Faculteit Economie en
Bedrijfswetenschappen

Katholieke
Universiteit
Leuven

Declarative Techniques for Modeling and Mining Business Processes

Proefschrift voorgedragen tot
het behalen van de graad van
Doctor in de Toegepaste
Economische Wetenschappen

door

Stijn Goedertier

Nummer 284 2008

Faculteit Economie en
Bedrijfswetenschappen

Katholieke
Universiteit
Leuven

Declarative Techniques for Modeling and Mining Business Processes

Proefschrift voorgedragen tot
het behalen van de graad van
Doctor in de Toegepaste
Economische Wetenschappen

door

Stijn Goedertier

Nummer 284 2008

Committee

Acknowledgements

There is probably much truth in the proverb that one goes faster alone, but further together. Its double-edged implication certainly applies to the work on my PhD dissertation. At first reflection, writing this dissertation seems to have been the autonomous act of formulating research questions, self-study, conceptualization, implementation, doing experiments, and writing down the results. Working autonomously has the advantage of not being burdened by the expense of synchronizing your thoughts with others. In this way, I have been able to make progress more quickly. A closer inspection, however, also reveals that at countless moments, writing my PhD has been all but a solitary act. It could not have been accomplished without the effort of many people. Their feedback, advice, and encouragements have allowed me to push the envelope beyond what I originally thought to be feasible. For this reason, I would like to mention just a few people to whom I owe my sincere respect and gratitude.

In the first place, I would like to thank my PhD supervisor Jan Vanthienen. As a full professor and head of our research group, Jan has many educational and academic responsibilities, but this never holds him back from doing important research in various areas such as knowledge representation, and information system architecture. His active involvement in the business rule community and acquaintance with founder Ron Ross has set the background of my PhD research. Our discussions in the first year of my PhD have helped me tremendously in getting into the business-rule mind set. At that time, Jan suggested me to write a PhD thesis on the relationship between rules and processes. Thank you, Jan, for this turned out to be a great idea! At the same time, Jan also gave me the freedom to actively explore new, fundamental research domains such as logic programming, machine learning, semantic web technologies, multi-agent systems, and process modeling. As a good supervisor, Jan has found financing for me to attend many international conferences and when I lacked the computing power to complete my benchmarking experiments, Jan provided me a machine on which I could remotely run my batches. Aside from work, Jan always has been prepared to have a casual drink, lunch, or dinner to discuss the more important matters in life. I happily remember our two-hour visit to Venice. Rather than being early in the airport, we went to visit Venice and discussed business process modeling over a slice of pizza on the Rialto bridge. Thank you, Jan, for all this effort. It has been a genuine pleasure working for you!

With regard to the finer details of workflow research, I am much indebted to Wil van der Aalst of the Technische Universiteit Eindhoven. He has coined the term 'declarative process modeling' and his seminal work on workflow modeling, process verification, and process mining has been most influential to my PhD. A quick look at the references to his work included in the back of this dissertation will immediately make this evident. As an attentive participant of conferences and workshops, Wil van der Aalst had also actively commented on my work even before he joined my PhD committee. These insightful comments, fresh ideas, and critical questions did not stop when he joined my PhD committee. As a key authority in the BPM field, Wil van der Aalst quickly identified the strong points and fallacies in my research and provided me with very relevant suggestions. During my two visits to the Technische Universiteit Eindhoven, he gave me the opportunity to present my work and get in touch with many of the Eindhoven-based researchers. Thank you, Wil, I feel honored that you are part of my PhD committee and I am grateful for all the time and effort you have put in.

I also was very fortunate to have Diogo Ferreira as an external committee member. His groundbreaking research on the application of planning, inductive logic programming, and data mining techniques to the field of business process management make him a pioneer in developing declarative techniques – *avant la lettre* – for process mining. His insightful comments, and critical questions have contributed much to this work. Furthermore, I much appreciate his help in giving a general direction to my PhD thesis and his encouragement to include older work arguing that one should "value the journey as much as the destination".

Bart Baesens has been another protagonist in the realization of this PhD thesis. I was fortunate to have him, for he has the empathy and quickness of thought to counsel PhD students in just about any topic. Bart is a world-renowned specialist in predictive data mining techniques for financial engineering applications in academia as well as in practice. His experience in benchmarking state-of-the art classification algorithms for credit scoring was very useful in the experimental setup comparing process discovery algorithms. I thank you Bart for indulging my many questions on these topics. Additionally, Bart also gave me the opportunity to figure as a speaker on a forum of the Data Mining Garden and introduced me to his industry contacts, allowing me to do a real-life process discovery experiment with a European telecom operator. Thank you Bart for this truly excellent support. I also much appreciate your inviting me to your home and discussing the text of my first seminar over a glass of old Port wine. Your ideas, interest in my work, and your sincere advice have helped me out on countless occasions.

I am also ever grateful to have Wilfried Lemahieu in my PhD committee. As a co-founder and Board member of the BPM forum Belgium, he is in an excellent position to counsel me on BPM related topics. I value his research about the service-based development of information system architectures. Thank you, Wilfried, for thoroughly reading through my text and providing me with detailed comments and critical remarks.

A necessary condition for succeeding in writing a PhD is having fun while working on it. I guess that David Martens greatly contributed to this aspect. Together, we have shared the office HOG 03.119 for over three years. During this time, I have learned to appreciate David's humor, quickness of thought, and work ethic. It took David only three years to complete his PhD. Sharing an office, I could not help but to learn a few things from him about acceptable classification models, benchmarking, and data mining. These influences are unmistakeably present in this text. David's vast knowhow in machine learning, statistics, and finance make him a financial engineering expert. He has a brilliant mind, and a contagious positive attitude that give him the potential to succeed both in business and academia.

I am also grateful for the stimulating atmosphere at the Leuven Institute for Research on Information Systems (LIRIS). For instance, I much enjoyed the discussions I had with Raf Haesen about service-oriented architecture, sorting out the many dependencies between functional and non-functional design concerns. Raf, we never quite got to the point at which we could write down our "theory of everything", but I look forward to reading your final solution in your PhD thesis! I also thank Lotte de Rore, whose experiences with software measurement and knowledge management at KBC will soon be the subject of a PhD thesis of her own. I also learned a lot from the work of my colleagues who got promoted these last four years. In particular, I recall the work of David Martens on ant colony optimization based and active-learning based classification techniques, Bjorn Cumps on business-ICT alignment, Johan Huysmans on comprehensible predictive models, Manu de Backer on horizontal and vertical business process verification, and Frank Goethals, who wrote the book on B2Bi and the extended enterprise. In the years to come, I am confident that we will here a lot about the work of Geert Monsieur and Jonas Poelmans. The research group also has a number of truly excellent researchers and teachers, such as Ferdi Put, Jacques Vandenblucke, Monique Snoeck, and Guido Dedene of whom I learned plenty during my master program.

I thank the people who took the time to consult me on process discovery techniques: Ana Karla Alves de Medeiros and Anne Rozinat from Technische Universiteit Eindhoven, Wim Hellemans of IKAN consulting, and Paul Aerts, Mark Messelis, Erwin Ruttens, and Patrick Van Der Velde of Telenet. Your ideas and experience with the practical aspects of business process intelligence have greatly contributed to this work. Thank you, Ana Karla, for taking the time to share your knowledge about your event logs, and your genetic miner algorithm with me and for giving me a copy of your PhD dissertation. Thank you, Anne, for sharing your knowledge about the benchmark metrics in the ProM framework.

I gratefully acknowledge the financial support of the Katholieke Universiteit Leuven for the doctoral research grant FLOF 3H040612. Information systems research is a design science that deals with the relevant, non-trivial information problems that many organizations in Belgium are confronted with. I am confident that in addition to our contribution to the literature, the techniques and insights gained from our research will disseminate into practice via the university's publications and educational programs.

If you work hard, you gotta play hard. The last four years, I have had the privilege of experiencing this wisdom with my dear friends of the "bloempotbende". Together we have completed our master's degree in business engineering. In four years' time, I have witnessed all of you obtaining leading positions in multinational consulting firms and financial institutions. I consider us lucky that up to now we have succeeded to fit in the right amount of pleasure in our busy time schedules. I hope we will continue doing this in the future! Many thanks to my dear friend Stijn Gasthuys and his wife Caroline Rauch for their sincere encouragements and allowing me to stay in touch with the real world out there. I am also very fortunate to have almighty destructor and IT guru Kris Bloemen, and kind, intelligent and hard-working Evy Hellinckx amongst my friends. Likewise, it has always been a pleasure to hear the delicate, but razor-sharp wits of Maarten Peeters, and the stories of the ever-pragmatic and clever Karin Odenthal. Last but not least, I was fortunate to work in the same research department as our friend Jade Herbots. Thanks, Jade, for your help, advice, and company. I hope, you liked having a "bloempot" friend around at work as much as I did.

I also need to thank my many dive buddies of CMAS scuba diving club Poseidon Leuven with whom I share my passion for aquatic sports. Among all the others, I wish to extend a special thanks to Filip Olaerts, Veerle Callens, Ronnie Sutens, Joeri Auwerx, Roger Oversteyns, Mieke Rens, Kobe Naesens, Jef Guelinckx, Joost Raeymaekers, Nelis Boucké, Lieven Thorrez, and Tiny Heremans. Being involved in our scuba diving club has been a blessing to me and I hope that I can continue doing this for many years to come.

I would also like to express my gratitude to my parents and family. Without their loving care and support, I would never have come to this point. For sure, my initial eagerness in pursuing a PhD stems from my father's calvinist life attitude and my mother's relentless enthusiasm, but, in the end, their advice and encouragements is what really has made me seeing it through. I thank my brothers Dries, Wouter, and Stefan for continuously broadening my perspective on the world, and my favorite sister Veerle, her husband Philippe, my niece Helena, and my nephews Laurens, and Simon for reminding me of what really matters in life.

