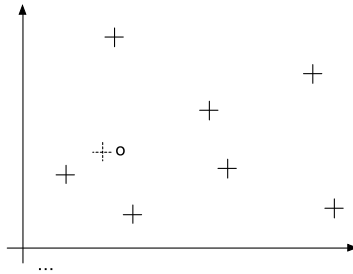


Figure 5.3 Sketch of obstructed and successful traces in n -dimensional space

Figure 5.3 sketches the points of successful traces related to an obstructed trace (o) that indicates the n -dimensional space in which kNN operates. Identifying those traces with the nearest distance to “o”, a variety of similarity metrics exist, such as the Manhattan or Cosine distance. Because the straightforward applicability of the approach is of primary interest here, the Euclidean distance that corresponds to the typical spatial understanding of a distance is introduced. The distance of each selected data point to all other data points is computed sequentially, i.e., the computational steps increase linearly with the size of the problem. In this respect, the caveat of kNN lies in the dimensionality because the required space increases exponentially with each added dimension.

5.1.2.2 kNN-Based Completion Trace

The kNN algorithm, dating back to Cover et al. [56], is adapted for use in the OLive-L approach by finding the nearest match between an obstructed trace and the successfully executed traces. The completion traces are then identified based on these k -candidates.

Definition 5.4 (Find k-nearest neighbors). Given a set of successful traces L_{tus} , an obstructed trace $\sigma_{tu\otimes}$, and a positive integer k , calculating the k nearest traces to $\sigma_{tu\otimes}$ is performed as follows:

1. For each trace σ_i in L_{tus} , assign its Parikh vector $\widehat{\sigma}_i$ to the n -dimensional space \mathcal{R}^n , where $n = |\{supp(\widehat{L_{tus}}) \cup supp(\widehat{\sigma_{tu\otimes}})\}|$.
2. Find the k nearest Parikh vectors of the successful traces $\{\widehat{\sigma}_1, \widehat{\sigma}_2, \dots, \widehat{\sigma}_k\}$ with minimal distance to the Parikh vector of the obstructed trace

$$\min_{\widehat{\sigma}_i \in L_{tus}}^k d(\widehat{\sigma}_i, \widehat{\sigma_{tu\otimes}}),$$

where d is the Euclidean distance metric $d(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$.

Given $\{\widehat{\sigma}_1, \widehat{\sigma}_2, \dots, \widehat{\sigma}_k\}$, the partial sequences of the corresponding traces $\{\sigma_1|_{\sigma_{tu\otimes}}, \sigma_2|_{\sigma_{tu\otimes}}, \dots, \sigma_k|_{\sigma_{tu\otimes}}\}$ contain all the events after the $|\sigma_{tu\otimes}|$ -th position of the trace, presenting the k potential sequences of events to complete from $\sigma_{tu\otimes}$.

If more than one candidate are found, then only one must be selected, for example, by an objective function that considers the length of the partial sequence to complete or the number of violations taken into account. For instance, if two successful candidates

$$\sigma_{tuS1} = \langle \langle t1, u1 \rangle, \langle t2, u1 \rangle, \langle t3, u2 \rangle, \langle t4, u6 \rangle, \langle t5, u8 \rangle, \langle t6, u9 \rangle \rangle \text{ and}$$

$$\sigma_{tuS2} = \langle \langle t1, u1 \rangle, \langle t2, u1 \rangle, \langle t3, u2 \rangle, \langle t4, u6 \rangle, \langle t5, u7 \rangle \rangle$$

are chosen as the nearest match, both having the same first three events $\langle t1, u1 \rangle, \langle t2, u1 \rangle, \langle t3, u2 \rangle$ in which only the executor of $t2$, namely $u1$, differs from the executor of $t2$ in the obstructed trace, $u4$, then the potential partial sequences of events to complete the execution, i.e., the completion traces

$$\sigma_{tuS1}|_{\sigma_{tu\otimes}} = \langle \langle t4, u6 \rangle, \langle t5, u8 \rangle, \langle t6, u9 \rangle \rangle \text{ and}$$

$$\sigma_{tuS2}|_{\sigma_{tu\otimes}} = \langle \langle t4, u6 \rangle, \langle t5, u7 \rangle \rangle$$

can be compared by an objective function. If the length of the partial sequence of events is minimized, then the completion trace $\sigma_{tuS2|\sigma_{tu\otimes}}$ is chosen to complete $\sigma_{tu\otimes}$.

5.1.2.3 Security-Sensitive Costing of Candidates

As examined in Chapter 2.3, the log can be used to enhance the model as well as the log by indicators, such that violations can be better assessed. Such a security-sensitive costing based on the log may consider a plethora of indicators that can be derived from the logs [17, 164, 165] and then incorporated and weighted into the overall cost of the related elements in the SecANet or the events of a log. More precisely, in addition to assigning indicators to places or transitions of a SecANet, these could also be assigned to each user-task event or each event that involves only a certain user or task. Then, the overall cost of each proposed candidate is summed to assess the solution in terms of its security-sensitivity.

This extraction of indicators and metrics used for security-sensitive assessments can focus on multiple aspects, some of which are presented in the following with examples:

- **Tasks:** The general relevance of a task can be assessed on the basis of the log, or how important a task is for successful execution (cf. Key Performance Indicators). Given that a log contains the appropriate attributes, identifying the tasks affected by unauthorized access in a break-glass case is possible. Accordingly, the possible cost of a violation affecting those tasks can be lowered.
- **User-task level:** Referred to as the user-task transitions of a SecANet or an access control matrix (ACL), the relevance of a corresponding permission (or user-task transition) is evaluated, depending on if a corresponding user-task authorization occurs in the log.
- **Data elements:** Interpreted in relation to tasks or users, exceeding a threshold (e.g., a credit of more than 5000 Euro), for example, could mean a higher risk for the tasks or users affected. As another example, dealing with a larger amount of money may improve the qualification and lower the risk (or cost) associated with a user.
- **Resources/Users:** In Chapter 2.3, many existing methods were identified, such as resource behavior indicators, as identified in Figure 2.11 and resource profiles. In addition, profiling techniques that identify the threat emanating from an insider could indicate how risky is the participation of a particular user in a process [34]. However, the egocentric perspective on an individual user can be considered, as well as the socio-centric perspective that relates the users with each other.

Specifically, the consideration of social networks that use different metrics for the type of collaboration of actors can be assessed from a security perspective.

To offer additional examples, the social network analysis provides different metrics that seem suitable for use in security-sensitive costing. A relationship can result, for example, from how often multiple actors are involved in the same process execution. This so-called “working together” metric indicates that the process is being handled well, but there remains an increased risk of collusion or fraud. Furthermore, considering which actors perform similar tasks is possible. A user who carried out a task similar to the obstructed one could be favored by the costs. Finally, as identified in Chapter 2, process logs can contain events that represent the execution or completion of tasks, as well as provide more detailed information about the state of processing, which can include the assignment of an activity to a specific actor, whose actual start of editing is recorded in another event. Other event types can define delegations, pauses, resumes, and the end of activities (see the Standard Transactional Life-cycle Model in Chapter 2). Metrics that refer to an event type can, for example, explicitly track delegations and derive information about the hierarchy of process participants to be used for role mining. Such hierarchies can then be associated with costs. For instance, if the head of a department carries out an activity that an employee would otherwise perform, then this activity results in a higher cost.

Based on the classification of logs, the feature weighting of the partitioned classes can assess the security-sensitivity of a solution. In determining the influence of individual features, i.e., the dimensions in an n -dimensional space, as classified as a successful or obstructed trace, a high attribution of certain user-task events suggests that the existence of these features is crucial in the selection of candidates. Thereby, based on the typical RELIEF Algorithm [129], for example, the vectors are determined with a dependence on both classes, the nearest hit to the class under consideration and the nearest miss from the other class. Assessing the importance of the different events is then possible to make the entire trace successful or obstructed, respectively. The k -candidates can then be multiplied with the obtained feature vector, such that the candidate with the highest summed up weight represents that it contains most of the user-task assignments that are crucial for successful execution, and thereby provides a “completeness measure.” A feature vector for obstructability can also be deduced by a feature weighting of the obstructed traces where the minimum value is chosen, as it represents an “obstructability measure.”

If there exists a SecANet model of the process, then replaying the overall trace $\sigma_{tu} \otimes \sigma_{tuS|\sigma_{m\otimes}}$, i.e., the obstructed and completion traces, can also indicate the missing tokens during replay firing. When the SecANet has costs assigned, the cost to

add missing tokens in the places, together with the cost to execute the transitions related to the events in the completion trace, sum to the overall cost of the solution.

5.2 OLive-L Experiments for Log-Based Obstruction Solving

An implementation of the OLive-M approach presented above is demonstrated through the following experiments applied to logs and traces related to the CEW SecANet.

5.2.1 Implementation

The implementation of the OLive-L obstruction solution was developed in Java 8 using the Apache Commons math library to calculate the Euclidean distance for the kNN algorithm. Based on two CSV files for the successful and the obstructed traces, and a positive integer value of k that encodes the vicinity to scan when finding the closest matches to a given obstructed trace, it returns a list containing at most k closest vectors, the related traces, and their distance from the obstructed trace vector. These experiments were conducted on a MacBook Pro with 8 GB RAM and an Intel Core i7 3 GHz CPU.

Table 5.1 Encoding of successful traces in 12-dimensional space

at1	ct1	dt1	at2	ct2	dt2	bt3	dt3	bt4	dt4	at5	dt5
0	0	1	0	1	0	0	1	0	1	1	0
1	0	0	0	1	0	1	0	1	0	0	1
0	1	0	1	0	0	1	0	0	0	0	1
0	1	0	0	0	1	0	1	0	1	1	0
0	1	0	0	0	1	0	1	0	0	1	0
0	1	0	0	0	1	1	0	1	0	1	0
0	0	1	0	1	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0	1	0	0	1
0	0	1	0	1	0	0	1	0	0	1	0
1	0	0	0	1	0	1	0	0	0	0	1
1	0	0	0	0	1	1	0	0	0	0	1

5.2.2 Experiment Preparation: Obtaining Traces

Comparable to real WSP instances, because acquiring real-world traces with a corresponding model along with all the authorization data required to perform the described analysis is difficult, successful and obstructed traces were generated by playing out firing sequences of the flattened Petri net from Figure 3.46 (i.e., sequential firing of the enabled transitions until an obstructed or a final marking is reached). With this data generation method, both evaluations build upon the same model to compare the results.

After generating the traces, the events were mapped to the users who executed each, according to the flattened user-task assignment, and filtered only by the relevant user-task events (in a real-world log, such events would contain the task name with the executing user/originator, cf. Definition 5.3). From this, successful and obstructed traces conforming to the user-task assignment and SoD/BoD constraints were generated. For each trace of the form σ_{tu} , the corresponding Parikh vector $\widehat{\sigma}_{tu}$ was built and assigned to the n -dimensional space. Table 5.1 displays the successful Parikh vectors of the traces as assigned to a 12-dimensional space, based on all possible user-task assignments.

Table 5.2 Solution for $k = 5$ with highlighted partial sequence

distance	closest vector	related trace
1.732	0,1,0,1,0,0,1,0,1,0,0,1	$\langle \langle t1, c \rangle, \langle t2, a \rangle, \langle t3, b \rangle, \langle t4, b \rangle, \langle t5, d \rangle \rangle$
2.0	0,1,0,1,0,0,1,0,0,0,0,1	$\langle \langle t1, c \rangle, \langle t2, a \rangle, \langle t3, b \rangle, \langle t5, d \rangle \rangle$
2.0	0,0,1,0,1,0,1,0,0,0,1,0	$\langle \langle t1, d \rangle, \langle t2, c \rangle, \langle t3, b \rangle, \langle t5, a \rangle \rangle$
2.236	1,0,0,0,1,0,1,0,1,0,0,1	$\langle \langle t1, a \rangle, \langle t2, c \rangle, \langle t3, b \rangle, \langle t4, b \rangle, \langle t5, d \rangle \rangle$
2.236	0,1,0,0,0,1,1,0,1,0,1,0	$\langle \langle t1, c \rangle, \langle t2, d \rangle, \langle t3, b \rangle, \langle t4, b \rangle, \langle t5, a \rangle \rangle$

5.2.3 Experiment Setup and Solution

The nearest neighbor of the successful traces to the corresponding obstructed trace was computed with the Euclidean distance measure. For comparability, the obstructed trace σ_{\otimes} from the model-based experiments, encoded as $(0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0)$, was also chosen. The solution for $k = 5$ is depicted in Table 5.2¹.

Trivially, if $k = 1$, then no decision is needed as to which partial sequence to choose. Interestingly, $\langle t5, d \rangle$ would be proposed, although the majority of

¹ The related event logs can be consulted at <https://doi.org/10.6094/UNIFR/228177>. The archive file `oliveL.zip` includes a manual on how to reproduce the results.

successful traces for $k = 5$ in Table 5.1 ends with $\langle t5, a \rangle$. Based on the candidates identified by the Euclidean distance, the $k = 5$ solution requires selecting one of these candidates with their completion trace by considering security violations or the minimum length of the partial trace. As the partial trace, i.e., the completion trace, is selected at the $|\sigma_{\otimes}|$ -th position, the second and third solutions are empty. Because the remaining completion traces have the same length, the length-criterion to identify completion traces is neglected.