And this leaves only you to mention, Cathérine, my clever girlfriend. Ever since I first laid eyes on you almost four years ago, I have been enchanted by your intellect, ethics, and sense of humor. Although I harbor the suspicion that a PhD "don't impress you all that much", your love and interest in me – first as a friend, then as my friend – have always been a huge support and I confidently look forward to our future together.

Stijn Goedertier – July 2008

CONTENTS

Committee **ii**

Acknowledgements **iii**

Contents **vii**

Samenvatting **xi**

1 Introduction **1**
- 1.1 Gaining Insight into Business Processes . . . 2
- 1.2 Structure and Contributions . . . 4

2 Declarative Process Modeling **9**
- 2.1 Flexibility and Compliance . . . 9
- 2.2 Design Principles . . . 11
- 2.3 Existing Languages . . . 19
- 2.4 Conclusion . . . 23

3 EM-BrA^{2}CE: a Unifying Framework **25**
- 3.1 Overview . . . 26
  - 3.1.1 State and State Space . . . 26
  - 3.1.2 Transition Types . . . 26
  - 3.1.3 Running Example . . . 28
- 3.2 The EM-BrA^{2}CE Vocabulary . . . 30
  - 3.2.1 Candidate Ontology Languages . . . 32
  - 3.2.2 SBVR as Ontology Language . . . 34
  - 3.2.3 The EM-BrA^{2}CE Vocabulary . . . 37
- 3.3 Execution Semantics . . . 53

3.3.1 Colored Petri Nets . . . 54
3.3.2 Places and Color Sets . . . 54
3.3.3 Transitions . . . 57
3.3.4 Unspecified semantics . . . 71
3.4 Conclusion . . . 73

4 Modeling and Mining within EM-BrA^{2}CE **75**
4.1 Documenting Business Concerns . . . 75
4.1.1 Data Concerns . . . 76
4.1.2 Control-flow . . . 78
4.1.3 Access Control . . . 85
4.1.4 Conclusion . . . 89
4.2 PENELOPE . . . 89
4.2.1 Preliminaries: the Event Calculus . . . 89
4.2.2 Language . . . 90
4.2.3 Visualization . . . 92
4.2.4 Verification . . . 96
4.2.5 Conclusion . . . 97
4.3 Simulation . . . 98
4.3.1 State Space . . . 98
4.3.2 Control Flow Concerns . . . 99
4.3.3 Access Control Concerns . . . 101
4.3.4 The Obtained Event Logs . . . 102
4.4 Declarative Process Mining . . . 103
4.4.1 Process Mining Tasks . . . 103
4.4.2 Declarative Mining Techniques . . . 105
4.5 Conclusion . . . 108

5 Declarative Process Discovery **109**
5.1 Process Discovery . . . 109
5.2 Related Work . . . 113
5.3 Preliminaries . . . 117
5.3.1 Inductive Logic Programming . . . 117
5.3.2 Event logs . . . 122
5.3.3 Petri nets . . . 123
5.4 Step 1: Frequent Temporal Constraints . . . 124
5.4.1 Frequent Temporal Constraints . . . 124
5.4.2 Deriving Parallelism and Locality . . . 126
5.5 Step 2: Artificial Negative Events . . . 128
5.5.1 A Configurable Completeness Assumption . . . 128
5.5.2 Generating Negative Events . . . 129
5.6 Step 3: Learn the Preconditions . . . 131
5.6.1 Process Discovery as Classification . . . 131
5.6.2 The Language Bias of AGNEs . . . 133
5.7 Step 4: Transformation into Petri Nets . . . 138

5.7.1 Rule-level Pruning 138
5.7.2 Petri Net Construction 139
5.8 Implementation 140
5.9 Towards Mining Access Rules 142
5.10 Conclusion 145

6 An Evaluation of Declarative Process Discovery **147**
6.1 New Recall and Precision Metrics 147
6.1.1 Existing Metrics 148
6.1.2 New Recall and Precision Metrics 150
6.2 Experimental Evaluation 152
6.2.1 Expressiveness 155
6.2.2 Robustness to Noise 157
6.2.3 Ability to Generalize 162
6.3 A Case Study 164
6.3.1 The Obtained Event Log 164
6.3.2 Validity of Assumptions 167
6.3.3 Results 168
6.4 Conclusion 175

7 Conclusion **177**
7.1 Conclusion 177
7.2 Limitations and Future Work 179
7.2.1 A Configurable Completeness Assumption 179
7.2.2 Learning Other Transition Types 180
7.2.3 Declarative Process Mining Models 180
7.2.4 Learning from Unstructured Processes 180

List of Figures **183**

List of Tables **185**

Bibliography **187**

Resume **205**

Doctoral Dissertations List **211**

Samenvatting

Organisaties worden vandaag de dag geconfronteerd met een schijnbare tegenstelling. Hoewel ze aan de ene kant veel geld geïnvesteerd hebben in de automatisering van hun bedrijfsprocessen, lijken ze hierdoor minder in staat om een goed inzicht te krijgen in de effectiviteit, efficiëntie, flexibiliteit, en conformiteit van deze processen. Deze paradox kan verklaard worden door een toename in schaalgrootte en complexiteit, die de samenwerking van steeds meer mensen vereist, mogelijks op verschillende plaatsen en tijdstippen. Informatiesystemen kunnen dergelijke bedrijfsprocessen automatiseren door de complexe coördinatietaken van bedrijfsprocessen te verbergen met metaforen zoals databanken, formulieren, documentstromen, en taakvakken. Het voordeel van deze organizatie van het werk is dat individuele actoren enkel die aspecten van bedrijfsprocessen moeten kennen die relevant zijn voor hun rol in de organisatie. De keerzijde van automatisering is dat organisaties hierdoor echter ook onvoldoende inzicht hebben in het volledige bedrijfsproces. Samen met de complexiteit verbergen informatiesystemen immers ook een hoop nuttige informatie.

Een gebrekkig inzicht in de bedrijfsprocessen bedreigt hun flexibiliteit en conformiteit. Flexibiliteit is belangrijk, omdat organisaties door continu wijzigende marktomstandigheden gedwongen worden hun bedrijfsprocessen snel en soepel aan te passen. Daarnaast moeten organisaties ook kunnen garanderen dan hun bedrijfsvoering conform is aan de wetten, richtlijnen, en normen die hun zowel extern als intern opgelegd worden. Zonder voldoende inzicht te hebben in de onderliggende bedrijfsprocessen, kunnen organisaties zeer moeilijk de impact van wijzigingen inschatten. Deze onwetendheid beperkt vaak onnodig de flexibiliteit. Voorts is het zo dat organisaties met onvoldoende inzicht in de eigen processen, geen enkele garantie kunnen bieden dat hun bedrijfsvoering conform is aan het geldende beleid en regelgeving.

Schandalen zoals de recent aan het licht gekomen fraude bij de Franse bank Société Générale tonen het belang aan van conformiteit en flexibiliteit. Door

het afleveren van valse bewijsstukken en het omzeilen van vaste controlemomenten, kon één effectenhandelaar een risicoloze arbitragehandel op prijsverschillen in futures omtoveren tot een risicovolle, speculatieve handel in deze financiële derivaten. Dit leidde tot een verlies van 4,9 miljard euro; het grootste verlies dat door een effectenhandelaar berokkend werd aan zijn werkgever. De niet-ingedekte, niet-geautoriseerde posities bleven lange tijd verborgen door een gebrekkige interne controle, en tekortkomingen in de IT beveiliging en toegangscontrole. Om hieraan te verhelpen, is het in de eerste plaats noodzakelijk om inzicht te verkrijgen in de operationele processen van de front office, middle office en back office en de hieraan gerelateerde controleprocessen. Dit wordt aangegeven in een actieplan dat auditor PricewaterhouseCoopers voor de bank opmaakte (PricewaterhouseCoopers, 2008).

Inzicht krijgen in bedrijfsprocessen

In deze tekst behandelen we twee benaderingen die gebruikt kunnen worden om het inzicht in de bedrijfsprocessen te verhogen: procesmodellering – *process modeling* – en procesontginning – *process mining.* Procesmodellering is de manuele constructie van een formeel model dat een relevant aspect van een bedrijfsproces beschrijft op basis van informatie die grotendeels verworven is uit interviews. Procesontginning, daarentegen, is de automatische constructie van een procesmodel op basis van de zogenaamde *event logs* uit informatiesystemen. In het onderzoek is getracht technieken te ontwikkelen voor procesmodellering en procesontginning die *declaratief* zijn.

Declaratieve Processmodellering

Procesmodellen moeten adequate informatie te verschaffen over de bedrijfsprocessen om zinvol te kunnen worden gebruikt bij hun ontwerp, implementatie, uitvoering, en analyse. Declaratieve procestalen, zoals geïntroduceerd door Pesic and van der Aalst (2006), bevatten deze informatie omdat ze de onderliggende bekommernissen die bedrijfsprocessen beïnvloeden, expliciet kunnen weergeven. We argumenteren dat een expliciet bewustzijn van deze bekommernissen toelaat om flexibiliteit en conformiteit van elkaar af te wegen, zowel tijdens het ontwerp als tijdens de uitvoering van bedrijfsprocessen.