To assess the security violations of the completion traces, the obstructed trace is checked against the different solutions and impacts on the given SoD and BoD constraints. Similarly to the model-based approach, both solutions, $\langle t5, a \rangle$ and $\langle t5, d \rangle$, violate one SoD constraint. However, reviewing the set of partial sequences provided in Table 5.2, a majority for $\langle t5, d \rangle$ can be identified. Additional techniques and corresponding limitations due to the uncertainties that a log may entail are discussed below.

5.2.4 Experiments with Extensive Logs

To illustrate the applicability of this implementation to real-world logs, the performance was tested on a large data set with a 65-dimensional space and 247,192 traces. This log did not allow partitioning based on a SecANet because no authorization specific data was available. Therefore, the traces were related to a randomly selected trace to assess the performance, and the qualitative aspect of the result had to be neglected. In this experiment, finding the five nearest neighbors for one trace required 0.31 seconds, which suggests the efficiency of the approach.

5.3 Discussion and Potentials of the OLive-L Approach

In realizing the OLive-L approach, this chapter first proposed an additional way to detect and separate obstructed and successful traces (RLC-1) by replaying traces on the SecANet model. The kNN method was leveraged to find the traces that complete obstructed execution (RLC-3). Depending on the actual capabilities of process monitoring, control, and enforceability that a PAIS provides, the solutions for the completion traces may only present a recommendation on how to proceed. Alternatively, the PAIS may steer the obstructed process towards its completion by the obtained completion trace. To determine the security-sensitivity of the solution candidates, in addition to examining the log-based indicators, the SecANet model quantified the costs of the proposed solutions (RLC-2). Based on the feature space

spanned by all user-task events, the feature vector was illustrated as to how it can evaluate the significance of the events in a candidate trace, for example, with regard to the successful execution of the process. Because the presented methods represent one possible approach to realizing the building blocks of the implementation, the limitations that exist in each step, as well as possible improvements, are discussed in the following.

5.3.1 Log and Partitioning

The solutions strongly depend on the size and quality (e.g., in terms of noise or granularity) of the log, especially to the extent to which successful and obstructed traces appear. At the same time, the log allows considering only those executions that are relevant in practice.

The possibilities of partitioning traces are extended by the SecANet model. Although the result of this partitioning may initially appear as just a reflection of the firing sequences of the “played out” model or the terminal language of the net, considering the users and tasks that are part of real processes make a difference. The log reflects a concrete selection of real-world process executions, such that the possibilities to solve an obstruction are reasonably limited. Therefore, a model-based solution might lack practical relevance if it does not represent a solution within the log-based approach.

Another issue concerning the replay is that traces not fully replayable are neglected. However, these traces could still indicate “good” outliers that result from a break-glass situation that was resolved and reviewed, so they are no longer replayable by the (idealized) model. An event attribute could additionally indicate the trace as completed, such that it would be included via attribute-based filtering.

Apart from replaying the traces, further conformance checking artifacts, such as rule checking or alignments, could use the SecANet encoding to partition traces. For example, the alignment of the log traces to the firing sequences of the SecANet model could be computed such that deviations from the synthesized traces are relatable to violations and then introduce more classifications.

5.3.2 kNN and Selection of Completion Segment

Although the kNN-solution provides a way to escape an obstructed trace, the necessary assumptions leave room for discussion and improvement. Variants for finding the nearest neighbors, such as other distance measures of Manhattan, Cosine, or edit

distance, could be explored. Alignments could also be used as an additional similarity metric to obtain the nearest matches of an obstructed trace to those successful executions.

When determining the completion trace, the approach also suggests opportunities for refinement when considering which events of the successful trace must be contained in the completion segment to complete the obstructed trace adequately. In contrast to the model, many uncertainties exist in the traces. For example, the obstruction does not necessarily occur in the final task of the obstructed trace. The reason for the obstruction might appear early in a trace, and concurrent activities could follow. Currently, the length of the obstructed trace is key for choosing the completion segment, i.e., the completion trace, because traces of equal length may have similar execution paths, so they already inherit a certain similarity. Candidate traces that have the same tasks executed after a certain length within the total of the obstructed trace could additionally be required. However, if this is not the case, then the completion segments of a candidate may become too short, such as in the second and third experimental candidates in Table 5.2.

Therefore, other techniques could be considered to determine the partial trace. The last common task of the obstructed and the considered candidate trace when traversing the traces from left to right might be considered. Alternatively, the Parikh vector of the obstructed trace could be subtracted from the Parikh vector of the candidate trace. Then the remaining events in the Parikh vector of the successful trace could be executed according to the order of occurrence in the corresponding trace. Based on predictive monitoring approaches, the next task of an obstructed trace could be predicted, such that the subsequent event or the starting point of the completion trace may be determined. Finally, by using a SecANet model, traces could be replayed and obstruction markings identified. Then, the marking obtained related to the obstructed marking can be checked, along with a determination of the remaining activities to be executed. For simplicity in this implementation, however, the length of the obstructed trace was chosen to be the most straightforward to illustrate the applicability of the approach efficiently.

5.3.3 Security-Sensitive Costing

Additional factors could be considered (e.g., in an objective function) to better assess the security-sensitivity. Implications regarding security violations from choosing the closest trace could also take additional features into account by conformance checking of an SoD rule for classifying the traces accordingly. A feature weighting could further partition traces regarding violations. Having logs partitioned into classes

of “conforming” and “violating,” a corresponding feature vector could then perform a security-based weighting instead of a pure success-based weighting. From this feature weighting, the events that contribute to violations could be indicated.

Additional indicators and measures could be refined using the manifold approaches sketched by machine learning and process mining. For example, the partitioned obstructed traces for predictive or prescriptive monitoring could predict obstructions to direct avoidance. Alternatively, these traces could be hypothetically completed when considering each as a completion trace to ensure that the proposed solution has a low risk of being obstructed again. Because the current focus, apart from the feature weighting, is on the successful traces, the advantage of this knowledge within the obstructed traces can be leveraged.

5.3.4 SecANet Discovery

If no SecANet is available, then mining a SecANet in the course of “process discovery” is conceivable. In this case, discovering all aspects separately is advisable, including the control flow and all user-task assignments of the complete executions. Some reasonable assumptions must be made for mining the SoD or BoD constraints. For example, the pairs of tasks that usually involve different users at the case level could be investigated and then defined as SoD constraints for these tasks. As identified in Chapter 2.3, process mining techniques focus on various resource perspectives, e.g., role mining, that may be used or adapted here. Then, an adequate process discovery method can obtain the control flow, user-task assignments, and constraints as inputs based on which of the usual SecANet encoding, as described in Chapter 3, could be performed, enabling the discovery of a SecANet based on a log to which the OLive-M approach is applied.

5.3.5 Log- and Model-Based OLive Extensions

When assessing violations, the SecANet is already considered by a replay to identify missing tokens during firing. However, additional specialized ways are sketched below that not only consider the SecANet but the respective model or log-based counterparts of the OLive approaches to resolve obstructions. The sequences of user-task events (σ_{tu}) and user-task transitions σ_{utt} are assumed to be mapped according to Definition 5.3. Therefore, the advantages of how both methods of the log- and model-based technique can be combined are explored in the following.

5.3.5.1 OLive-LM: Refining the Log-Based approach with the Model-Based approach

In addition to using the SecANet to assess security-sensitivity, elements of the model-based OLive-M approach can assess violation. For example, the obstruction trace on the SecANet can be replayed to the end in the obstruction marking, followed by the candidate trace without the partial trace for completion. From the perspective of the model-based approach, the resulting marking represents a live marking that must be reached from the obstructed marking to fire the partial trace to complete the execution. By subtracting the place vector of the obstructed marking from the place vector of the live marking, the result could reveal the tokens that must be added to complete the execution. If there are no positive integer solutions of the tokens to add, then the solution may be too far from the model, such as when tasks are skipped.

Eventually, in the case of the resulting vector ≥ 0 , based on the added tokens and the events in the partial completion trace, the costs of the related places and transitions can be summed. To perform such a cost-based evaluation, the corresponding costs can then be assigned to the model during the log-based model-enhancement.

5.3.5.2 OLive-ML: Refining the Model-Based Approach with the Log-Based Approach.

Just as the OLive approach can be leveraged for the log-based approach, logs can be integrated into the model-based OLive-M approach to provide more realistic and possibly faster solutions. Two possibilities are considered in checking the solution Parikh vector X of the OLive-M approach with the corresponding traces or directly inserting the successful partial traces of the log as a Parikh vector X into the marking equation.

Using Logs to Address the Problem of Replayability: The first option of checking the solution Parikh vector X of the OLive-M approach with the corresponding traces addresses the problem of replayability. In contrast to the replayability of the Parikh vector, the log replay has the advantage that the traces in the partitioned log already contain an order that can be deduced by the trace tuple or by the timestamp of the corresponding trace event, if provided. After the ILP model involving the OLive-M state equation is solved, the resulting X vector is related to the logs. The solution also provides a live marking m_{live} that consists of the obstructed marking m_{\otimes} and the addition of the tokens in Δ .

Instead of checking the replayability of the solution vector X , the transitions or events in X could be checked if they are completely contained in some of the successful traces. To filter only those traces that correspond to the solution, X must

be fully contained in the Parikh vector of each of the considered traces σ_{uttS} , i.e., $\sigma_{uttS} \in L_{uttS}$ and $\widehat{\sigma_{uttS}} \geq X$. For each identified trace $\widehat{\sigma_{uttS}}$, the replay of the trace without the events in the Parikh solution, i.e., $\widehat{\sigma_{uttS}} - X$, can be checked if it ends in m_{live} . This could either be checked by a SecANet replay, or directly by the marking equation $m_{live} = m_0 + A(\widehat{\sigma_{uttS}} - X)$. Therefore, the replay on the model directly excludes those traces that have gaps in the firing sequences, which occur in the subtraction of the Parikh vector. In contrast, the extent to which checking the replay by the marking equation is sufficient would have to be considered.

Finally, if the live marking obtained after replaying the sequence is equal to the live marking obtained by the OLive-M solution by adding Δ to m_{\otimes} , then X is replayable because it contains all the events that have not yet been replayed from the successful trace. This combined model- and log-based approach excludes spurious solutions and eliminates the possibly exhaustive replay analysis of the Parikh vectors, as described in Chapter 4. Moreover, relating the ILP solution of X to a limited set of successful traces does not necessarily suggest a restriction on the possible ILP solutions. If no corresponding trace is available to check X , then the replayability of X can still be examined, as described in Chapter 4. By using logs to check replayability, the combination of the model and the log shows a method, assuming that a small ILP instance can be solved efficiently, for how a solution that resolves obstructions is achieved efficiently.

Using Logs to Address the Problem of Larger ILP Instances: The second option of inserting successful partial traces of the log directly as the Parikh vector X in the marking equation allows for solving a system of linear equations instead of an ILP instance. A simplifying assumption for this method is that only successful traces with the same tasks in the same order as in the obstructed trace are selected. The completion segment of such a successful trace $\sigma_{uttS} \in L_{uttS}$ is denoted as $\sigma_{uttS|\sigma_{tu\otimes}|}$, where σ_{\otimes} denotes the sequence that leads to the obstruction marking.

As mentioned above, this choice of the completion segment can be significantly more multi-faceted and differentiated. Although the simplifying assumption allows for a fast identification of possible solutions by setting $X = \widehat{\sigma_{uttS|\sigma_{tu\otimes}|}}$, additional solutions could be lost. By inserting these values assigned to X , every linear equation has only one independent variable such that Δ can be directly solvable. Based on these observations, the solutions can be further restricted by Δ such that $0 \leq \Delta \leq 2$ in which the result only allows for the addition of single tokens. A solution means that the process can be completed by adding the obtained tokens Δ and firing the completion sequence $\sigma_{uttS|\sigma_{tu\otimes}|}$. As before, the costs can be identified for each possible solution based on the assigned cost in the model. After solving each possible successful trace, the solution with the least cost can be identified. While this solution

approach is efficient for systems of linear equations, smaller ILP instances may be more exhaustive. With increasing problem sizes, a considerable amount of space is required but remains efficiently solvable compared to larger ILP instances.