In de tekst karakteriseren en motiveren we declaratieve procesmodelleringstechnieken, en nemen we een aantal bestaande technieken onder de loep. Uit een literatuurstudie concluderen we dat er reeds vele talen voor declaratieve procesmodellering bestaan. De tekst introduceert daarom een veralgemenend raamwerk voor declaratieve procesmodellering raamwerk waarbinnen deze bestaande talen gepositioneerd kunnen worden. Dit raamwerk heet het EM-BrA^{2}CE raamwerk, en staat voor ‘Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events’. Het bestaat uit een formele ontolgie en een formeel uitvoeringsmodel, dat gebruikt kan worden als een informele taal voor het documenteren van vele bedrijfsbekommernissen rond bedrijfsprocessen, en als een ontologische basis om bestaande en nieuwe talen voor declaratieve procesmodellering te vergelijken en te ontwikkelen. De formele ontolgie van de taal is gedefinieerd als uitbreiding op de nieuwe Semantics for Business Vocabulary and Business Rules (SBVR) specificatie (Object Management Group, 2008). Het formele uitvoeringsmodel is gedefinieerd in termen van een Colored Petri Net. Als *proof-of-concept* van het uitvoeringsmodel, werden twee declaratieve simulatiemodellen geïmplementeerd. De publicaties over deze onderwerpen zijn:

Goedertier, S. and Vanthienen, J. (2007b). A vocabulary and execution model for declarative service orchestration. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Proceedings of the 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 496–501. Springer

Goedertier, S. and Vanthienen, J. (2007a). Declarative process modeling with business vocabulary and business rules. In Meersman, R., Tari, Z., and Herrero, P., editors, *OTM Workshops (1)*, volume 4805 of *Lecture Notes in Computer Science*, pages 603–612. Springer

Het EM-BrA^{2}CE raamwerk legt het ontologische fundament van de technieken die in de verdere hoofdstukken van de tekst aan bod komen. Meer bepaald wordt een taal besproken voor het modelleren van de vervaldagen op rechten en plichten die ontstaan tussen de agenten in een bedrijfsproces. Deze taal heeft een logische grondslag in de event calculus, maar zijn vocabulaire en uitvoeringsmodel kan gesitueerd worden binnen het EM-BrA^{2}CE raamwerk. Voor deze taal wordt een techniek aangegeven om deze regels te visualiseren in de Business Process Modeling Notation (BPMN). Voorts wordt ook ingegaan op een techniek voor het modelleren van rol-gebaseerde regels voor toegangscontrole. De publicaties over deze onderwerpen zijn ondermeer:

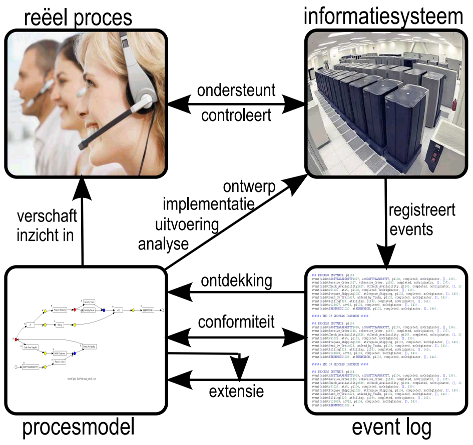
Goedertier, S. and Vanthienen, J. (2006d). Designing compliant business processes with obligations and permissions. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer

Goedertier, S., Mues, C., and Vanthienen, J. (2007d). Specifying process-aware access control rules in SBVR. In Paschke, A. and Biletskiy, Y., editors, *Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007)*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer. (Best Paper Award)

Declaratieve Processontgining

Niet alle informatie over bedrijfsprocessen kan vergaard worden uit interviews en bestaande documentatie. Factoren als procesautomatisering, en personeelsverloop kunnen maken dat deze kwalitatieve bronnen van informatie inadequaat en zelfs inaccuraat worden. Procesmodellen leggen in wezen vast hoe de bedrijfsprocessen er zouden moeten uitzien, maar zeggen niet met zekerheid hoe ze *in werkelijkheid* plaatsvinden.

Procesontginning, of process mining, is een nieuwe, kwantitatieve manier om na te gaan hoe de bedrijfsprocessen in werkelijkheid verlopen. Hiertoe wordt een analyse gemaakt van de *event logs* van informatiesystemen. Onderstaande figuur geeft de levenscyclus aan die verbonden is met de analyse van event logs. Vandaag de dag worden heel wat processen door informatiesystemen in event logs geregistreerd. In event logs vindt men in chronologische volgorde terug wie, wanneer, welke activiteit verricht heeft. De analyse van event logs kan een accuraat beeld opleveren van wat er zich in werkelijkheid afspeelt binnen een organisatie. Meer bepaald kunnen event logs gebruikt worden voor het aanvullen van bestaande procesmodellen met additionele informatie (extensie en performantieanalyse), en conformiteit (conformiteitsanalyse) van de bedrijfsprocessen, en voor het in kaart brengen van bedrijfsprocessen door de geautomatiseerde constructie van procesmodellen (ontdekkingsgebaseerde procesontginning of *process discovery*).



De levenscyclus van process mining (van der Aalst, 2007)

In dit doctoraat komen in de eerste plaats declaratieve technieken aan bod voor ontdekkingsgedreven procesontginning of process discovery. Het resultaat van deze analyse zijn formele procesmodellen. Om bruikbaar te zijn, moeten deze procesmodellen voldoen aan criteria zoals accuraatheid, verstaanbaarheid, en justifieerbaarheid. Accuraatheid verwijst naar de mate waarin het model het gedrag in het event log correct weergeeft. Verstaanbaarheid verwijst naar de mate waarin eindgebruikers het ontdekte model overzichtelijk en begrijpbaar vinden. Justifieerbaarheid verwijst naar de mate waarin het ontdekte model in overeenstemming is met de voorkennis en wensen van de eindgebruiker.

Bestaande technieken voor procesontginning focussen vooral op het eerste criterium: accuraatheid. *Declaratieve* technieken voor procesontginning richten zich ook op de verstaanbaarheid en justifieerbaarheid van de ontgonnen modellen. *Declaratieve* technieken voor procesontginning zijn meer verstaanbaar omdat ze pogen procesmodellen voor te stellen aan de hand van declaratieve voorstellingsvormen. Daarenboven verhogen declaratieve technieken de justifieerbaarheid van de ontgonnen modellen. Dit komt omdat deze technieken toelaten de apriori kennis, inductieve bias, en taal bias van een leeralgoritme in te stellen.

Inductief logisch programmeren (ILP) is een leertechniek die inherent declaratief is. Ferreira and Ferreira (2006) tonen hoe ontdekkingsgedreven procesontginning voorgesteld kan worden als een eerste-orde classificatieprobleem op event logs aangevuld met negatieve events. Dergelijke negatieve events geven weer dat een bepaalde statusovergang niet kon plaatsvinden. Wanneer we de transitietypes binnen het EM-BrA^{2}CE raamwerk beschouwen, kunnen vele process mining taken voorgesteld worden als een dergelijk classificatieprobleem.

In de tekst focussen we op één process mining taak in het bijzonder: *process discovery*; dit is het in kaart brengen van de volgorde beperkingen op de activiteiten in een bedrijfsproces. Vele event logs bevatten van nature geen negatieve events die aangeven dat een bepaalde activiteit niet kon plaatsvinden. In principe kan process discovery daarom niet voorgesteld worden als een classificatieprobleem dat onderscheid maakt tussen positieve en negatieve events en kan de techniek van Ferreira and Ferreira (2006) niet toegepast worden. Om aan dit probleem tegemoet te komen, beschrijven we een techniek om artificiële negatieve events te genereren, genaamd AGNEs (process discovery by Artificially Generated Negative Events). De generatie van artificiële negatieve events komt neer op een configureerbare inductieve bias. De AGNEs techniek is geïmplementeerd als een mining plugin in het ProM raamwerk.

De papers omtrent AGNEs zijn onder andere:

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2008a). Process Mining as First-Order Classification Learning on Logs with Negative Events. In *Proceedings of the 3rd Workshop on Business Processes Intelligence (BPI'07)*, volume 4928 of *Lecture Notes in Computer Science.* Springer

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008c). Robust process discovery with artificial negative events. resubmitted for review to the *Journal of Machine Learning Research* on September 1, 2008

Door process discovery voor te stellen als een eerste-orde classificatieprobleem op event logs met artificiële negatieve events, kunnen de traditionele metrieken voor het kwantificeren van specificiteit (specificity) en volledigheid (recall) toegepast worden voor het kwantificeren van de specificiteit en volledigheid van een procesmodel ten opzicht van een event log. In de tekst stellen we twee nieuwe metrieken voor: r_B^p (behavioral recall of positive events) and s_B^n (behavioral specificity of negative events).

Deze nieuwe metrieken, in combinatie met bestaande technieken, werden gebruikt voor een uitgebreide evaluatie van de AGNEs techniek voor process discovery in zowel een experimentele als een praktijkopstelling. De experimentele

opstelling is evalueert de expressiviteit, robuustheid aan ruis, en de mate waarin het algoritme in staat is om te veralgemenen op basis van onvolledige event logs. Een benchmarkstudie van deze omvang is nog niet verschenen in de literatuur rond process mining. De studie toont aan dat de AGNEs techniek expressief is, robuust is aan ruis, en in staat om te gaan met onvolledige event logs. De praktijkopstelling evalueert de schaalbaarheid van de AGNEs techniek met betrekking tot grote event logs en de mate waarin de techniek omgaat met verscheidene assumpties die traditioneel gemaakt worden door process discovery technieken. De praktijkopstelling vond plaats op event logs die geregistreerd werden bij een Europese telecomoperator. De resultaten van de praktijkopstelling wijzen uit dat AGNEs schaalbaar en robuust blijft, ondanks het feit dat de verkregen event log vele assumpties overtreedt die traditioneel gemaakt worden in het domein van process discovery. Niettemin is de AGNEs techniek voor process discovery nog steeds vatbaar voor vele verbeteringen. Deze en andere verbeteringen worden aangegeven in de slotconclusie van de tekst.

CHAPTER 1

Introduction

Organizations currently face an information paradox: although they automate an increasing number of their processes, they seem less capable of gaining insight into the effectiveness, efficiency, flexibility, and compliance of these processes. This can be explained by an increased scale and complexity of real-life processes, which involve more people who collaborate at different working hours and locations. To automate the coordination of such complex processes, organizations have grown to rely on information systems. Information systems hide the complexity of coordinating business processes with metaphors like databases, forms, document flows, and work queues. The advantage of this organization of work, is that individuals only need to know these aspects of processes that are relevant to their role in the organization. The downside of automating business process coordination, is that organizations are left with insufficient insight into the entire, end-to-end process. Together with complexity, information systems also hide much useful information about business processes.

A limited insight into business processes threatens their flexibility and conformance. Flexibility is important, because continuously changing conditions force organizations to rapidly and flexibly adapt their processes. Furthermore, organizations are required to guarantee compliance to regulations and policies. With insufficient insight into the underlying processes and their business concerns, managers cannot assess the impact of changes, and unnecessarily limit process flexibility. Furthermore, organizations that only have a limited understanding of their own processes have no way of guaranteeing compliance to policies and regulations. A good understanding is vital for verifying and guaranteeing business process compliance (Sadiq et al., 2007), setting up a coherent access control policy (Sandhu et al., 1996), and optimizing and redesigning business processes (Mansar and Reijers, 2005).

1.1 Gaining Insight into Business Processes

In this text, we deal with two approaches that can be used to raise an organization's understanding of its processes: business process modeling and business process mining. **Process modeling** aims at manually constructing a process model from information that is often obtained from interviews. **Process mining**, in contrast, is the automated collection of useful knowledge from information system event logs. Both process modeling and process mining are important tasks in the multi-disciplinary field of business process management (BPM). BPM comprises many management tasks involving the design, implementation, enactment, and evaluation of business processes (van der Aalst et al., 2003). These tasks can be structured in a so-called BPM life cycle (van der Aalst et al., 2003; van der Aalst and van Hee, 2002; zur Muehlen, 2004). Figure 1.1 situates process modeling and process mining in such a multi-disciplinary BPM life cycle.

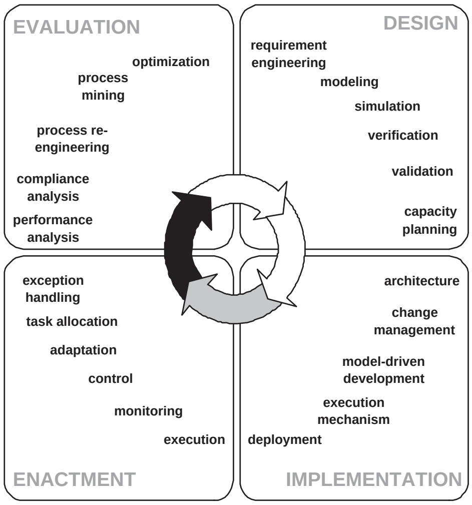


Figure 1.1: Process modeling and mining in a multi-disciplinary BPM life cycle

Process Modeling

Traditionally, practitioners have been obtaining information about processes from interviews and existing documentation. The obtained information gives practitioners an idea about the business concerns that govern business processes. From the obtained information process models are constructed. These process models are, among others, used for the purpose of simulation, verification, and the implementation of automated process support in information systems. Process modeling is a qualitative technique that can provide answers to business questions such as:

- Compliance: What are the procedures that are required in order to be compliant with regulations and policies? How do our employees say they carry out these procedures?
- Performance optimization and process redesign: How do the current and redesigned processes look like?
- Access control: Which access control policy is being used? Which access rights can be revoked without interfering with people's work?

The way business processes take place is determined by business concerns such as policies, regulations, time constraints, and profitability concerns. However, when modeling business processes, the underlying business concerns often remain implicit. Without properly documenting and formally modeling the underlying business concerns, it will, at a future time point, become difficult to make changes to processes, both during business process enactment and business process redesign.

Process Mining

Not all information about business processes can be obtained from interviews, and existing documentation. Due to factors like business process automation, and personnel attrition these inherently qualitative sources of information risk to become inadequate or even inaccurate. From interviews and documentation only, organizations cannot really know how process *actually* take place.

A new and promising way of acquiring insight into business processes is the analysis of the event logs of information systems. In many organizations, event logs conceal an untapped reservoir of knowledge about the way employees and customers conduct every-day business transactions. Event logs are ubiquitously available in many organizations. Popular Enterprise Resource Planning (ERP) systems such as SAP, Oracle e-Business Suite and workflow management systems (WfMSs) such as ARIS, TIBCO and Microsoft Biztalk already keep track of such event logs. Acquiring information from event logs is a new, quantitative technique that can provide answers to business questions such as:

- Compliance: How do customers or employees *actually* use the information system? Is the procedure that is used in practice in compliance with the procedure that is imposed by management or external regulations?
- Performance optimization and process redesign: Which processes have an above-average cycle time and where is the bottleneck situated? What are the conditions that affect routing choices at different decision points in the process model?
- Access control: Which users have access authorizations, but rarely make use of them? For which users groups that previously were manually granted authorizations, new authorization rules could be put in place?

Process modeling and mining are to a large extent complementary tasks. Process models can be verified by comparing them to the recordings in event logs or to discovered process models. In addition, a discovered process model can be of invaluable input to process modeling.

In order to be useful in practice, discovered process models must be accurate, comprehensible, and justifiable. Accuracy refers to the extent to which the induced model fits the behavior in the event log and can be generalized towards unseen behavior. Comprehensibility refers to the extent to which an induced model is comprehensible to end-users. Justifiability refers to the extent to which an induced model is aligned with the existing domain knowledge.

1.2 Structure and Contributions

This section outlines the main contributions of this text regarding process modeling and process mining.

Techniques for Declarative Process Modeling

We start in Chapter 2 from the aforementioned requirement of making business process concerns explicit with declarative process modeling languages. In an exploratory study, we have a look at the declarative modeling techniques in the literature. The contributions of this chapter include:

- A motivation for declarative process modeling.
- A characterization of declarative and procedural process modeling languages.
- An overview of existing languages for declarative process modeling.

From the literature review, it is concluded that there already exist a large number of declarative process modeling languages. Unfortunately, these languages each model only one specific aspect of the many concerns that exist in reality. Moreover, these languages are based on heterogeneous process ontologies (vocabularies) and make use of very different knowledge representation paradigms. To be able to integrate, compare, and develop new languages for declarative process modeling, we define in Chapter 3 a framework with a unifying ontology and execution model for declarative process modeling. This unifying framework is called called the EM-BrA^{2}CE Framework. The framework provides a formal vocabulary and an execution model, but it is not a language and does not provide model verification techniques, nor process visualization diagrams. Instead, it can be used both as an expressive informal language for documenting business concerns, and as an ontological foundation to compare and develop formal declarative languages. As a proof-of-concept, two simulation models of declarative process models were built. The contributions of this part include:

- A vocabulary for declarative process modeling, defined in terms of the new Semantics for Business Vocabulary and Business Rules (SBVR) specification (Object Management Group, 2008).
- An execution model for declarative process modeling.

- Sixteen patterns of declarative process modeling.
- A proof-of-concept consisting of two simulation models.

The publications about this topic are:

Goedertier, S. and Vanthienen, J. (2007b). A vocabulary and execution model for declarative service orchestration. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Proceedings of the 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 496–501. Springer

Goedertier, S. and Vanthienen, J. (2007a). Declarative process modeling with business vocabulary and business rules. In Meersman, R., Tari, Z., and Herrero, P., editors, *OTM Workshops (1)*, volume 4805 of *Lecture Notes in Computer Science*, pages 603–612. Springer

The EM-BrA^{2}CE Framework lays the ontological foundation for the techniques discussed in the subsequent chapters of this text. In Chapter 4, for instance, a language for expressing the due dates on obligations and permissions has been developed. This language has a logical underpinning in the event calculus, but its vocabulary and execution model can be situated within the EM-BrA^{2}CE Framework. The contributions of this chapter include:

- A language for modeling the due dates on obligations and permissions.
- A technique for visualizing rules about obligations and permissions in the Business Process Modeling Notation (BPMN).
- A technique for verbalizing role-based access control rules, formalized in defeasible logic.

The publications about these topics include:

Goedertier, S. and Vanthienen, J. (2006d). Designing compliant business processes with obligations and permissions. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer

Goedertier, S., Mues, C., and Vanthienen, J. (2007d). Specifying process-aware access control rules in SBVR. In Paschke, A. and Biletskiy, Y., editors, *Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007)*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer. (Best Paper Award)

In the remainder of the text, the ontology of the EM-BrA^{2}CE Framework is used for designing declarative process mining techniques. The simulation models are used to generate event logs.

Techniques for Declarative Process Mining

In the end of Chapter 4 we characterize declarative process mining. We start out from the requirements that process mining techniques must be accurate, comprehensible, and justifiable. Whereas existing process mining techniques focuss on accuracy only, declarative process mining techniques also target the comprehensibility and justifiability of the discovered knowledge. Inductive Logic Programming (ILP) is a machine learning technique that is particularly suited for building declarative process mining techniques. Ferreira and Ferreira (2006) show that process discovery can be formulated as an ILP classification learning problem on event logs with negative events. Negative events record when a state transition cannot take place. Considering the transition types in the EM-BrA^{2}CE Framework, we identify a number of process mining learning tasks that can be represented as ILP classification learning on event logs with negative events. The contributions of this part include:

- A characterization of declarative process mining techniques.
- A formulation of process mining tasks as a classification problem.