In summary, while the OLive-M approach may be practical for smaller ILP instances, depending on the sizes of the log and the possibly larger ILP instances, the first or second options presented above must be determined as to which is more suitable to enhance the OLive-M approach based on logs. So, the logs can build a solution base to be included if necessary. In both approaches described in this section, the finite nature of the logs could be used to ease computation and limit solutions to realistic possibilities. The additional computing steps are light, such as the linear search of comparing the X vector with each Parikh vector of the successful traces. In addition, by the Parikh mapping, all traces can be transferred as points into an n -dimensional space, such that a point encodes multiple traces that have been used to execute the same events in a different order. This simplification of the representations of the traces to check reduces the search space and memory requirements. A log already contains a certain degree of evidence for how the real-world executions may work, so by using logs, the results of the model-based approach can be adjusted to reality. As a drawback to using logs, the theoretically conceivable solutions that do not appear in the log are potentially suppressed. Complementing logs with successful traces synthesized from the SecANet model can be considered to counter this scenario. However, because synthesis means computing the reachability, this must be done in a workable way. Otherwise, only checking the replayability when required is more appropriate.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





Towards Intelligent Security- and Process-Aware Information Systems

6

The innovative and disruptive power with which the digital transformation changes organizations along with more regulations force enterprises to automate the implementation of regulatory rules in their information systems. The tendency of these regulations to unintentionally impede the operative business is consequently increasing. Although the concepts of governance, risk, and compliance management set a framework for the realization of regulations, their implementation in enterprise information systems for automating business processes can lead to technical obstructions, which occur from the enforcement of access control security that blocks the execution of business processes or an additional combination with an unexpectedly diminishing user base. The potential occurrence of such obstructions has an increasing effect on the high-level of uncertainty organizations must deal with. However, this challenge has not been sufficiently addressed. By developing methods for the analysis, detection, and handling of obstructions, this dissertation contributes to the engineering of information systems that provide flexible solutions to harmonize the conflicting goals of business and security at the process level. The introduction of indicator-based process security extends classic IT security concepts and provides a conceptual basis for the work. By considering indicators for compliance, the processes once again perform within the comprehensive frame designated by corporate governance. In this way, an obstructed process execution may still be completed within compliance while recognizing security requirements.

The Petri net-based SecANet representation covers the different aspects of a security-aware business process that incorporates indicators by providing a comprehensive starting point for analysis, detection, capturing, and resolution of obstructions. The model-based OLive-M approach then resolves an obstructed state with minimal security violations by considering the indicators assigned to the SecANet nodes as costs. Moreover, based on an obstructed trace, the log-based OLive-L method detects the most similar historical successful trace to complete

the process. Both approaches constitute use cases of the net and propose to which users the tasks are to be assigned to security-sensitively resolve obstructions. Therefore, the SecANet, OLive-M, and OLive-L methods compose a holistic approach that addresses obstructability by considering all inputs (model, policy, log) resulting from the design, runtime, and audit phases of a security-aware process. More specifically, the contributions of this thesis are summarized as follows:

- This work derives the paradigm of security in business processes by extending the safety-oriented classic IT security paradigm of “keep bad things from happening” to a liveness-oriented indicator-based view of “make good things happen” that pursues the process goal. When the enforcement of safety properties blocks process execution, the interface provided by the inclusion of indicators enables additional security-related insights from the data to be considered (e.g., the fraud risk of an initially unauthorized user-task assignment) for assessing the compliance of potential solutions. Therefore, just as the digitization of processes fosters obstructions, another essential component of the digital transformation, namely data, forms the basis for handling these obstructions.
- The systematic review of existing research on security-related obstructions in PAIS within the context of the design, runtime, and auditing phases of business processes provides a foundation for further research in this field. While the existing literature provides approaches for related problems concerning the design and execution phases, e.g., the workflow satisfiability problem (WSP) or resilience, this work illustrates the potential use of logs within this respect for the first time. In particular, the extraction of information from process log data using process mining, e.g., methods for generating statistical information about the outcome of a process (i.e., predictive monitoring) or by checking the conformance of (safety) properties on process traces, can determine a wide range of indicators based on realistic process behavior. Requirements for the introduced notions of obstructability and completability of security-aware processes are deduced by examining the distinct phases of process executions and their entities.
- The developed SecANet approach captures all aspects of a security-aware process into a formal representation integrating indicators as costs to explore security-sensitive behavior. This approach establishes a profound and expandable framework that explicitly addresses obstructions in security-aware workflows and provides comprehensive support for subsequent analyses and handlings. In contrast to the typical structural assumptions of security-aware processes in the context of WSP research, the representation can model specifications that contain conditional branching tasks and cyclic behavior, which are strongly motivated by their occurrence in real-world applications. The SecANet approach

provides a detailed basis to answer questions about obstructions in business processes against existing research. The approach also facilitates the application of existing Petri-net (or workflow net-related) analysis methods, such as an analysis of the language-based properties of a SecANet, the introduction of SecANet soundness, as well as the possibility to create workflow-net characteristics for a SecANet (SecA-WF-Net). The developed SecANet+ approach integrates additional restrictions modularly, e.g., encoding the user's (un)availability to investigate resilience.

- The developed OLive-M approach demonstrates for the first time how, by using a SecANet, obstructions can be solved by practically handling them, instead of changing the policy beforehand or preventing their occurrence. In finding a solution, the approach minimizes the number of violations while completing the process simultaneously. While the approach provides the framework for the security-sensitive handling of obstructions, the specific details of the weighting and determination of related indicators that still act in a compliant framework depend on the organizational and regulatory context. In practice, an organization could define a level or threshold of security-sensitivity such that solutions within these thresholds are assumed compliant—for example, the risk of the security violations associated with the security-sensitive completion of the blocked process execution is lower than the associated risk of damage.
- The developed OLive-L approach represents a SecANet use case for the log-based resolution of obstructions that, given an obstructed trace, proposes a completion trace. In contrast to the designed process model, logs already contain traces that encode how emergencies or other unforeseen exceptional circumstances (e.g., suddenly absent employees) were handled before and approved by audit. Such behaviors encoded in the log can additionally enable alternative process execution to solve obstructions. Moreover, the OLive-L and OLive-M approaches can beneficially complement each other by deducing indicators by comparing completion traces with the model or relating theoretical results against the background of actual process behavior encoded in the log.

The developed methods are evaluated as effective. The SecANet allows for the identification of obstructions, and the construction of SecANet models can be formally proven that the behavior of the original process model is preserved. Theoretical worst-case considerations show that the complexity of the SecANet encoding has polynomial runtime behavior and constant space requirements. The increased structural complexity that a SecANet entails must be weighed against its added value. Empirical analyses based on generated security-aware process data and real-world examples provide strong indications that the complexity of solving SecANet-

based WSP instances undercuts the runtime of typical SAT-Solvers. The analysis of obstructability is more complex however still moderate for mid-sized problem instances. The OLive-M and OLive-L approaches effectively resolve obstructed executions through experiments that find solutions to correct obstructions, even in larger models or logs. For common smaller problem instances, the experiments suggest that the methods discover solutions efficiently in terms of time and memory consumption.

This work tackles the uncertainty of if a security-aware process contains obstructions that paradoxically result from the intention to achieve a supposedly secure state through the enforcement of security controls. Thus, handling such obstructions can improve security in business processes. By adding process execution sequences representing compliant behavior, resolving obstructions in a security-aware and process-aware information system extends the intrinsic limits of mechanisms that enforce safety-oriented access control security. By opening up security-sensitive behavior that complements classic security concepts, business goals can be considered along with reducing the risk of fraud when computing solutions to obstructive situations. The approaches contribute to the security-aware automation in PAISs and facilitate the implementation of the increasing number of regulations and regulatory changes. The “digesting” of regulation and interpretation regarding the weighting of risks and costs remains a challenging problem and must be approached based on the entrepreneurial context from which this thesis abstracts. In a broader context, by developing automated methods that consider empirical aspects, this work contributes towards the advancement of reliable, agile, and autonomous solutions in information systems that leverage recent advances in big data analytics, artificial intelligence, and machine learning. In addition, research in future access control systems for PAIS is expanded within the fields of information systems and cybersecurity, which are also concerned with economic benefits and solving practical problems.

6.1 Application

Along with contributing to existing research related to satisfiability and resilience, this work provides a solid foundation for future research in the field of obstructability in security-aware processes and the development of software tools for the analysis and handling of obstructions. The integration of the developed methods into a software solution for the security-aware analysis of business processes, the Security Workflow Analysis Toolkit (SWAT), provides a foundation to transfer the contri-

butions of this work¹. SWAT contains methods for the preventive model-based or forensic log-based analysis of business processes and verification of security properties to show the principal integrability of the developed methods in a PAIS. These methods enable process designers and auditors to bridge the gap between the technical level on which corresponding methods operate and the business level they interpret.

In particular, the WF-Net-oriented Petri-net editor developed therein allows for the analysis of obstructability and satisfiability². Simultaneously, based on the encoding, existing Petri net analysis tools can be used for the reasoning on obstructability and satisfiability, which lowers the hurdle of leveraging these approaches. The model- and log-based techniques integrate additional solvers (ILP) and software libraries (kNN), respectively. To optionally assign costs to the nodes in a SecANet, these approaches rely on P/T cost Petri nets (P/TCost-nets), which extend the Petri net type definition (PNTD) of Place/Transition nets used in SWAT.

The use of the developed methods in a PAIS could manifest in practical applications. In the design and audit phases, the methods can make the security-aware process specification less obstructive, satisfiable, more resilient, or assess the associated risks, e.g., of obstructability. The occurrence of obstructive sequences of user-task assignments and task executions, which are subjected to a more detailed investigation, can be retraced and visualized in detail with the help of the Petri net model representation. Concerning the structural complexity of SecANet models, the concentration on self-contained security-aware sub-processes, i.e., only users exclusively authorized for tasks of the sub-process, seems to be useful to maintain clarity. During runtime, applying the developed methods to resolve obstructions could recommend who performs which tasks, for example, in a “Break-the-Glass” situation, or as an assisted delegation, showing the potential best delegates (with the least violation) to the delegator. However, to adequately tackle the implementation of regulations, automating the handling of obstructions is crucial. In this example, the efficient solvability of practical problem sizes of security-aware processes suggests that solving obstructions in a timely (online) fashion is realistic. The automated application of the methods in a PAIS then mimics an autonomous delegator that assigns outstanding tasks to the appropriate users based on experience, competence, and expertise. A PAIS usually offers task assignments to its users, typically as work items, so could also provide additional mitigating actions by creating “break-

¹ See <https://doi.org/10.6094/UNIFR/228177> for a manual on how to use SWAT and related tools in the SecANet context.

² For example, successfully checking the WF-Net conditions “option to complete” or “no dead wf-task” resolves a SecA-WF-Net as obstruction-free or satisfiable, respectively.

able” work items. Similar to typical “Break-Glass” scenarios, these could imply that the resolved obstructed cases are prioritized for audit.

6.2 Extension

The developed methods, especially the framework established by the SecANet approach and the theoretical basis, offer room for extension and adaptation into further questions of security-aware business processes. Three possible directions are identified in the following.

6.2.1 Beyond Security-Sensitivity: Multi-Objective Solutions

The OLive-M approach can be used to determine the resilience of a uniformly costed SecANet to estimate the minimal amount of users needed to complete a workflow. In addition, it could exclusively incorporate resilience-oriented indicators, e.g., the probability of user presence or the working together and handover-of-work metrics from social network analysis. Then, in the case of a non-resilient workflow or an unexpected user-absence that obstructs the execution of the process, a resilience-sensitive solution can be identified. Although security-related obstructions can correlate to the sudden non-availability of users, security and resilience remain at odds. A security-aware business process without any security requirements are trivially the most resilient and require only one user who is allowed to perform all tasks. The solution of a security-related obstruction with additional consideration of the resilience must lead to a trade-off solution. Multiple cost dimensions beyond security-sensitivity alone should be included and weighted to find an optimal solution for adequately addressing this multi-objectivity. As a plastic example, depending on the entrepreneurial and regulatory context, a dashboard offered by a PAIS could be used by a Chief Compliance Officer to set the weighting of the security- and resilience-oriented parameters by two sliders. In this way, further objectives assigned as a separate cost dimension, for example, regarding the performance of the process (KPI), could be added. The unraveling of presumably opposing indicators so far settled in a single cost dimension would be enabled through a more fine-grained approach that better reflects reality.

6.2.2 Beyond the Case: Inter-Instance- and Inter-Process-Related Obstructions

By expanding the focus from single process executions to the overall interrelations between processes orchestrated and steered by a PAIS, new problems regarding obstructive situations could be identified that accompany new possibilities for solutions beyond the individual case perspective. The SecANet encoding must then be extended to represent entire process architectures, e.g., with constraints between processes or their instances and users participating in different processes. Obstructions could result from the dependencies within the execution of several processes and their interplay with the overall policy. Besides control-flow dependencies between processes, this could affect the data flow, e.g., when a process is waiting for a processing file to complete because another obstructed process is idly accessing the file. Such more complex security-aware workflow specifications could also be related to satisfiability and resilience aspects. For example, requesting a vacation in such a system would reveal the impact of a user's absence from a process and provide a realistic overall view to determine the possibility of increasing obstructive risk or changing levels of resilience.

For resolving obstructed processes, this approach also makes it possible to consider the affected case along with its relation to other ongoing process executions, such as the users involved. An indicator could then correspond to the importance of executing specific (core) processes against the background of all processes running in a PAIS.

6.2.3 Beyond Predictions: Corrective Monitoring upon Occurring Obstructions

Due to the comparable problem setting, the developed methods can complement predictive monitoring by considering obstruction-related metrics as outcome-oriented predictions. Considering an obstructability metric during execution is comparable to an obstruction-free enforcement mechanism. For example, depending on a certain threshold of the obstructability metric, an attempt to assign a user to a task may or may not be permitted. Correspondingly, a completability metric that also captures the similarity of the execution to obstructed sequences, or a trend metric indicating if a process execution is tending towards an obstruction or completion, could refine predictions.

The predictive monitoring within the process mining could be complemented through a further step that prevents or avoids process execution from “going wrong,” while also correcting an occurring obstruction at runtime as a variant of online process mining. Such corrective monitoring can resolve the obstructed process execution at runtime by steering it to completion on-the-fly.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Bibliography

- [1] W. M. P. van der Aalst. “Business process management as the “Killer App” for Petri nets”. In: *Software & Systems Modeling* 14.2 (May 2015), pp. 685–691. ISSN: 1619–1374. URL: <https://doi.org/10.1007/s10270-014-0424-2>.
- [2] W. M. P. van der Aalst. “Three Good Reasons for Using a Petri-Net-Based Workflow Management System”. In: *Information and Process Integration in Enterprises: Rethinking Documents*. Ed. by Toshiro Wakayama, Srikanth Kannapan, Chan Meng Khoong, Shamkant Navathe, and JoAnne Yates. Boston, MA: Springer US, 1998, pp. 161–182. ISBN: 978-1-4615-5499-8. URL: https://doi.org/10.1007/978-1-4615-5499-8_10.
- [3] Wil M. P. van der Aalst. *Process Mining – Data Science in Action, Second Edition*. Springer, 2016. ISBN: 978-3-662-49850-7. URL: <https://doi.org/10.1007/978-3-662-49851-4>.
- [4] Wil M. P. van der Aalst. “The Application of Petri Nets to Workflow Management”. In: *Journal of Circuits, Systems, and Computers* 8.1 (1998), pp. 21–66. URL: <http://dx.doi.org/10.1142/S0218126698000043>.
- [5] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreech, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaessos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. “Process Mining Manifesto”. In: *Business Process Management Workshops – BPM 2011 International Workshops, Clermont-Ferrand, France, August*

- 29, 2011, *Revised Selected Papers, Part I*. Ed. by Florian Daniel, Kamel Barkaoui, and Schahram Dustdar. Vol. 99. Lecture Notes in Business Information Processing. Springer, 2011, pp. 169–194. ISBN: 978-3-642-28107-5. URL: https://doi.org/10.1007/978-3-642-28108-2_19.
- [6] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. “Workflow Patterns”. In: *Distributed and Parallel Databases* 14.1 (2003), pp. 5–51. URL: <https://doi.org/10.1023/A:1022883727209>.
- [7] Wil MP van der Aalst. “Structural characterizations of sound workflow nets”. In: *Computing Science Reports* 96.23 (1996), pp. 18–22.
- [8] Rafael Accorsi, Jason Crampton, Michael Huth, and Stefanie Rinderle-Ma. “Verifiably Secure Process-Aware Information Systems (Dagstuhl Seminar 13341)”. In: *Dagstuhl Reports* 3.8 (2013), pp. 73–86. URL: <https://doi.org/10.4230/DagRep.3.8.73>.
- [9] Rafael Accorsi, Julius Holderer, Thomas Stocker, and Richard M. Zahoransky. “Security Workflow Analysis Toolkit”. In: *Sicherheit 2014: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 19.-21. März 2014, Wien, Österreich*. Ed. by Stefan Katzenbeisser, Volkmar Lotz, and Edgar R. Weippl. Vol. P-228. LNI. GI, 2014, pp. 433–442. ISBN: 978-3-88579-622-0. URL: <https://dl.gi.de/20.500.12116/20062>.
- [10] Rafael Accorsi and Andreas Lehmann. “Automatic Information Flow Analysis of Business Process Models”. In: *Business Process Management – 10th International Conference, BPM 2012, Tallinn, Estonia, September 3–6, 2012. Proceedings*. Ed. by Alistair Barros, Avigdor Gal, and Ekkart Kindler. Vol. 7481. Lecture Notes in Computer Science. Springer, 2012, pp. 172–187. URL: https://doi.org/10.1007/978-3-642-32885-5%5C_13.
- [11] Rafael Accorsi and Claus Wonnemann. “Strong non-leak guarantees for workflow models”. In: *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 – 24, 2011*. Ed. by William C. Chu, W. Eric Wong, Mathew J. Palakal, and Chih-Cheng Hung. ACM, 2011, pp. 308–314. URL: <https://doi.org/10.1145/1982185.1982254>.
- [12] Nabil R. Adam, Vijayalakshmi Atluri, and Wei-kuang Huang. “Modeling and Analysis of Workflows Using Petri Nets”. In: *J. Intell. Inf. Syst.* 10.2 (1998), pp. 131–158. URL: <https://doi.org/10.1023/A:1008656726700>.
- [13] Bowen Alpern and Fred B. Schneider. “Defining Liveness”. In: *Inf. Process. Lett.* 21.4 (1985), pp. 181–185. URL: [https://doi.org/10.1016/0020-0190\(85\)90056-0](https://doi.org/10.1016/0020-0190(85)90056-0).
- [14] Bowen Alpern and Fred B. Schneider. “Recognizing Safety and Liveness”. In: *Distributed Computing* 2.3 (1987), pp. 117–126. URL: <https://doi.org/10.1007/BF01782772>.
- [15] James P. Anderson. *Computer security technology planning study vols. I and III*. Tech. rep. ESD-TR-73-51. Hanscom AFB, MA, Fort Washington, Pennsylvania: HQ Electronic Systems Division, Oct. 1972.
- [16] Anon. “This Court is not sentencing Mr Drumm for causing the financial crisis” – *Judge’s remarks in full*. 2018. URL: <https://www.irishtimes.com/news/crime-andlaw/this-court-is-not-sentencing-mr-drumm-for-causing-the-financial-crisis-judge-sremarks-in-full-1.3537672> (visited on 11/23/2018).
- [17] Michael Arias, Jorge Munoz-Gama, and Marcos Sepúlveda. “Towards a Taxonomy of Human Resource Allocation Criteria”. In: *Business Process Management Workshops –*

- BPM 2017 International Workshops, Barcelona, Spain, September 10–11, 2017*, Revised Papers. Ed. by Ernest Teniente and Matthias Weidlich. Vol. 308. Lecture Notes in Business Information Processing. Springer, 2017, pp. 475–483. ISBN: 978-3-319-74029-4. URL: https://doi.org/10.1007/978-3-319-74030-0_37.
- [18] André Arnold and John Plaice. *Finite Transition Systems: Semantics of Communicating Systems*. GBR: Prentice Hall International (UK) Ltd., 1994. ISBN: 0130929905.
- [19] Vijayalakshmi Atluri and Wei-kuang Huang. “A Petri Net Based Safety Analysis of Workflow Authorization Models”. In: *Journal of Computer Security* 8.2/3 (2000), pp. 209–240. URL: <http://content.iospress.com/articles/journal-of-computer-security/jcs113>.
- [20] The institute of internal auditors. *Simplifying Segregation of Duties*. 2018. URL: <https://iaonline.theiia.org/simplifying-segregation-of-duties> (visited on 11/23/2018).
- [21] Hanan El Bakkali. “Enhancing Workflow Systems Resiliency by Using Delegation and Priority Concepts.” In: *Journal of Digital Information Management* 11.4 (2013), pp. 267–276. ISSN: 09727272.
- [22] Kamila Barylska, Eike Best, Uli Schlachter, and Valentin Spreckels. “Properties of Plain, Pure, and Safe Petri Nets”. In: *Trans. Petri Nets Other Model. Concurr. Lecture Notes in Computer Science* 12 (2017). Ed. by Maciej Koutny, Jetty Kleijn, and Wojciech Penczek, pp. 1–18. URL: https://doi.org/10.1007/978-3-662-55862-1_1.
- [23] David A. Basin, Samuel J. Burri, and Günter Karjoth. “Dynamic enforcement of abstract separation of duty constraints”. In: *ACM Trans. Inf. Syst. Secur.* 15.3 (2012), p. 13.
- [24] David A. Basin, Samuel J. Burri, and Günter Karjoth. “Obstruction-Free Authorization Enforcement: Aligning Security with Business Objectives”. In: *CSF*. IEEE Computer Society, 2011, pp. 99–113. ISBN: 978-1-61284-644-6.
- [25] David A. Basin, Samuel J. Burri, and Günter Karjoth. “Optimal workflow-aware authorizations”. In: *SACMAT*. Ed. by Vijay Atluri, Jaideep Vaidya, Axel Kern, and Murat Kantarcioglu. ACM, 2012, pp. 93–102. ISBN: 978-1-4503-1295-0.
- [26] David A. Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zalinescu. “Enforceable Security Policies Revisited”. In: *ACM Trans. Inf. Syst. Secur.* 16.1 (2013), p. 3.
- [27] Gustavo E. A. P. A. Batista and Maria Carolina Monard. “An Analysis of Four Missing Data Treatment Methods for Supervised Learning”. In: *Appl. Artif. Intell.* 17.5–6 (2003), pp. 519–533. URL: <https://doi.org/10.1080/713827181>.
- [28] David Elliott Bell. “Bell-La Padula Model”. In: *Encyclopedia of Cryptography and Security (2nd Ed.)*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Springer, 2011, pp. 74–79. ISBN: 978-1-4419-5905-8.
- [29] Daniel Le Berre and Stéphanie Roussel. “Sat4j 2.3.2: on the fly solver configuration System Description”. In: *J. Satisf. Boolean Model. Comput.* 8.3/4 (2014), pp. 197–202. URL: <https://doi.org/10.3233/sat190098>.
- [30] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. “The Specification and Enforcement of Authorization Constraints in Workflow Management Systems”. In: *ACM Trans. Inf. Syst. Secur.* 2.1 (1999), pp. 65–104.
- [31] Clara Bertolissi, Daniel Ricardo dos Santos, and Silvio Ranise. “Automated Synthesis of Run-time Monitors to Enforce Authorization Policies in Business Processes”. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore*, April 14–17, 2015. Ed. by Feng Bao,

- Steven Miller, Jianying Zhou, and Gail-Joon Ahn. ACM, 2015, pp. 297–308. ISBN: 978-1-4503-3245-3. URL: <https://doi.org/10.1145/2714576.2714633>.
- [32] Eike Best. “Structure theory of Petri nets: the free choice hiatus”. In: *Advanced Course on Petri Nets*. Springer, 1986, pp. 168–205.
- [33] Matt Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, 2004. ISBN: 0321247442.
- [34] Matt Bishop, Heather M. Conboy, Huong Phan, Borislava I. Simidchieva, George S. Avrunin, Lori A. Clarke, Leon J. Osterweil, and Sean Peisert. “Insider Threat Identification by Process Analysis”. In: *35. IEEE Security and Privacy Workshops, SPW 2014, San Jose, CA, USA, May 17–18, 2014. IEEE Computer Society, 2014*, pp. 251–264. ISBN: 978-1-4799-5103-1. URL: <https://doi.org/10.1109/SPW.2014.40>.
- [35] Reinhardt Botha and Jan Eloff. “Separation of Duties for Access Control Enforcement in Workflow Environments”. In: *IBM Systems Journal* 40.3 (Mar. 2001), pp. 666–682.
- [36] Achim D. Brucker and Isabelle Hang. “Secure and Compliant Implementation of Business Process-Driven Systems”. In: *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*. Ed. by Marcello La Rosa and Pnina Soffer. Vol. 132. *Lecture Notes in Business Information Processing*. Springer, 2012, pp. 662–674. ISBN: 978-3-642-36284-2. URL: https://doi.org/10.1007/978-3-642-36285-9_66.
- [37] Achim D. Brucker and Helmut Petritsch. “Extending access control models with break-glass”. In: *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Proceedings*. Ed. by Barbara Carminati and James Joshi. Stresa, Italy: ACM, June 2009, pp. 197–206. ISBN: 978-1-60558-537-6.
- [38] Julien Brunel, Frédéric Cuppens, Nora Cuppens-Boulahia, Thierry Sans, and Jean-Paul Bodeveix. “Security policy compliance with violation management”. In: *Proceedings of the 2007 ACM workshop on Formal methods in security engineering, FMSE 2007, Fairfax, VA, USA, November 2, 2007*. Ed. by Peng Ning, Vijay Atluri, Virgil D. Gligor, and Heiko Mantel. ACM, 2007, pp. 31–40. ISBN: 978-1-59593-887-9. URL: <https://doi.org/10.1145/1314436.1314441>.
- [39] Samuel J. Burri. “Modeling and enforcing workflow authorizations”. PhD thesis. Zürich: ETH, 2012.
- [40] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking – Relating Processes and Models*. Springer, 2018. ISBN: 978-3-319-99413-0. URL: <https://doi.org/10.1007/978-3-319-99414-7>.
- [41] Association of Certified Fraud Examiners. *Report to the Nations – Global Study on Occupational Fraud and Abuse*. 2018. URL: <https://s3-us-west-2.amazonaws.com/acfe-public/2018-report-to-the-nations.pdf> (visited on 11/23/2018).
- [42] Anis Charfi and Mira Mezini. “Aspect-oriented workflow languages”. In: *OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”*. Springer, 2006, pp. 183–200.
- [43] Liang Chen and Jason Crampton. “Risk-Aware Role-Based Access Control”. In: *Security and Trust Management – 7th International Workshop, STM 2011, Copenhagen, Denmark, June 27–28, 2011, Revised Selected Papers*. Ed. by Catherine Meadows and M. Carmen Fernández Gago. Vol. 7170. *Lecture Notes in Computer Science*. Springer, 2011, pp. 140–156. ISBN: 978-3-642-29962-9. URL: https://doi.org/10.1007/978-3-642-29963-6_11.