For many process mining tasks, negative events are not naturally present in the event logs.

In Chapter 5, we focus on process discovery. Event logs rarely contain negative events to record that a particular activity could not have taken place. Without negative events, in principle, process discovery cannot be represented as a classification problem that discriminates between positive and negative events. To overcome the problem, we develop a configurable technique to artificially generate negative events (AGNEs). By generating artificial negative events, a classification learner is given a configurable completeness assumption as inductive bias. Using an existing Inductive Logic Programming (ILP) classification learner, we have implemented a new process discovery technique as a mining plugin in the ProM framework. The contributions of this part include:

- A declarative technique for discovering frequent temporal constraints and for deriving parallelism and locality information.
- A configurable algorithm to generate artificial negative events for the purpose of process discovery.
- A configurable language bias specification that takes into account frequent temporal constraints and prior knowledge.
- An algorithm to convert a set of discovered classification rules into a Petri net representation.

This is included in the following papers:

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2008a). Process Mining as First-Order Classification Learning on Logs with Negative Events. In *Proceedings of the 3rd Workshop on Business Processes Intelligence (BPI'07)*, volume 4928 of *Lecture Notes in Computer Science*. Springer

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008c). Robust process discovery with artificial negative events. resubmitted for review to the *Journal of Machine Learning Research* on September 1, 2008

Having represented process discovery as a classification technique and having developed a technique for generating artificial negative events, the traditional metrics for evaluating classification models can be applied for quantifying the recall and precision of a discovered process model vis-à-vis an event log. This is one of the contributions of Chapter 6. In particular, we propose two new metrics: behavioral recall of positive events (r_B^p), and behavioral precision of negative events (p_B^n). These metrics use simple heuristics to determine a good firing sequence of duplicate and silent transitions. We motivate this choice with the comprehensibility requirement: human end-users are unlikely to use anything more than heuristics to determine a suitable firing sequence. As a result, the r_B^p and p_B^n metrics can be calculated even for complex process models, with many empty and duplicate transitions. Moreover, the r_B^p and p_B^n metrics can be calculated during the same event log replay, further contributing to their efficiency. The p_B^n is calculated from event log supplemented with artificial negative events. We argue that any precision metric based on the original event log makes a completeness assumption. This has its implications for the setup of cross-validation experiments. The contributions are:

- The new r_B^p and p_B^n metrics for quantifying the recall and preciseness of a process model vis-à-vis an event log, based on artificially generated negative events.

- The consequence of the completeness assumption on the setup of cross-validation experiments.

Using these new metrics, and some existing metrics for comparison, Chapter 6 describes an extensive evaluation of the AGNEs technique in both an experimental and real-life setup. The experimental setup aims at evaluating the performance of the technique in comparison to four state-of-the-art process discovery algorithms. Experiments allow to evaluate the expressiveness, robustness to noise, and ability to deal with incomplete events logs. A large-scale benchmark study of this kind is the first in process discovery literature. It shows that the AGNEs technique is expressive, robust to noise, and capable of dealing with incomplete event logs. The real-life setup aims at evaluating the scalability of the AGNEs technique with respect to real-life event logs, and also evaluates the extent to which AGNEs copes with several violations of assumptions that are generally made about process discovery event logs. The contributions are:

- A large-scale, comparative benchmark study regarding expressiveness, robustness, and ability to generalize.
- A real-life case study.

Chapter 7 concludes the text and summarizes the outcomes of the research. In addition, it identifies some important areas of future work.

CHAPTER 2

Declarative Process Modeling

Business process models needs to fulfill a number of requirements to be useful when designing, implementing, enacting, and analyzing business processes. In general, they must contribute to process flexibility, compliance, efficiency, and effectiveness, but this is a general requirement, which merely reflects the main goals of business process management. The real challenge for business process models consists of providing information systems with adequate information to deal with the often conflicting requirements of flexibility and compliance. A declarative approach to process modeling, as identified by Pesic and van der Aalst (2006), is likely to be capable of providing adequate information. The reason is that declarative process models explicitly reflect the underlying business concerns that govern business processes. An explicit awareness of these underlying business concerns allows to balance flexibility and compliance requirements, both at runtime and design-time. This chapter further characterizes declarative process modeling and provides an overview of existing languages.

2.1 Flexibility and Compliance

Business process reengineering (Davenport, 1993; Hammer and Champy, 1993) and business process redesign (Reijers and Limam, 2005) aim at improving the business process flexibility, compliance, efficiency, and effectiveness.

- **Process flexibility** is the extent to which an organization can deal with business process change occurring both at design-time and at run-time. Socio-economic factors like globalization, outsourcing, mergers and acquisitions have made business environments more complex and prone to change. In such a setting organizations must be able to flexibly adapt their business policy and business processes to accommodate new market situations.

The work of Suchman (1995) and Schmidt and Simone (1996) has sparked the idea of flexibility as a major requirement for computer-supported collaborative work (CSCW). In a special issue, the CSCW journal compiles an anthology of directions in the research on flexibility (Klein et al., 2000). Likewise, van der Aalst and Jablonski (2000) provide a taxonomy of change, suggest solutions, and discus open problems.

- Control over business processes is undoubtedly as important as their flexibility. **Process compliance** is the extent to which a process is in correspondence with **business policy**, the whole of internally defined business constraints, and **business regulation**, the whole of externally imposed business constraints. Recently organizations are confronted by an increasing number of regulators imposing regulations that potentially affect every process within their organization. The Sarbanes-Oxley Act, for instance, not only has a substantial impact on business processes such as accounting but also on IT processes such as access management and software release management (O'Conor, 2005). In general, compliance to internal policies and external regulations can be an important driver for automating business process support.

- **Process effectiveness** is the extent to which a business process realizes its business goals.

- **Process efficiency** is the extent to which the organization of the business process is capable of minimizing the amount of utilized resources such as personnel, materials, time, machine capacity.

Of the aforemention performance criteria, efficiency and effectiveness are to some extent secondary to flexibility and compliance. When processes are flexible and compliant, they are likely to be efficient and effective. Achieving both flexibility and compliance is however non-trivial.

When designing information systems, it is a challenge to strike the right balance between flexibility and compliance. An information system can at first sight make the business processes of organizations more compliant. By automating the coordination of work, organizations have better control over the activities that are undertaken by their employees or customers. Moreover, information systems can help organizations in demonstrating business process compliance by restricting the activities that can be undertaken. However, the downside of restrictive information systems is that automated business processes risk to become inflexible. In particular, ill-conceived automated processes have proved to be difficult to adapt at design time, where each changed requirement triggers a lengthy development cycle in which it is impossible to identify and include all control and correction steps a priori (Heinl et al., 1999). Moreover, at run time, automated processes are often found too rigid to deal with the contingencies of real-life situations (Sadiq et al., 2005). Consequently, resolving the paradox between flexibility and compliance is one of the major concerns of business process management.

2.2 Design Principles

In order to strike the right balance between compliance and flexibility, information systems must be provided with sufficient information to adequately deal with the idiosyncracies of every-day situations. Real-life business processes are affected by business concerns such as those enumerated in Table 2.1. Process modelers often only implicitly think about these business concerns when they model business processes and pay little attention to documenting why specific design choices have been made. Instead of making these concerns explicit, they are implicitly used to determine task control flows, information flows and work allocation schemes. In other words these aspects remain implicit but their effects are in a way hard-coded directly in procedural process models.

Table 2.1: Concerns that affect processes

Concern	Definition
regulations	Externally imposed directives such as among others legal requirements, standards, and contracts.
policies	Internally defined directives involving among others business strategies, tactics, and operational procedures.
benefits and costs	The incurred benefits, and costs of an activity.
time constraints	Concerns about concurrency, synchronization, due dates, and durations.
resource constraints	Capacity and availability constraints of the resources that carry out activities.
information prerequisites	The information required to make decisions.
non-functional requirements	Technical requirements such as throughput, and response time.
common-sense constraints	Common-sense constraints, such as the law of physics.

The paradigm shift from procedural to declarative process modeling has been identified by Pesic and van der Aalst (2006) who introduce the ConDec language for declarative process modeling. In this section we contrast procedural and declarative modeling approaches. A business process model is called *procedural* when it contains explicit, prescriptive information about how processes should proceed, but only implicitly keeps track of why these design choices have been made. When modeling business processes procedurally, modelers risk to make a number of modeling assumptions that are not present in the earlier specified requirements, this is called the assumption bias of a model. Consequently, procedural models risk to be over-specified as they are likely to impose more restrictions on the control flow, information flow and work allocation in business process models than is strictly required. A process model is *declarative* when it explicitly takes into account the concerns that govern a process. Declarative modeling models the minimal business concerns that exist, and leave as much freedom as is permissible to determine a valid and suitable execution plan at execution time. Declarative process modeling does not merely focus on *how* an end state *must* be reached, but rather considers what *is*, *must*, *ought* and *can* be done in order to achieve the business goals. Declarative process models are modeled with declarative languages. Table 2.2 summarizes the differences between procedural and declarative process modeling. They can be considered as design principles of declarative languages.