- [44] Pau-Chen Cheng, Pankaj Rohatgi, Claudia Keser, Paul A. Karger, Grant M. Wagner, and Angela Schuett Reninger. “Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control”. In: *2007 IEEE Symposium on Security and Privacy (S&P 2007)*, 20–23 May 2007, Oakland, California, USA. IEEE Computer Society, 2007, pp. 222–230. ISBN: 0-7695-2848-1. URL: <https://doi.org/10.1109/SP.2007.21>.
- [45] Feng Chu and Xiaolan Xie. “Deadlock analysis of Petri nets using siphons and mathematical programming”. In: *IEEE Trans. Robotics and Automation* 13.6 (1997), pp. 793–804. URL: <https://doi.org/10.1109/70.650158>.
- [46] Michael R. Clarkson and Fred B. Schneider. “Hyperproperties”. In: *Journal of Computer Security* 18.6 (2010), pp. 1157–1210. URL: <https://doi.org/10.3233/JCS-2009-0393>.
- [47] David A. Cohen, Jason Crampton, Andrei Gagarin, Gregory Z. Gutin, and Mark Jones. “Algorithms for the workflow satisfiability problem engineered for counting constraints”. In: *J. Comb. Optim.* 32.1 (2016), pp. 3–24. URL: <https://doi.org/10.1007/s10878-015-9877-7>.
- [48] David A. Cohen, Jason Crampton, Andrei Gagarin, Gregory Z. Gutin, and Mark Jones. “Engineering Algorithms for Workflow Satisfiability Problem with User-Independent Constraints”. In: *Frontiers in Algorithmics – 8th International Workshop, FAW 2014*, Zhangjiajie, China, June 28–30, 2014. Proceedings. Ed. by Jianer Chen, John E. Hopcroft, and Jianxin Wang. Vol. 8497. Lecture Notes in Computer Science. Springer, 2014, pp. 48–59. ISBN: 978-3-319-08015-4. URL: https://doi.org/10.1007/978-3-319-08016-1_5.
- [49] David A. Cohen, Jason Crampton, Andrei Gagarin, Gregory Z. Gutin, and Mark Jones. “Iterative Plan Construction for the Workflow Satisfiability Problem”. In: *J. Artif. Intell. Res.* 51 (2014), pp. 555–577. URL: <https://doi.org/10.1613/jair.4435>.
- [50] David A. Cohen, Jason Crampton, Gregory Gutin, and Magnus Wahlström. “Parameterized Complexity of the Workflow Satisfiability Problem”. In: *Combinatorial Optimization and Graph Algorithms. Springer, 2017*, pp. 101–120.
- [51] Frederic G Commoner. *Deadlocks in Petri-nets*. Massachusetts Computer Assoc., Incorporated, 1972.
- [52] Luca Compagna, Daniel Ricardo dos Santos, Serena Elisa Ponta, and Silvio Ranise. “Aegis: Automatic Enforcement of Security Policies in Workflow-driven Web Applications”. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22–24, 2017*. Ed. by Gail-Joon Ahn, Alexander Pretschner, and Gabriel Ghinita. ACM, 2017, pp. 321–328. ISBN: 978-1-4503-4523-1. URL: <https://doi.org/10.1145/3029806.3029813>.
- [53] Raffaele Conforti, Marcello La Rosa, and Arthur H. M. ter Hofstede. “Filtering Out Infrequent Behavior from Business Process Event Logs”. In: *IEEE Trans. Knowl. Data Eng.* 29.2 (2017), pp. 300–314. URL: <https://doi.org/10.1109/TKDE.2016.2614680>.
- [54] Ruadhan Mac Cormaic. *Anglo trial: Three ex-bankers jailed over 7bn Euro fraud*. 2018. URL: <https://www.irishtimes.com/business/financial-services/anglo-trial-three-ex-bankers-jailed-over-7bn-fraud-1.2738637> (visited on 11/23/2018).
- [55] Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. “Synthesizing Petri Nets from State-Based models”. In: *ICCAD*. Ed. by Richard L. Rudell. IEEE Computer Society, 1995, pp. 164–171. ISBN: 0-8186-7213-7.

- [56] Thomas Cover and Peter Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967), pp. 21–27. ISSN: 0018-9448.
- [57] J. Crampton and M. Huth. “Synthesizing and Verifying Plans for Constrained Workflows: Transferring Tools from Formal Methods”. In: *Proceedings of the 2011 Workshop on Verification and Validation of Planning and Scheduling Systems*. To appear. 2011.
- [58] Jason Crampton. “A reference monitor for workflow systems with constrained task execution”. In: *SACMAT*. Ed. by Elena Ferrari and Gail-Joon Ahn. ACM, 2005, pp. 38–47. ISBN: 1-59593-045-0.
- [59] Jason Crampton, Robert Crowston, Gregory Gutin, Mark Jones, and M. S. Ramanujan. “Fixed-Parameter Tractability of Workflow Satisfiability in the Presence of Seniority Constraints”. In: *FAW-AAIM*. Ed. by Michael R. Fellows, Xuehou Tan, and Binhai Zhu. Vol. 7924. Lecture Notes in Computer Science. Springer, 2013, pp. 198–209. ISBN: 978-3-642-38756-2.
- [60] Jason Crampton, Andrei Gagarin, Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. “On the Workflow Satisfiability Problem with Class-Independent Constraints for Hierarchical Organizations”. In: *ACM Trans. Priv. Secur.* 19.3 (2016), 8:1–8:29. URL: <https://doi.org/10.1145/2988239>.
- [61] Jason Crampton and Gregory Gutin. “Constraint expressions and workflow satisfiability”. In: *SACMAT*. Ed. by Mauro Conti, Jaideep Vaidya, and Andreas Schaad. ACM, 2013, pp. 73–84. ISBN: 978-1-4503-1950-8.
- [62] Jason Crampton, Gregory Z. Gutin, and Daniel Karapetyan. “Valued Workflow Satisfiability Problem”. In: *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1–3, 2015*. Ed. by Edgar R. Weippl, Florian Kerschbaum, and Adam J. Lee. ACM, 2015, pp. 3–13. ISBN: 978-1-4503-3556-0. URL: <https://doi.org/10.1145/2752952.2752961>.
- [63] Jason Crampton, Gregory Z. Gutin, Daniel Karapetyan, and Rémi Watrigant. “The bi-objective workflow satisfiability problem and workflow resiliency”. In: *Journal of Computer Security* 25.1 (2017), pp. 83–115. URL: <https://doi.org/10.3233/JCS-16849>.
- [64] Jason Crampton, Gregory Gutin, and Anders Yeo. “On the Parameterized Complexity and Kernelization of the Workflow Satisfiability Problem”. In: *ACM Trans. Inf. Syst. Secur.* 16.1 (2013), p. 4.
- [65] Jason Crampton, Gregory Gutin, and Anders Yeo. “On the parameterized complexity of the workflow satisfiability problem”. In: *ACM Conference on Computer and Communications Security*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM, 2012, pp. 857–868. ISBN: 978-1-4503-1651-4.
- [66] Jason Crampton and Michael Huth. “On the Modeling and Verification of Security-Aware and Process-Aware Information Systems”. In: *Business Process Management Workshops (2)*. Ed. by Florian Daniel, Kamel Barkaoui, and Schahram Dustdar. Vol. 100. Lecture Notes in Business Information Processing. Springer, 2011, pp. 423–434. ISBN: 978-3-642-28114-3.
- [67] Jason Crampton, Michael Huth, and Jim Huan-Pu Kuo. “Authorization Enforcement in Workflows: Maintaining Realizability Via Automated Reasoning”. In: *PAAR@IJCAR*. Ed. by Pascal Fontaine, Renate A. Schmidt, and Stephan Schulz. Vol. 21. EPiC Series. EasyChair, 2012, pp. 29–42.

- [68] Jason Crampton, Michael Huth, and Jim Huan-Pu Kuo. “Authorized workflow schemas: deciding realizability through LTL (F) model checking”. In: *STTT* 16.1 (2014), pp. 31–48.
- [69] Jason Crampton and Hemanth Khambhammettu. “Delegation and satisfiability in workflow systems”. In: *SACMAT*. Ed. by Indrakshi Ray and Ninghui Li. ACM, 2008, pp. 31–40. ISBN: 978-1-60558-129-3.
- [70] Jason Crampton and Charles Morisset. “An Auto-delegation Mechanism for Access Control Systems”. In: *STM*. Ed. by Jorge Cuéllar, Javier Lopez, Gilles Barthe, and Alexander Pretschner. Vol. 6710. Lecture Notes in Computer Science. Springer, 2010, pp. 1–16. ISBN: 978-3-642-22443-0.
- [71] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. ISBN: 978-3-319-21274-6. URL: <https://doi.org/10.1007/978-3-319-21275-3>.
- [72] Gordon Deegan. *David Drumm prosecution team paid 1.1m Euro. 2018*. URL: <https://www.irishtimes.com/news/crime-and-law/david-drumm-prosecution-team-paid-1-1m-1.3599801?mode=sample&auth-failed=1&pw-orig=https%3A%2F%2Fwww.irishtimes.com%2Fnews%2FCrime-and-law%2Fdavid-drumm-prosecution-team-paid-1-1m-1.3599801> (visited on 11/23/2018).
- [73] J. Desel and J. Esparza. *Free choice Petri nets*. Vol. 40. Cambridge Univ Pr, 1995.
- [74] Jörg Desel. “On Cyclic Behaviour of Unbounded Petri Nets”. In: *13th International Conference on Application of Concurrency to System Design, ACSD 2013*, Barcelona, Spain, 8–10 July, 2013. Ed. by Josep Carmona, Mihai T. Lazarescu, and Marta Pietkiewicz-Koutny. IEEE Computer Society, 2013, pp. 110–119. ISBN: 978-0-7695-5035-0. URL: <https://doi.org/10.1109/ACSD.2013.14>.
- [75] Jörg Desel and Javier Esparza. “Reachability in Cyclic Extended Free-Choice Systems”. In: *Theor. Comput. Sci.* 114.1 (1993), pp. 93–118. URL: [https://doi.org/10.1016/0304-3975\(93\)90154-L](https://doi.org/10.1016/0304-3975(93)90154-L).
- [76] Jörg Desel and Javier Esparza. “Reachability in cyclic extended free-choice systems.” In: *TCS 114, Elsevier Science Publishers B.V.* (1993).
- [77] AA Desrochers and RY Al-Jaar. “Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis, the Institute of Electrical and Electronics Engineers”. In: *Inc., New York* (1995).
- [78] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. “Semantics and Analysis of Business Process Models in BPMN”. In: *Inf. Softw. Technol.* 50.12 (Nov. 2008), pp. 1281–1294. ISSN: 0950-5849. URL: <http://dx.doi.org/10.1016/j.infsof.2008.02.006>.
- [79] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. “Semantics and analysis of business process models in BPMN”. In: *Information & Software Technology* 50.12 (2008), pp. 1281–1294. URL: <https://doi.org/10.1016/j.infsof.2008.02.006>.
- [80] Nathan Dimmock, András Belokosztolszki, David M. Eyers, Jean Bacon, and Ken Moody. “Using trust and risk in role-based access control policies”. In: *9th ACM Symposium on Access Control Models and Technologies, SACMAT 2004, Yorktown Heights, New York, USA, June 2–4, 2004, Proceedings*. Ed. by Trent Jaeger and Elena Ferrari. ACM, 2004, pp. 156–162. ISBN: 1-58113-872-5. URL: <https://doi.org/10.1145/990036.990062>.

- [81] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013, pp. I–XXVII, 1–399. ISBN: 978-3-642-33142-8.
- [82] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005.
- [83] The Economist. *Rise of the No Men: The past decade has brought a compliance boom in banking*. 2019. URL: <https://www.economist.com/finance-and-economics/2019/05/02/the-past-decade-has-brought-a-compliance-boom-in-banking> (visited on 05/05/2019).
- [84] André Elisseeff, Theodoros Evgeniou, and Massimiliano Pontil. “Stability of Randomized Learning Algorithms”. In: *J. Mach. Learn. Res.* 6 (2005), pp. 55–79. URL: <http://jmlr.org/papers/v6/elisseeff05a.html>.
- [85] Charles Elkan. “The Foundations of Cost-Sensitive Learning”. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4–10, 2001*. Ed. by Bernhard Nebel. Morgan Kaufmann, 2001, pp. 973–978. ISBN: 1-55860-777-3. URL: <http://ijcai.org/proceedings/2001-1>.
- [86] Clarence A. Ellis and Gary J. Nutt. “Modeling and Enactment of Workflow Systems”. In: *Application and Theory of Petri Nets*. Ed. by Marco Ajmone Marsan. Vol. 691. Lecture Notes in Computer Science. Springer, 1993, pp. 1–16. ISBN: 3-540-56863-8.
- [87] Javier Esparza. “Decidability and Complexity of Petri Net Problems - An Introduction”. In: *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*. Ed. by Wolfgang Reisig and Grzegorz Rozenberg. Vol. 1491. *Lecture Notes in Computer Science*. Springer, 1996, pp. 374–428. ISBN: 3-540-65306-6. URL: https://doi.org/10.1007/3-540-65306-6_20.
- [88] Javier Esparza. “Reachability in Live and Safe Free-Choice Petri Nets is NPComplete”. In: *Theor. Comput. Sci.* 198.1–2 (1998), pp. 211–224. URL: [https://doi.org/10.1016/S0304-3975\(97\)00235-1](https://doi.org/10.1016/S0304-3975(97)00235-1).
- [89] Javier Esparza and Stephan Melzer. “Verification of safety properties using integer programming: Beyond the state equation”. In: *Formal Methods in System Design* 16 (2000), pp. 159–189.
- [90] Javier Esparza and Manuel Silva. “Circuits, handles, bridges and nets”. In: *Advances in Petri Nets 1990*. Ed. by Grzegorz Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 210–242. ISBN: 978-3-540-46369-6.
- [91] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. “Predicting process behaviour using deep learning”. In: *Decision Support Systems 100 (2017)*, pp. 129–140. URL: <https://doi.org/10.1016/j.dss.2017.04.003>.
- [92] FCW. *White Paper: Sustainable Compliance: How to Align Compliance, Security and Business Goals*. 2013. URL: <https://fcw.com/whitepapers/2013/03/netiqsustainable-compliance-031413.aspx> (visited on 11/23/2018).
- [93] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. “Proposed NIST standard for role-based access control”. In: *ACM Trans. Inf. Syst. Secur.* 4.3 (2001), pp. 224–274. URL: <https://doi.org/10.1145/501978.501980>.

- [94] Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin). *Mindestanforderungen an das Risikomanagement – MaRisk – BTO Anforderungen an die Aufbau- und Ablauforganisation*. 2018. URL: https://www.bafin.de/SharedDocs/Veroeffentlichungen/DE/Rundschreiben/2017/rs_1709_marisk_ba.html (visited on 11/23/2018).
- [95] Hartmann J. Genrich and Kurt Lautenbach. “System Modelling with High-Level Petri Nets”. In: *Theor. Comput. Sci.* 13 (1981), pp. 109–136. URL: [https://doi.org/10.1016/0304-3975\(81\)90113-4](https://doi.org/10.1016/0304-3975(81)90113-4).
- [96] Martin Gill and Geoff Taylor. “Preventing money laundering or obstructing business? Financial companies’ perspectives on ‘know your customer’ procedures”. In: *British Journal of Criminology* 44.4 (2004), pp. 582–594.
- [97] Joseph A. Goguen and José Meseguer. “Security Policies and Security Models”. In: *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26–28, 1982*. IEEE Computer Society, 1982, pp. 11–20. ISBN: 0-8186-0410-7. URL: <https://doi.org/10.1109/SP.1982.10014>.
- [98] Dieter Gollmann. “Computer security”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.5 (July 2010), pp. 544–554. ISSN: 1939–5108. URL: <http://dx.doi.org/10.1002/wics.106>.
- [99] Jan Friso Groote and Marc Voorhoeve. “Operational semantics for Petri net components”. In: *Theor. Comput. Sci.* 379.1–2 (2007), pp. 1–19. URL: <https://doi.org/10.1016/j.tcs.2007.01.003>.
- [100] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2016. URL: <http://www.gurobi.com>.
- [101] M. Hack. *Analysis of production schemata by Petri nets*. Tech. rep. USA, 1972.
- [102] Peter Hamilton. *Business 2018: the winners and the losers*. 2018. URL: <https://www.irishtimes.com/business/work/business-2018-the-winners-and-the-losers-1.3738179?mode=sample&auth-failed=1&pw-origin=https%3A%2F%2Fwww.irishtimes.com%2Fbusiness%2Fwork%2Fbusiness-2018-the-winners-and-the-losers-1.3738179> (visited on 11/23/2018).
- [103] Kevin W. Hamlen, J. Gregory Morrisett, and Fred B. Schneider. “Computability classes for enforcement mechanisms”. In: *ACM Trans. Program. Lang. Syst.* 28.1 (2006), pp. 175–205. URL: <http://doi.acm.org/10.1145/1111596.1111601>.
- [104] Paul Harmon and Jorge Garcia. *The State of Business Process Management 2020*. BPTrends Report. Available at <http://www.bptrends.com/2020>. (Visited on 11/03/2020).
- [105] Arthur H.M. ter Hofstede, Wil M.P. van der Aalst, Michael Adams, and Nick Russell, eds. *Modern Business Process Automation – YAWL and its Support Environment*. Springer, 2010. ISBN: 978-3-642-03120-5.
- [106] Julius Holderer. “eLog: Complex Event Log Synthesis Tool”. In: *Tagungsband Multi-konferenz Wirtschaftsinformatik 2014 (MKWI 2014)*. Ed. by Dennis Kundisch, Leena Suhl, and Lars Beckmann. Universität Paderborn, 2014, pp. 1937–1950. ISBN: 978-3-00-045311-3.
- [107] Julius Holderer, Rafael Accorsi, and Günter Müller. “When four-eyes become too much: a survey on the interplay of authorization constraints and workflow resilience”. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13–17, 2015*. Ed. by Roger L. Wainwright, Juan Manuel Corchado,

- Alessio Bechini, and Jiman Hong. ACM, 2015, pp. 1245–1248. ISBN: 978-1-4503-3196-8. URL: <http://doi.acm.org/10.1145/2695664.2699497>.
- [108] Julius Holderer, Josep Carmona, and Günter Müller. “Security-Sensitive Tackling of Obstructed Workflow Executions”. In: *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016 Satellite event of the conference: 37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and 16th International Conference on Application of Concurrency to System Design ACSD 2016, Torun, Poland, June 20–21, 2016*. Ed. by Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona. Vol. 1592. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 126–137. URL: <http://ceur-ws.org/Vol-1592/paper09.pdf>.
- [109] Julius Holderer, Josep Carmona, Farbod Taymouri, and Günter Müller. “Log- and Model-Based Techniques for Security-Sensitive Tackling of Obstructed Workflow Executions”. In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Ed. by Maciej Koutny, Jetty Kleijn, and Wojciech Penczek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 43–69. ISBN: 978-3-662-55862-1. URL: https://doi.org/10.1007/978-3-662-55862-1_3.
- [110] John E. Hopcroft and Robert Endre Tarjan. “Dividing a Graph into Triconnected Components”. In: *SIAM J. Comput.* 2.3 (1973), pp. 135–158. URL: <http://dx.doi.org/10.1137/0202012>.
- [111] Hejiao Huang and H el ene Kirchner. “Component-Based Security Policy Design with Colored Petri Nets”. In: *Semantics and Algebraic Specification*. Ed. by Jens Palsberg. Vol. 5700. Lecture Notes in Computer Science. Springer, 2009, pp. 21–42. ISBN: 978-3-642-04163-1.
- [112] Hejiao Huang and H el ene Kirchner. “Formal Specification and Verification of Modular Security Policy Based on Colored Petri Nets”. In: *IEEE Transactions on Dependable and Secure Computing* 8.6 (2011), pp. 852–865.
- [113] Hejiao Huang and H el ene Kirchner. “Secure Interoperation Design in Multi-Domains Environments Based on Colored Petri Nets”. In: *Information Sciences* 221 (2013), pp. 591–606.
- [114] Oscar H. Ibarra, Shlomo Moran, and Roger Hui. “A Generalization of the Fast LUP Matrix Decomposition Algorithm and Applications”. In: *J. Algorithms* 3.1 (1982), pp. 45–56. URL: [https://doi.org/10.1016/0196-6774\(82\)90007-4](https://doi.org/10.1016/0196-6774(82)90007-4).
- [115] “IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams”. In: *IEEE Std 1849-2016* (Nov. 2016), pp. 1–50.
- [116] ISO/IEC 19505-1. *Information technology - Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure*. Geneva, Switzerland: International Organization for Standardization, 2012.
- [117] ISO/IEC 19505-2. *Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 2: Superstructure*. Geneva, Switzerland: International Organization for Standardization, 2012.
- [118] ISO/IEC 19510. *Information technology – Object Management Group Business Process Model and Notation*. Geneva, Switzerland: International Organization for Standardization, 2013.

- [119] Stefan Jablonski and Christoph Bussler. *Workflow Management - Modeling Concepts, Architecture and Implementation*. International Thomson, 1996, pp. I–X, 1–351. ISBN: 978-1-85032-222-1.
- [120] Kurt Jensen. “Coloured Petri Nets and the Invariant-Method”. In: *Theor. Comput. Sci.* 14 (1981), pp. 317–336. URL: [https://doi.org/10.1016/0304-3975\(81\)90049-9](https://doi.org/10.1016/0304-3975(81)90049-9).
- [121] Yixin Jiang, Chuang Lin, Hao Yin, and Zhangxi Tan. “Security Analysis of Mandatory Access Control Model”. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 6. 2004, pp. 5013–5018.
- [122] Li Jiao, To-Yat Cheung, and Weiming Lu. “On liveness and boundedness of asymmetric choice nets”. In: *Theor. Comput. Sci.* 311.1–3 (2004), pp. 165–197. URL: [https://doi.org/10.1016/S0304-3975\(03\)00359-1](https://doi.org/10.1016/S0304-3975(03)00359-1).
- [123] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, et al. “Web services business process execution language version 2.0”. In: *OASIS standard* 11.120 (2007), p. 5.
- [124] Anna A. Kalenkova, Wil M. P. van der Aalst, Irina A. Lomazova, and Vladimir A. Rubin. “Process mining using BPMN: relating event logs and process models”. In: *Software and Systems Modeling* 16.4 (2017), pp. 1019–1048. URL: <https://doi.org/10.1007/s10270-015-0502-0>.
- [125] Daniel Karapetyan, Andrew J. Parkes, Gregory Z. Gutin, and Andrei Gagarin. “Pattern-Based Approach to the Workflow Satisfiability Problem with User-Independent Constraints”. In: *J. Artif. Intell. Res.* 66 (2019), pp. 85–122. URL: <https://doi.org/10.1613/jair.1.11339>.
- [126] Joost-Pieter Katoen. “Causal Behaviours and Nets”. In: *Application and Theory of Petri Nets 1995, 16th International Conference, Turin, Italy, June 26–30, 1995, Proceedings*. Ed. by Giorgio De Michelis and Michel Diaz. Vol. 935. Lecture Notes in Computer Science. Springer, 1995, pp. 258–277. ISBN: 3-540-60029-9. URL: https://doi.org/10.1007/3-540-60029-9_44.
- [127] Basel Katt, Michael Hafner, and Xinwen Zhang. “A Usage Control Policy Specification with Petri Nets”. In: *CollaborateCom*. IEEE, 2009, pp. 1–8. ISBN: 978-963-9799-76-9.
- [128] Arif Akram Khan and Philip W. L. Fong. “Satisfiability and Feasibility in a Relationship-Based Workflow Authorization Model”. In: *ESORICS*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. Lecture Notes in Computer Science. Springer, 2012, pp. 109–126. ISBN: 978-3-642-33166-4.
- [129] Kenji Kira and Larry A. Rendell. “A Practical Approach to Feature Selection”. In: *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1–3, 1992*. Ed. by Derek H. Sleeman and Peter Edwards. Morgan Kaufmann, 1992, pp. 249–256. ISBN: 1-55860-247-X. URL: <https://doi.org/10.1016/b978-1-55860-247-2.50037-1>.
- [130] Katharina Kneisel. “Postakquisitionsaudits mit Compliance Fokus: “Red Flags” für die Interne Revision”. In: *Zeitschrift Interne Revision* 6 (2016), pp. 278–285.
- [131] Maria Krambia-Kapardis. “Financial Crisis, Fraud, and Corruption”. In: *Corporate Fraud and Corruption: A Holistic Approach to Preventing Financial Crises*. New York: Palgrave Macmillan US, 2016, pp. 5–38. ISBN: 978-1-137-40643-9. URL: https://doi.org/10.1057/9781137406439_2.