Table 2.2: Procedural versus declarative process modeling

	Procedural modeling	**Declarative modeling**
Business concerns	implicit	explicit
Rule enforcement	what, when and how	what
Communication	what, how	what
Execution scenario	design-time	run-time
Execution mechanism	state-driven	goal-driven
Model granularity	process-centric	activity-centric
Modality	what *must*	what *must*, *ought* and *can*
Assumption bias	over-specified	under-specified
Alteration	design time	design and run time
Coordinator/Worker	human-machine	agent
Coordination/Activity	coordination $\neq$ activity	coordination = activity
Activity life cycle	single event	multiple life cycle events
Language	procedural	declarative

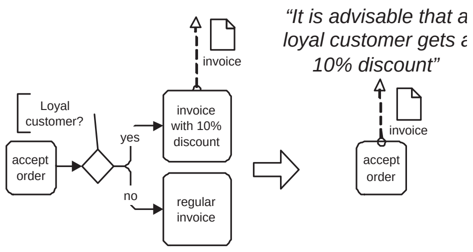
These differences in design principles are discussed in the subsequent sections. However, no dichotomy is implied. Process modeling languages can combine both procedural and declarative modeling aspects. Moreover, as both modeling paradigms are complementary, they can potentially be used together to realize their combined advantages. At design-time, for instance, business concerns can be made traceable by carefully documenting them using a declarative modeling approach, whereas at runtime business process support is implemented using a procedural business process language. Another synergy can be realized by modeling business concerns as atomic units of business logic in a declarative modeling languages, and visualizing a subset of these concerns using a graphical model of a procedural modeling language. The choice of modeling language also depends on the application domain. Dynamic, human-centric, non-standardized business process are most likely to require the run-time flexibility offered by of declarative process modeling. Examples are, for instance, order processing, the handling of distress calls in calling centers, insurance claim handling, or the coordination of patient processes in hospitals. Static, machine-centric, standardized business processes are most likely only to require a procedural representation of the coordination work. Examples are for instance the processing of production orders or standardized financial transactions.

Business Concerns Made Explicit

Declarative process modeling makes the underlying business concerns explicit in the form of business vocabulary and business rules. Business rules are atomic, formal expressions of business concerns. They are idealized as a common language between the business-side and IT-side of organizations. Such a language allows the business-side to formally represent models of how it operates internally and how it can legally interact with business partners. At the same time, such a common language allows the IT-side to have Information Systems support business processes accordingly, with as little development effort as possible. Ideally, Information System Technology must support declarative business process models in such a way that they become human-understandable, yet machine-executable specifications. In this way, changes to policies and regulations can be traced back to the business processes were they are to be enforced, without any need for translation.

Declarative Business Rule Enforcement

Procedural process languages predominantly focus on the control-flow perspective of business processes. In such process languages it might be possible to **enforce business rules** using a control-flow-based modeling construct. For instance, the enforcement of a derivation or integrity constraint can be directly modeled in BPEL as a calculation or input validation step. The left-hand side of Figure 2.1 represents an excerpt from a BPMN model that models the enforcement of a discount rule as a decision shape in BPMN. Another example involves the enforcement of authorization rules. The left-hand side of Figure 2.2 models an authorization rule as a decision shape in BPMN. The disadvantage of procedural process modeling is that business rules cannot be formulated independently from the process models in which they are to be enforced. Consequently, the same business rule is often duplicated in several procedural process models. When the business rule changes it is likely that all process models must be reexamined. Declarative process modeling separates business rule modeling from business rule enforcement. In particular, it does not make use of control flow to indicate when and how business rules are to be enforced. Instead, it is left to the execution semantics of the declarative process models to define an execution model in which different kinds of business rules are automatically enforced. The latter is indicated in the right-hand side of Figures 2.1 and 2.2. This separation of business rule specification and enforcement facilitates design-time flexibility.



invoice
Loyal customer?
accept order
yes
no
invoice with 10% discount
regular invoice
"It is advisable that a loyal customer gets a 10% discount"
invoice
accept order

Figure 2.1: Declarative rule enforcement: derivation

Declarative Communication Logic

Procedural process models are overburdened with **communication activities** intended to notify an external business partner about the occurrence of a relevant business event or to transmit information. Figure 2.3 represents an excerpt

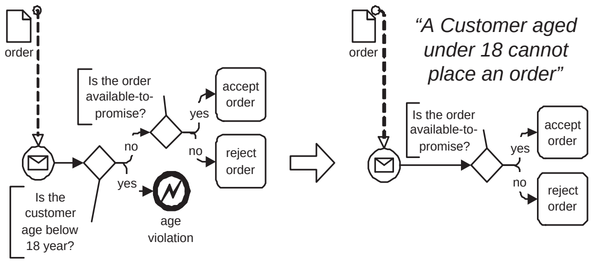

order
Is the order available-to-promise?
yes
no
accept order
reject order
Is the customer age below 18 year?
yes
age violation
order
"A Customer aged under 18 cannot place an order"
Is the order available-to-promise?
yes
no
accept order
reject order

Figure 2.2: Declarative rule enforcement: authorization

from the BPMN specification (Object Management Group, 2006b, p. 107) that contains the communication activities 'receive order' and 'send invoice'. Such communication activities depict communication logic in a procedural manner, because they specify how and when business events are communicated and information is transmitted. Declarative process models are only concerned with the ability of business agents to perceive business events and business concepts. When an agent (for instance a business partner) can perceive a particular event, the event becomes non-repudiable to the agent, irrespective of how the agent is notified of the event. The execution semantics of a declarative process model determines how events are communicated. In particular, events can be communicated as messages that are sent by the producer (push model), retrieved by the consumer (pull model), or via a publish-subscribe mechanism.

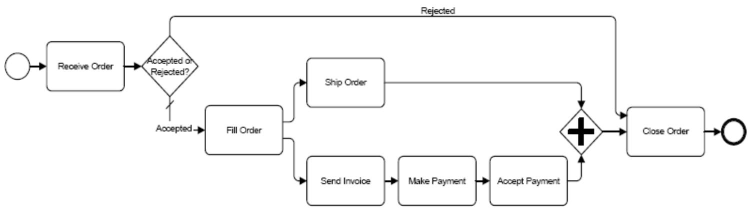


Figure 2.3: Separating communication logic from process models (Object Management Group, 2006b, p. 107)

Dynamic, Goal-driven Execution

Unlike procedural process modeling, declarative process modeling does not involve the pre-computation of task control flows, information flows and work allocation schemes. Whereas procedural process models inherently contain pre-computed activity dependencies, these activity dependencies remain implicit in declarative process models. An explicit enumeration of all activity dependencies is often not required – and often even difficult to obtain (Heinl et al., 1999). For model checking (verification) purposes, execution trajectories can still be obtained from implicit process models. During the execution of a declarative process model, a suitable execution scenario is constructed (either by a human or machine coordinator) that realizes the business goals of the process model. The latter is called **goal-driven** execution and its automation is akin to planning in the domain of Artificial Intelligence (Nau et al., 2004). In contrast, the execution mechanism of procedural process modeling languages is called **state-driven**.

Activity-level Granularity

Declarative process models also have a more fine-grained **model granularity** than procedural process models. Whereas procedural process languages are **process centric** in that they model business processes, declarative process languages are **activity centric**, as they model the business concerns related to a set of activity types. Business process models are composed of activity types, but the same activity type can occur in multiple business process models. In addition, many business concerns range over activity types and are not specific to one business process model in particular. Therefore activity-centric models have the advantage that these governing aspects are not a-priori straitjacketed into a particular business process model. For instance, the regulation that a purchase order must never be paid prior to the reception of an invoice, can possibly be relevant in different business processes. To allow the reuse of this regulation, it must be specified across the boundaries of artificially delineated business process models. Although the process-oriented view on organizations has lead to a better understanding of the value chain (Davenport, 1993; Porter, 1985) and has improved business process redesign, there is little motivation in letting this process-centricity set the granularity for process modeling. When required, a process-centric model can be obtained from an activity-centric model, the converse is not generally true.

Differentiation by Modality

Another point of difference is the **modality** that is attached to the information in process models. Procedural process models inherently have the necessity modality (what *must*) attached, whereas procedural process languages allow to differentiate by attaching different modalities like intention (what *ought*), advice (what *should*), possibility (what *can*) and factuality (what *is*) to parts of the process model. These modalities offer run-time flexibility. In particular, they allow to distinguish between what is strictly required (hard constraint) and what is merely desirable (soft constraint) behavior in a business process. This can help the coordinator of a business process to come up with a suitable yet valid execution plan.

The idea of different modalities is related to the research of Suchman (1995), Schmidt and Simone (1996) and Ross (2003). Suchman (1995) points out that business process models can never fully represent cooperative work in all its facets. In any organization, representations of work are required to create a common understanding of the work and thus facilitate coordination. However, workers may and should have conflicting views on the work. Suchman warns that a normative, *prescriptive account* of how the work gets done might severely differ from specific working practices. Although representations of work are a useful tool to reason about work and to steer activities, they risk to become useless when used outside the context of the work. According to the seminal work of Suchman (1987) representations of work need to be under-specified such that they are *plans for situated action*, in which the worker uses a plan as a guideline to go about but also determines the most suitable activity to undertake by himself from the context of the process *in situ*. To emphasize her point Suchman uses the metaphor of a map. "*Just as it would seem absurd to claim that a map in some strong sense controlled the travelers movements through the world, it is wrong to imagine plans as controlling actions. On the other hand, the question of how a map is produced for specific purposes, how in any actual instance it is interpreted vis-à-vis the world, and how its use is a resource for traversing the world, is a reasonable and productive one.*"

The situated action perspective on process models implies that there is little benefit in the automation of work coordination. However, this implication is not in congruence with the empirical evidence of many successful BPM systems and similar implementations found in contemporary organizations. Regrettably, the outcome of Suchman's work resulted in a decreased interest of some CSCW researchers in workflow management systems (WfMS) (van der Aalst, 2007). It is likely that Suchman's view on the use of process models and workflow management systems (WfMS) needs to be refined. Schmidt and Simone (1996) and Schmidt (1999) distinguish between two, according to them equally possible, accounts of process models by contrasting the metaphor of process models as *scripts* with Suchman's metaphor (1987) of process models as *maps*. A business process model can play the role of a *script* when it contains explicit, prescriptive information about how processes should proceed by making pre-computations of task dependencies. "*A script offers a limited selection of safe, secure, legal, valid, advisable, efficient or otherwise prescribed moves while excluding moves that generally would be considered unsafe, etc*" The application of a script can relieve the worker of computing "*a myriad of task interdependencies*" and optimization concerns. Conversely, a business process model can play the role of a *map* when it contains a codified set of functional requirements that provide a heuristic framework for distributed decision making. It is important that in the vision of Schmidt and Simone a same process model can be either a script or a map, depending on whether the context of the process conforms to relevant, pre-defined task interdependencies or not.