- [132] Leslie Lamport. “Proving the Correctness of Multiprocess Programs”. In: *IEEE Trans. Software Eng.* 3.2 (1977), pp. 125–143. URL: <https://doi.org/10.1109/TSE.1977.229904>.
- [133] Maria Leitner, Zhendong Ma, and Stefanie Rinderle-Ma. “A Cross-Layer Security Analysis for Process-Aware Information Systems”. In: *CoRR abs/1507.03415* (2015). arXiv:1507.03415. URL: <http://arxiv.org/abs/1507.03415>.
- [134] Maria Leitner and Stefanie Rinderle-Ma. “A systematic review on security in Process-Aware Information Systems – Constitution, challenges, and future directions”. In: *Information and Software Technology* 56.3 (2014), pp. 273–293.
- [135] Ninghui Li, Mahesh V. Tripunitara, and Qihua Wang. “Resiliency policies in access control”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, Alexandria, VA, USA, Ioctober 30 – November 3, 2006. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM, 2006, pp. 113–123. URL: <https://doi.org/10.1145/1180405.1180421>.
- [136] GuanJun Liu and Changjun Jiang. “Co-NP-Hardness of the Soundness Problem for Asymmetric-Choice Workflow Nets”. In: *IEEE Trans. Systems, Man, and Cybernetics: Systems* 45.8 (2015), pp. 1201–1204. URL: <https://doi.org/10.1109/TSMC.2014.2386802>.
- [137] Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. “Language-Theoretic Abstraction Refinement”. In: *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*. Ed. by Juan de Lara and Andrea Zisman. Vol. 7212. Lecture Notes in Computer Science. Springer, 2012, pp. 362–376. ISBN: 978-3-642-28871-5. URL: https://doi.org/10.1007/978-3-642-28872-2_25.
- [138] Meghna Lowalekar, Ritesh Kumar Tiwari, and Kamalakar Karlapalem. “Security Policy Satisfiability and Failure Resilience in Workflows”. In: *FIDIS*. Ed. by Vashek Matyás, Simone Fischer-Hübner, Daniel Cvrcek, and Petr Svenda. Vol. 298. IFIP Advances in Information and Communication Technology. Springer, 2008, pp. 197–210. ISBN: 978-3-642-03314-8.
- [139] Yahui Lu, Li Zhang, and Jianguang Sun. “Using colored Petri nets to model and analyze workflow with separation of duty constraints”. In: *The International Journal of Advanced Manufacturing Technology* 40.1 (Jan. 2009), pp. 179–192. ISSN: 1433–3015. URL: <https://doi.org/10.1007/s00170-007-1316-1>.
- [140] Hongyan Ma. “Process-aware information systems: Bridging people and software through process technology”. In: *JASIST* 58.3 (2007), pp. 455–456. URL: <https://doi.org/10.1002/asi.20456>.
- [141] John C. Mace, Charles Morisset, and Aad P. A. van Moorsel. “Quantitative Workflow Resiliency”. In: *Computer Security – ESORICS 2014 – 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7–11, 2014. Proceedings, Part I*. Ed. by Mirosław Kutylowski and Jaideep Vaidya. Vol. 8712. Lecture Notes in Computer Science. Springer, 2014, pp. 344–361. ISBN: 978-3-319-11202-2. URL: https://doi.org/10.1007/978-3-319-11203-9_20.
- [142] John C. Mace, Charles Morisset, and Aad P. A. van Moorsel. “WRAD: Tool Support for Workflow Resiliency Analysis and Design”. In: *Software Engineering for Resilient Systems – 8th International Workshop, SERENE 2016, Gothenburg, Sweden, Septem-*

- ber 5–6, 2016, *Proceedings*. Ed. by Ivica Crnkovic and Elena Troubitsyna. Vol. 9823. Lecture Notes in Computer Science. Springer, 2016, pp. 79–87. ISBN: 978-3-319-45891-5. URL: https://doi.org/10.1007/978-3-319-45892-2_6.
- [143] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. “Predictive Monitoring of Business Processes”. In: *Advanced Information Systems Engineering – 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16–20, 2014, Proceedings*. Ed. by Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff. Vol. 8484. Lecture Notes in Computer Science. Springer, 2014, pp. 457–472. ISBN: 978-3-319-07880-9. URL: https://doi.org/10.1007/978-3-319-07881-6_31.
- [144] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. “Predictive Monitoring of Business Processes: A Survey”. In: *IEEE Trans. Services Computing* 11.6 (2018), pp. 962–977. URL: <https://doi.org/10.1109/TSC.2017.2772256>.
- [145] Fabio Massacci, Federica Paci, and Olga Gadyatskaya. “Dynamic resiliency to user assignment”. In: (2010).
- [146] Ernst W. Mayr. “An Algorithm for the General Petri Net Reachability Problem”. In: *SIAM J. Comput.* 13.3 (1984), pp. 441–460. URL: <https://doi.org/10.1137/0213029>.
- [147] Joe McGrath. *Corporate and White-collar Crime in Ireland: A New Architecture of Regulatory Enforcement*. Oxford University Press, 2015.
- [148] John McLean. “A general theory of composition for trace sets closed under selective interleaving functions”. In: *1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 16–18, 1994*. IEEE Computer Society, 1994, pp. 79–93. ISBN: 0-8186-5675-1. URL: <https://doi.org/10.1109/RISP.1994.296590>.
- [149] Andreas Metzger, Rod Franklin, and Yagil Engel. “Predictive Monitoring of Heterogeneous Service-Oriented Business Networks: The Transport and Logistics Case”. In: *2012 Annual SRII Global Conference, San Jose, CA, USA, July 24–27, 2012*. IEEE Computer Society, 2012, pp. 313–322. ISBN: 978-1-4673-2318-5. URL: <https://doi.org/10.1109/SRII.2012.42>.
- [150] Andreas Metzger, Philipp Leitner, Dragan Ivanovic, Eric Schmieders, Rod Franklin, Manuel Carro, Schahram Dustdar, and Klaus Pohl. “Comparing and Combining Predictive Business Process Monitoring Techniques”. In: *IEEE Trans. Systems, Man, and Cybernetics: Systems* 45.2 (2015), pp. 276–290. URL: <https://doi.org/10.1109/TSMC.2014.2347265>.
- [151] Michael K. Molloy. “Performance Analysis Using Stochastic Petri Nets”. In: *IEEE Trans. Computers* 31.9 (1982), pp. 913–917. URL: <https://doi.org/10.1109/TC.1982.1676110>.
- [152] Günter Müller. “Digitale Transformation: Digitalisierungsdilemma und Vertrauenskrise”. In: *Protektion 4.0: Das Digitalisierungsdilemma*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–78. ISBN: 978-3-662-56262-8. URL: https://doi.org/10.1007/978-3-662-56262-8_1.
- [153] Günter Müller and Rafael Accorsi. “Why Are Business Processes Not Secure?” In: *Number Theory and Cryptography – Papers in Honor of Johannes Buchmann on the Occasion of His 60th Birthday*. Ed. by Marc Fischlin and Stefan Katzenbeisser. Vol.

8260. Lecture Notes in Computer Science. Springer, 2013, pp. 240–254. ISBN: 978-3-642-42000-9.
- [154] Jens Müller, Jutta A. Mülle, Silvia von Stackelberg, and Klemens Böhm. “Secure Business Processes in Service-Oriented Architectures – A Requirements Analysis”. In: *8th IEEE European Conference on Web Services (ECOWS 2010), 1–3 December 2010, Ayia Napa, Cyprus*. Ed. by Antonio Brogi, Cesare Pautasso, and George Angelos Papadopoulos. IEEE Computer Society, 2010, pp. 35–42. ISBN: 978-1-4244-9397-5.
- [155] Tadao Murata. “Petri Nets: Properties, Analysis and Applications”. In: *Proceedings of the IEEE 77.4* (Apr. 1989), pp. 541–574.
- [156] Qun Ni, Elisa Bertino, and Jorge Lobo. “Risk-based access control systems built on fuzzy inferences”. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13–16, 2010*. Ed. by Dengguo Feng, David A. Basin, and Peng Liu. ACM, 2010, pp. 250–260. ISBN: 978-1-60558-936-7. URL: <https://doi.org/10.1145/1755688.1755719>.
- [157] Masato Notomi and Tadao Murata. “Hierarchically Organized Petri Net State Space for Reachability and Deadlock Analysis”. In: *Proceedings of the 6th International Parallel Processing Symposium, Beverly Hills, CA, USA, March 1992*. Ed. by Viktor K. Prasanna and Larry H. Canter. IEEE Computer Society, 1992, pp. 616–623. ISBN: 0-8186-2672-0. URL: <https://doi.org/10.1109/IPPS.1992.222996>.
- [158] Federica Paci, Rodolfo Ferrini, Yuqing Sun, and Elisa Bertino. “Authorization and User Failure Resiliency for WS-BPEL Business Processes”. In: *ICSOC*. Ed. by Athman Bouguettaya, Ingolf Krüger, and Tiziana Margaria. Vol. 5364. Lecture Notes in Computer Science. 2008, pp. 116–131. ISBN: 978-3-540-89647-0.
- [159] Pier Paolo Pasolini. “10 giugno 1962” (*Poem*), *Mamma Roma*. Ed. by (English translation of the author). Rizzoli, 1962.
- [160] James L. Peterson. “Petri Nets”. In: *ACM Comput. Surv.* 9.3 (1977), pp. 223–252. URL: <https://doi.org/10.1145/356698.356702>.
- [161] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. USA: Prentice Hall PTR, 1981. ISBN: 0136619835.
- [162] Carl Adam Petri. “Kommunikation mit Automaten”. Dissertation, Schriften des IIM. Bonn: Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, 1962.
- [163] Helmut Petritsch. *Break-Glass – Handling Exceptional Situations in Access Control*. Springer, 2014. ISBN: 978-3-658-07364-0. URL: <https://doi.org/10.1007/978-3-658-07365-7>.
- [164] Anastasiia Pika. “Mining process risks and resource profiles”. PhD thesis. Queensland University of Technology, 2015.
- [165] Anastasiia Pika, Michael Leyer, Moe ThandarWynn, Colin J. Fidge, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. “Mining Resource Profiles from Event Logs”. In: *ACM Trans. Management Inf. Syst.* 8.1 (2017), 1:1–1:30. URL: <https://doi.org/10.1145/3041218>.
- [166] Sonja Praviilovic, Annalisa Appice, and Donato Malerba. “Process Mining to Forecast the Future of Running Cases”. In: *New Frontiers in Mining Complex Patterns – Second International Workshop, NFMCP 2013, Held in Conjunction with ECML-PKDD 2013, Prague, Czech Republic, September 27, 2013, Revised Selected Papers*. Ed. by Annalisa Appice, Michelangelo Ceci, Corrado Loglisci, Giuseppe Manco, Elio Mas-

- ciari, and Zbigniew W. Ras. Vol. 8399. Lecture Notes in Computer Science. Springer, 2013, pp. 67–81. ISBN: 978-3-319-08406-0. URL: https://doi.org/10.1007/978-3-319-08407-7_5.
- [167] Lutz Priebe and Harro Wimmel. *Petri-Netze (German Edition)*. Springer, 2008. ISBN: 9783540769705.
- [168] Chander Ramchandani. “Analysis of asynchronous concurrent systems by timed Petri nets.” PhD thesis. Massachusetts Institute of Technology, 1973.
- [169] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. “Where Did I Misbehave? Diagnostic Information in Compliance Checking”. In: *Business Process Management – 10th International Conference, BPM 2012, Tallinn, Estonia, September 3–6, 2012. Proceedings. 2012*, pp. 262–278. URL: https://doi.org/10.1007/978-3-642-32885-5_21.
- [170] Jan C Recker and Jan Mendling. “On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages”. In: *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*. Namur University Press. 2006, pp. 521–532.
- [171] Wolfgang Reisig. “Petri Nets with Individual Tokens”. In: *Applications and Theory of Petri Nets. Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets, Varenna, Italy, September 27–30, 1982*. Ed. by Anastasia Pagnoni and Grzegorz Rozenberg. Vol. 66. Informatik-Fachberichte. Springer, 1982, pp. 229–249. ISBN: 3-540-12309-1. URL: https://doi.org/10.1007/978-3-642-69028-0_16.
- [172] Wolfgang Reisig. “Understanding Petri Nets Modeling Techniques, Analysis Methods, Case Studies”. In: *Bulletin of the EATCS 112 (2014)*. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/250>.
- [173] Wolfgang Reisig and Jörg Desel. “Konzepte der Petrinetze”. In: *Informatik Spektrum 37.3 (2014)*, pp. 172–190. URL: <https://doi.org/10.1007/s00287-013-0758-0>.
- [174] Grzegorz Rozenberg and Arto Salomaa, eds. *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, 1997. ISBN: 978-3-642-63863-3. URL: <https://doi.org/10.1007/978-3-642-59136-5>.
- [175] Anne Rozinat and Wil M. P. van der Aalst. “Conformance checking of processes based on monitoring real behavior.” In: *Information Systems 33.1 (Feb. 28, 2008)*, pp. 64–95.
- [176] Jan Saat, Ulrik Franke, Robert Lagerstrom, and Mathias Ekstedt. “Enterprise architecture meta models for IT/business alignment situations”. In: *Enterprise Distributed Object Computing Conference (EDOC), 2010 14th IEEE International*. IEEE. 2010, pp. 14–23.
- [177] Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini. “Modeling and Verifying Security Policies in Business Processes”. In: *Enterprise, Business-Process and Information Systems Modeling – 15th International Conference, BPMDS 2014, 19th International Conference, EMMSAD 2014, Held at CAiSE 2014, Thessaloniki, Greece, June 16–17, 2014*. Proceedings. Ed. by Ilia Bider, Khaled Gaaloul, John Krogstie, Selmin Nurcan, Henderik Alex Proper, Rainer Schmidt, and Pnina Soffer. Vol. 175. Lecture Notes in Business Information Processing. Springer, 2014, pp. 200–214. ISBN: 978-3-662-43744-5. URL: https://doi.org/10.1007/978-3-662-43745-2_14.
- [178] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. “Role-Based Access Control Models”. In: *IEEE Computer 29.2 (1996)*, pp. 38–47.

- [179] Ravi S. Sandhu and Jaehong Park. “Usage Control: A Vision for Next Generation Access Control”. In: *Computer Network Security, Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2003, St. Petersburg, Russia, September 21–23, 2003, Proceedings*. Ed. by Vladimir Gorodetsky, Leonard J. Popyack, and Victor A. Skormin. Vol. 2776. Lecture Notes in Computer Science. Springer, 2003, pp. 17–31. ISBN: 3-540-40797-9. URL: https://doi.org/10.1007/978-3-540-45215-7_2.
- [180] Ravi S. Sandhu and Pierangela Samarati. “Authentication, Access Control, and Audit”. In: *ACM Comput. Surv.* 28.1 (1996), pp. 241–243. URL: <https://doi.org/10.1145/234313.234412>.
- [181] Daniel Ricardo dos Santos. “Automatic Techniques for the Synthesis and Assisted Deployment of Security Policies in Workflow-based Applications”. PhD thesis. Trento, Italy: University of Trento, 2017.
- [182] Fred B. Schneider. “Enforceable security policies”. In: *ACM Trans. Inf. Syst. Secur.* 3.1 (2000), pp. 30–50. URL: <http://doi.acm.org/10.1145/353323.353382>.
- [183] Fred B. Schneider, J. Gregory Morrisett, and Robert Harper. “A Language-Based Approach to Security”. In: *Informatics – 10 Years Back. 10 Years Ahead*. Ed. by Reinhard Wilhelm. Vol. 2000. Lecture Notes in Computer Science. Springer, 2001, pp. 86–101. ISBN: 3-540-41635-8. URL: https://doi.org/10.1007/3-540-44577-3_6.
- [184] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. ISBN: 978-0-471-98232-6.
- [185] Alexander Schröder and Oliver Engels. “Compliance Management”. In: *Anforderungen an die Interne Revision: Grundsätze, Methoden, Perspektiven* 8 (2009), p. 315.
- [186] Manuel Silva, Enrique Teruel, and José M. Colom. “Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems.” In: *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*. Ed. by Reisig, W. and Rozenberg, G. Vol. 1491. Springer-Verlag, 1998, pp. 309–373.
- [187] Ramavarapu S. Sreenivas. “An Application of Independent, Increasing, Free-Choice Petri Nets to the Synthesis of Policies that Enforce Liveness in Arbitrary Petri Nets”. In: *Automatica* 34.12 (1998), pp. 1613–1615. URL: [https://doi.org/10.1016/S0005-1098\(98\)80016-2](https://doi.org/10.1016/S0005-1098(98)80016-2).
- [188] Ramavarapu S. Sreenivas. “On a free-choice equivalent of a Petri net”. In: *Proceedings of the 36th IEEE Conference on Decision and Control*. Vol. 4. IEEE, 1997, pp. 4092–4097.
- [189] Thomas Stocker. “Sicherheit in Geschäftsprozessen: Prozessrekonstruktion als Werkzeug nachgelagerter Prozessanalysen”. PhD thesis. Albert-Ludwigs-Universität Freiburg, 2014.
- [190] Thomas Stocker and Frank Böhr. “IF-Net: A Meta-Model for Security-Oriented Process Specification”. In: *STM*. Ed. by Rafael Accorsi and Silvio Ranise. Vol. 8203. Lecture Notes in Computer Science. Springer, 2013, pp. 191–206. ISBN: 978-3-642-41097-0.
- [191] Mark Strembeck and Jan Mendling. “Modeling process-related RBAC models with extended UML activity models”. In: *Information & Software Technology* 53.5 (2011), pp. 456–483. URL: <https://doi.org/10.1016/j.infsof.2010.11.015>.

- [192] Suriadi Suriadi, Robert Andrews, Arthur H. M. ter Hofstede, and Moe Thandar Wynn. “Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs”. In: *Inf. Syst.* 64 (2017), pp. 132–150. URL: <https://doi.org/10.1016/j.is.2016.07.011>.
- [193] Arnt Syring. *MERCoR-Semiformale Compliance-Regeln zur überwachung automatisierter Geschäftsprozesse*. Kovac, Dr. Verlag, 2014.
- [194] Kaijun Tan, Jason Crampton, and Carl A. Gunter. “The Consistency of Task-Based Authorization Constraints in Workflow Systems”. In: *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28–30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 155. ISBN: 0-7695-2169-X. URL: <http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.22>.
- [195] Niek Tax, Sebastiaan J. van Zelst, and Irene Teinemaa. “An Experimental Evaluation of the Generalizing Capabilities of Process Discovery Techniques and Black-Box Sequence Models”. In: *Enterprise, Business-Process and Information Systems Modeling – 19th International Conference, BPMDS 2018, 23rd International Conference, EMMSAD 2018, Held at CAiSE 2018, Tallinn, Estonia, June 11–12, 2018, Proceedings*. Ed. by Jens Gulden, Iris Reinhardt-Berger, Rainer Schmidt, Sérgio Guerreiro, Wided Guédria, and Palash Bera. Vol. 318. Lecture Notes in Business Information Processing. Springer, 2018, pp. 165–180. ISBN: 978-3-319-91703-0. URL: https://doi.org/10.1007/978-3-319-91704-7_11.
- [196] Hugh Taylor. *The Joy of SOX: Why Sarbanes-Oxley and Services Oriented Architecture May Be the Best Thing That Ever Happened to You*. John Wiley & Sons, 2006.
- [197] Farbod Taymouri. “ALI: Alignment for Large Instances”. In: URL: <https://www.cs.upc.edu/taymouri/tool.html> (visited on 10/25/2020).
- [198] Farbod Taymouri. “Light methods for conformance checking of business processes”. PhD thesis. UPC, Departament de Ciències de la Computació, Dec. 2018. URL: <http://hdl.handle.net/2117/127494>.
- [199] Farbod Taymouri and Josep Carmona. “Computing Alignments of Well-Formed Process Models using Local Search”. In: *ACM Transactions on Software Engineering and Methodology* 29.3 (July 2020), pp. 1–41. ISSN: 1557-7392. URL: <http://dx.doi.org/10.1145/3394056>.
- [200] Farbod Taymouri and Josep Carmona. “Model and Event Log Reductions to Boost the Computation of Alignments”. In: *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016), Graz, Austria, December 15–16, 2016*. Ed. by Paolo Ceravolo, Christian Guetl, and Stefanie Rinderle-Ma. Vol. 1757. CEURWorkshop Proceedings. CEUR-WS.org, 2016, pp. 50–62. URL: <http://ceur-ws.org/Vol-1757/paper4.pdf>.
- [201] Irene Teinemaa. “Predictive and Prescriptive Monitoring of Business Process Outcomes”. In: *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 co-located with 17th International Conference on Business Process Management, BPM 2019, Vienna, Austria, September 1–6, 2019*. Ed. by Benoît Depaire, Johannes De Smedt, Marlon Dumas, Dirk Fahland, Akhil Kumar, Henrik Leopold, Manfred Reichert, Stefanie Rinderle-Ma, Stefan Schulte, Stefan Seidel, and Wil M. P. van der Aalst. Vol. 2420. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 15–19. URL: <http://ceur-ws.org/Vol-2420/paperDA4.pdf>.

- [202] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. “Outcome-Oriented Predictive Process Monitoring: Review and Benchmark”. In: *TKDD* 13.2 (2019), 17:1–17:57. URL: <https://doi.org/10.1145/3301300>.
- [203] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Irene Teinemaa. “Survey and Cross-benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring”. In: *ACM Trans. Intell. Syst. Technol.* 10.4 (2019), 34:1–34:34.
- [204] Jacques Wainer, Paulo Barthelmeß, and Akhil Kumar. “W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints”. In: *Int. J. Cooperative Inf. Syst.* 12.4 (2003), pp. 455–485. URL: <https://doi.org/10.1142/S0218843003000814>.
- [205] Jianmin Wang, Shaoxu Song, Xuemin Lin, Xiaochen Zhu, and Jian Pei. “Cleaning structured event logs: A graph repair approach”. In: *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13–17, 2015*. Ed. by Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman. IEEE Computer Society, 2015, pp. 30–41. ISBN: 978-1-4799-7964-6. URL: <https://doi.org/10.1109/ICDE.2015.7113270>.
- [206] Qihua Wang and Ninghui Li. “Satisfiability and Resiliency in Workflow Authorization Systems”. In: *ACM Trans. Inf. Syst. Secur.* 13.4 (2010), 40:1–40:35. URL: <https://doi.org/10.1145/1880022.1880034>.
- [207] Qihua Wang and Ninghui Li. “Satisfiability and Resiliency in Workflow Authorization Systems”. In: *ACM Trans. Inf. Syst. Secur.* 13.4 (Dec. 2010), 40:1–40:35. ISSN: 1094–9224. URL: <http://doi.acm.org/10.1145/1880022.1880034>.
- [208] Qihua Wang and Ninghui Li. “Satisfiability and Resiliency in Workflow Systems”. In: *ESORICS*. Ed. by Joachim Biskup and Javier Lopez. Vol. 4734. Lecture Notes in Computer Science. Springer, 2007, pp. 90–105. ISBN: 978-3-540-74834-2.
- [209] Qihua Wang, Ninghui Li, and Hong Chen. “On the Security of Delegation in Access Control Systems”. In: *ESORICS*. Ed. by Sushil Jajodia and Javier López. Vol. 5283. Lecture Notes in Computer Science. Springer, 2008, pp. 317–332. ISBN: 978-3-540-88312-8.
- [210] Willis H Ware. *Security controls for computer systems. report of defense science board task force on computer security*. Tech. rep. Rand Corp Santa Monica CA, 1979.
- [211] Paul Watzlawick. “Menschliche Kommunikation”. In: *Formen, Störungen, Paradoxien* 6 (1969).
- [212] Mathias Weske. *Business Process Management – Concepts, Languages, Architectures, Third Edition*. Springer, 2019. ISBN: 978-3-662-59431-5. URL: <https://doi.org/10.1007/978-3-662-59432-2>.
- [213] Glynn Winskel. “Petri Nets, Algebras, Morphisms, and Compositionality”. In: *Inf. Comput.* 72.3 (1987), pp. 197–238. URL: [https://doi.org/10.1016/0890-5401\(87\)90032-0](https://doi.org/10.1016/0890-5401(87)90032-0).
- [214] Claus Hendrik Wonnemann. “Mechanismen zur Sicherheitszertifizierung formalisierter Geschäftsprozesse”. In: *Albert-Ludwigs-Universität Freiburg* (2011).
- [215] Richard Zahoransky. “Interdependenzen in Prozessarchitekturen: Simulation gegenseitiger Beeinflussung sicherheitsorientierter Geschäftsprozesse”. PhD thesis. Albert-Ludwigs-Universität Freiburg, 2017.

-
- [216] Richard M. Zahoransky, Julius Holderer, Adrian Lange, and Christian Brenig. “Process Analysis as First Step towards Automated Business Security”. In: *24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12–15, 2016. 2016*, Research Paper 46. URL: http://aisel.aisnet.org/ecis2016_rp/46.
 - [217] ZhaoLi Zhang, Fan Hong, and Junguo Liao. “Modeling Chinese Wall Policy Using Colored Petri Nets”. In: *CIT*. IEEE Computer Society, 2006, p. 162. ISBN: 0-7695-2687-X.
 - [218] ZhaoLi Zhang, Fan Hong, and Hai-Jun Xiao. “Verification of Strict Integrity Policy via Petri Nets”. In: *JCSNC*. IEEE Computer Society, 2006, p. 23.
 - [219] Chunfu Zhong, Wenlong He, Zhiwu Li, Naiqi Wu, and Ting Qu. “Deadlock analysis and control using Petri net decomposition techniques”. In: *Inf. Sci.* 482 (2019), pp. 440–456. URL: <https://doi.org/10.1016/j.ins.2019.01.029>.