An even more refined view of Schmidt and Simone's dichotomy is to acknowledge that a process model can be used as a script and a map *within the same execution context*. Interestingly this combined view corresponds to the relationship between business process models and business rules depicted by (Ross, 2003). In Ross' view, process models consist of a set of pre-computed task dependencies, effectively called scripts, that can be supplemented with business rules that are either a number of **strict rules** that must be observed at all times or a number of heuristic **guidelines** that can just as easily be discarded. In terms of Suchman's traveler metaphor this would suggest the existence of a series of predefined sub-trajectories, scripts, that control a traveler's movements and a number of strict rules and guidelines that give direction to a travelers's movements, but leave the traveler with some freedom in choosing his or her own destination and trajectory.

Assumption Bias

The business rules in declarative process models can be traced back to an original business concern. Consequently, declarative process models are likely only to contain a minimum of constraints regarding a particular business process. This is not generally the case for procedural process models. Because procedural process models are the result of an implicit pre-computation of task dependencies, it is not generally guaranteed that procedural process models do not include a number of additional assumptions that *overly specify* the underlying business process. The claim that procedural process models are often over specified was first made by Pesic and van der Aalst (2006).

Runtime Alteration

The declarative information about a business process allows a coordinator to reason about the effect of run-time alteration of the execution plan. Such adaption can be seen as deviating from the outlined soft constraints to better fit the idiosyncracies and contingencies of real-life situations. Procedural process models do not allow this form of reasoning. In procedural process models all control flows, information flows and work allocation policies have been pre-computed. Without information about the strict business rules (hard constraints) and guidelines (soft constraints) that have led to a particular process model, it is difficult to reason about the effect of a run-time alteration.

No Human-Machine Distinction

Information systems and machinery have lead to an extensive automation of both work and coordination work. But not all activities in every business process can be fully automated. Likewise, not every business process lends itself to the same degree of automated coordination. In many cases, some of the (coordination) work is performed by machines and some of it by humans. Ideally, declarative process models make abstraction from the differences between humans and machines in performing (coordination) work. Rather than making an ontological distinction between concepts like humans and machines, both concepts are unified through the use of the **agent** metaphor (Woolridge and Wooldridge, 2001). Agents can be entire organizations, organizational units or individual workers and machines. In many cases, individual agents – whether humans, machines or a combination of both – act on behalf of the organization to which they pertain. For example, a transport activity might require the scheduling of a driver, truck, and trailer that act as an ad-hoc group of agents.

Coordination Work is Work

Business process management is about the **coordination of work** (Schmidt and Simone, 1996). Procedural process models are often an explicit specification of the coordination work. In contrast, declarative process models make no difference between coordination work and regular work. What may appear as work to an external agent, may very well be coordination work to another agent. For instance, a sales representative may instruct the expedition department to ship an order by a particular due date, but this activity may conceal the coordination of many other activities within the expedition department.

Multi-state Activities

Procedural process models do not explicitly consider the **life cycle** of the activities within a business process, but represent activities as actions that happen instantaneously. This is, for instance, the case for workflow nets (Eshuis and Dehnert, 2003; van der Aalst et al., 1994). Whereas this simplifying representation might be a useful abstraction for process visualization or verification purposes, it does not take into account the fact that activities have a life cycle of their own that consists of creation, planning, execution, and exception handling events. Declarative process models, in contrast, consider other events in the life cycle of activities such as the creation, scheduling, assignment, start, fact manipulation, completion, skipping, cancelation and redoing of an activity.

Third-person Perspective

The growing popularity of the Internet based on new IP-based communication protocols and technologies such as XML, has given rise to the requirement of automated coordination of business processes across the boundaries of individual organizations. As a consequence, it is not always technically or economically viable to have processes coordinated centrally. Another consequence of distribution is that it is unlikely that process designers can come up with only one representation of work. In many cases all business partners that participate in a cooperation might have different representations of the cooperative work. These representations are to be kept in part private from other process business partners.

Declarative business process models must take into account these disparate perspectives on processes. When modeling behavior it is proposed to adopt a *third-person perspective* – what will an actor with a particular role do in response to what others do? – rather than a *first-person perspective* – what will I do in response to what others do? In a third-person perspective all roles, actors and organization structures are named without the modeler adopting a particular viewpoint. A third-person modeling perspective has the advantage that it is possible to distinguish multiple interacting actors within a single organization. Another advantage is that business rules can be more easily shared in a business community when they are expressed from a third-person perspective. The modeling perspective distinction is, for instance, present in the literature about process orchestration and choreography (Bussler, 2001; Peltz, 2003) or in the distinction between internal and external Agent-Object-Relationship models (Wagner, 2003).

2.3 Existing Languages

A common idea of declarative business process modeling is that a process is seen as a *trajectory* in a *state space* and that declarative constraints are used to define the valid movements in that state space. The differences between declarative process languages can in part be brought back to a different perception of *state space*, *transition types*, and *transition constraints*. Table 2.3 provides a summary of a number of declarative process modeling languages in the literature, and enumerates the state space composition, transition types, and transition constraints for these languages.

Table 2.3: A chronological overview of declarative process modeling languages

Reference	State Space	Summary
ADEPT($_{\text{flex}}$) (Reichert and Dadam, 1998)	data object state, activity state	▸ The freedom of choice to change the process model of the process instance at runtime, while preserving control flow and data flow consistency regarding the addition, deletion and movement of tasks.
case handling (van der Aalst et al., 2005)	data object state, activity state	▸ The freedom of choice to complete, skip, and redo activities within a number of preconditions and postconditions (hard constraints) involving the completion of preceding activities and the data object state.
OWL-S (The OWL Services Coalition, 2004)	concept state	▸ The description of web services in terms of their inputs, outputs, preconditions, and effects (IOPEs).
WSMO (Roman et al., 2005)	concept state	▸ The description of web services in terms of their inputs, outputs, preconditions, and effects (IOPEs).
constraint specification (Sadiq et al., 2005)	activity trace	▸ A constraint specification framework with order, fork, serial, exclusion, and inclusion constraints (hard constraints) over a state space composed of activity traces that can occur within a sub-process (a pocket of flexibility).
(Ferreira and Ferreira, 2006)	proposition state	▸ An integrated life cycle of the planning, user-feedback, and automated learning of the process logic of activities, represented in terms of their preconditions, and effects on case data with first-order logic.
ConDec (Pesic and van der Aalst, 2006)	event history	▸ A template language for Linear Temporal Logic (LTL) that describes the temporal relationships that must be observed (hard constraints) by the activities in a process.
PENELOPE (Goedertier and Vanthienen, 2006d)	event history, time	▸ Allows the modeling of the obligations and permissions that arise from (not) performing activities within specific due dates.
Artifact-centric modeling (Bhattacharya et al., 2007)	artifact state	▸ The modeling of the preconditions and effects on artifacts by executing services and the specification of business rules that specify when particular services are to be invoked.

Reichert and Dadam (1998) describe the rationale of the ADEPT($_{\text{flex}}$) workflow management system (WfMS) in which end-users can change the process model of the process instance at runtime. ADEPT($_{\text{flex}}$) provides extensive user support to prevent non-permissible structural changes. In particular ADEPT($_{\text{flex}}$) preserves control flow and data flow consistency regarding the addition, deletion and movement of tasks.

van der Aalst et al. (2005) describe the formal semantics of the case handling paradigm. Case handling is one of the few declarative modeling approaches that originates from commercial workflow management systems (WfMSs), in particular, the FLOWer WfMS of Pallas Athena. It provides the user with the freedom of choice to complete, skip, and redo activities within a number of constraints based on availability of case data and the executed activities. The state space of the case-handling paradigm comprises the state of case data objects and activities. Furthermore, the system consists of data transition types (such as define, and confirm) and and activity transitions types (such as complete, skip, and redo). Although there is still a *preferred* or *normal* control-flow defined between the activities, much of the semantics of a case handling model resides in the mandatory constraint. For each activity definition, a modeler must indicate whether particular data objects are *mandatory* in order to be able to complete the activity.

In the constraint specification framework of Sadiq et al. (2005), order, fork, serialism, exclusion, and inclusion constraints can be specified in a state space composed of activity traces. The authors also show how it can be advantageous to combine both declarative and procedural aspects in process models. They present a foundation set of constraints for *partial* process modeling. A process model can contain, in addition to pre-defined activities and control flow, several so-called *pockets of flexibility.* Such pockets consist of activities, sub-processes and so-called order and inclusion constraints. Each time during enactment when a pocket of flexibility is encountered, the elicitation of the work within the pocket is done by a human end-user through a so-called "build" activity. During this build activity, the end-user constructs an instance template, a process specification, that satisfies the constraints set of the pocket of flexibility. The authors describe verification techniques for detecting redundant and conflicting constraints in the constraint set. The language has been implemented as a part of the WfMS prototype Chameleon.

The semantic web community also has produced a number of specifications that contain aspects of declarative process modeling. The largest initiatives can be situated within the the frameworks of the OWL-based Web Service Ontology (OWL-S) (The OWL Services Coalition, 2004) and the Web Service Modeling Ontology (WSMO) (Roman et al., 2005) ontologies. Although OWL-S and WSMO propose different sets of languages for expressing logical conditions, they both intend to describe the interface of semantic web services in terms of their inputs, outputs, preconditions, and effects (IOPEs). The state space of a semantic web service consists of the concepts, attributes of concepts, and relationships of concepts. WSMO allows service requesters to define a goal state, and the execution model is to select and invoke the web service that best satisfies this goal. The transition type can be considered to be the binding of a goal to web services, and the act of invoking a web service. The preconditions of a web service are transition constraints that determine whether a particular web service can provide the service, given the provided inputs and the state of the system. Using goals, preconditions, and effects, AI planning techniques can be used to plan a suitable invocation sequence of web services (Kopecký et al., 2006). In summary, semantic web services provide the freedom of choice to select web service and to determine a suitable invocation plan.

Ferreira and Ferreira (2005, 2006) consider an approach that aims at the run-time learning and planning of process models, rather than the design-time modeling. The state of a process consists of propositions about concepts. Performing an activity transforms the system from one particular state to another, so the transition type in the language is the act of performing an activity. The language allows to specify the preconditions of activities in terms of first-order logic. Preconditions are transitions constraints within the language: an activity can only be performed when its preconditions are satisfied. Furthermore, the effects of performing an activity can be described in terms of first-order literals that describe the sets of propositions that are either added (add-list) or removed (remove-lists) from the previous state. Rather than requiring the design-time specification of the preconditions and effects of activities, the authors propose the use of inductive logic programming techniques to discover these specifications at run-time. A partial-order planner is used to suggest possible execution plans, based on the current knowledge about preconditions and effects, that can either be accepted or rejected by the end-user. The flexibility of the approach stems from the fact that it does not require the design-time specification of an overly restrictive process model, and the run-time suggestion of alternative execution plans.

Pesic et al. (2007b); Pesic and van der Aalst (2006) propose a declarative language for modeling, enacting, and verifying declarative process specifications. Enacting a ConDec process model generates a trace of events, called the event history. The state space of the language can be seen as the set of all possible event histories. Each time an activity is performed, this is recorded as an event in the event history, so performing activities is the transition type considered by the ConDec language. The language allows specifying twenty constraint types that are defined as constraint templates in Linear Temporal Logic (LTL). Three classes of constraint types are included: existence constraints, relation constraints, and negation constraints. Existence constraints are activity cardinality constraints that specify how many times an activity of a particular type can be executed in a given process instance. Relation constraints are activity order constraints that specify the ordering between activity types and their existence dependencies. Negation constraints are activity exclusion constraints that specify that the occurrence of activities of some activity type exclude others. Some constraint types must be satisfied prior to the execution of a particular activity, whereas other constraints must only be satisfied upon termination of the process instance. A ConDec process model is a combination of LTL expressions that can be converted into a Buchi automaton, useful for the enactment and verification of the system. A ConDec process model can be verified by checking whether the model contains dead activities or conflicting constraints. The ConDec language is part of the DECLARE WfMS prototype (Pesic et al., 2007a).

Several authors describe languages for intelligent agents to reason about contract state (Governatori, 2005; Knottenbelt and Clark, 2004; Marín and Sartor, 1999; Paschke and Bichler, 2005; Yolum and Singh, 2004). Contracts represent regulations and are important business concern governing business processes. Contract modeling is akin to declarative process modeling. In Goedertier and Vanthienen (2006d) we define a language for modeling a contract in terms of a set of so-called temporal deontic assignment rules. This language is the PENELOPE language. The properties of this language are discussed in detail in Section 4.2. In the language, temporal deontic assignments are obligations and permissions of agents to perform a particular activity within an indicated deadline. The existence of temporal deontic assignments depends on earlier performed activities and the system time. Therefore, the state space of the PENELOPE language consists of the event history and the system time. The language considers two transition types: performing activities and deadline violations. Transitions in the language are constrained by the requirement that obligations and permissions should not be violated (soft constraint). The specification of a set of temporal deontic rules can quickly become incomprehensible. Therefore, it is indicated how, under a number of limitations, a set of temporal deontic rules can be visualized in a graph-oriented process modeling language. Furthermore, it is indicated how verification of temporal deontic assignments could be performed.

Bhattacharya et al. (2007) formally describe a so-called artifact-centric process modeling approach, that allows to model processes in terms of the preconditions and effects of services. The state space of their language consists of artifacts. Artifacts are object-oriented data structures with attributes and states. The transition type in the language is the act of invoking a service. Services can create or destroy artifacts and can read and write their attributes. In order to invoke a service on a set of artifacts, the preconditions of the service must be fulfilled. In the spirit of OWL-S and WSMO, artifact-centric modeling also provides for the specification of the effects of services, but the authors do not propose a planning mechanism that reasons about the effect of invoking service in order to obtain a particular goal state. Instead, the system is given dynamism by means of business rules. These business rules are production rules that can either determine the conditions under which an artifact can change its state, or the conditions under which a particular service must be invoked. In addition to business rules, users can also directly invoke services. The modeling approach is in spirit similar to previous languages, but interestingly, the authors also present a number of complexity results concerning reachability of an end state for an artifact class, dead-lock detection, and the detection of redundant attributes.

The idea of declarative business process modeling is also related to the business rules approach (Kardasis and Loucopoulos, 2005; Ross, 2003). By documenting and formalizing business rules, it is hoped that changes to these rules will no longer result in an avalanche of required information system updates and will thus reduce the IT bottleneck when bringing about business changes. Consequently there is a vivid interest among practitioners (Debevoise, 2005) and commercial software vendors in the confluence of business rules and business process modeling. ILOG, for example offers ILOG JRules integration solutions for several existing BPM products and Microsoft has added business rules functionality to BizTalk.

In general, commercial approaches integrate production rule specification with imperative business process modeling. The integration is realized by including explicit calls to a rule engine in the business process model. In BizTalk, for example, such a call is represented as a so-called *decision shape*. On the basis of the information that a process gets back from the rule engine, the process is carried further. Because it is required to specify when and how business rules must be enforced, these these approaches do not belong to the category of declarative process modeling languages characterized in Section 2.2.

2.4 Conclusion

It is a challenge to design information systems that provide flexible, yet compliant support for business processes. Flexibility and compliance require making a tradeoff both at design-time and execution time. To balance flexibility and compliance, the business concerns that govern business process must be made explicit in declarative process models. In this chapter we have given an overview of various characteristics that distinguish between declarative and procedural modeling approaches. Furthermore, we have given an overview of existing languages for declarative process modeling.

Although there already exist many languages for declarative process modeling, these languages all are fundamentally different. They differ in that they can represent different business concerns, and consequently consider different state spaces, transition types, and transition constraints. Furthermore, these language have a different vocabulary, are expressed with different ontology languages, and have different execution semantics.

- **Different business concerns.** Each of these languages only allows to model a subset of the many real-life business concerns that exist in reality. For instance, the ConDec language and the PENELOPE language only allow to express business rules about sequence and timing constraints, i.e. the control flow aspects (Jablonski and Bussler, 1996). Semantic Web Service languages such as OWL-S (The OWL Services Coalition, 2006) and WSMO (Roman et al., 2005), on the other hand, also include some organizational and data model aspects.

- **Different state space.** Because existing languages model different business concerns, they have different conceptions about the state space of a business process. Approaches such as artifact-centric process modeling and semantic web services consider the facts about business concepts to be the only discriminant of process states. The case handling considers both the data object state and the current activity state. The ConDec language perceives the event history (the trace of executed transitions), whereas PENELOPE also includes the system time, in order to take into account due dates.

- **Different transition types.** The aforementioned languages have different usage contexts, such as service-oriented design or a WfMS process model. Therefore they appear to consider very different transition types. However, many of these transition types are different in name only. For instance, there is no real conceptual difference between binding a service request to a web service, and assigning an activity to an agent or worker. Likewise, there is little difference between invoking a service on a web service and starting to perform an activity. A closer look to the considered transition types reveals that in most of the languages only the performance of an activity is considered to be an activity type. As mentioned, declarative languages must consider activities to have a rich life cycle, with many different transition types.

- **Different constraints types.** Even when languages consider a similar state space and transition types, they have different ways of expressing transition constraints. The ConDec language for instance, expresses temporal constraints that (eventually) must hold between activities in a trace, whereas the PENELOPE language discusses the existence of temporal deontic assignments.

- **Different knowledge representation and reasoning paradigms.** Finally, every language uses a different ontology, different ontology language, and different languages for expressing constraints. For instance, the ConDec language of makes use of Linear Temporal Logic (LTL) as underlying paradigm whereas the PENELOPE language makes use of the Event Calculus.

In conclusion, there exist many languages for declarative process modeling, but none of these languages is by itself really suitable for capturing all the aforementioned business concerns. In the next chapter, we introduce a framework for declarative process modeling within which it is possible to position each individual language for declarative process modeling. This framework lays the ontological foundation for the techniques for declarative process modeling and mining discussed in the remaining chapters in this text.

CHAPTER 3

EM-BrA^{2}CE: Unifying Framework for Declarative Process Modeling

From the previous chapter, it can be concluded that there already exist a large number of formal languages that can be identified as declarative approaches to process modeling. Unfortunately, these languages each model only one specific aspect of the many concerns that exist in reality. Moreover, these languages are based on heterogeneous process ontologies (vocabularies) and make use of very different knowledge representation paradigms.

This chapter takes a different approach to declarative process modeling. Rather than defining one formal language, we define a unifying framework for declarative process modeling, within which it is possible to position the aforementioned languages. The framework is called the EM-BrA^{2}CE Framework. EM-BrA^{2}CE stands for 'Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events'. It consists of a formal vocabulary and an execution model, but it is not one language and does not provide model verification techniques, nor process visualization diagrams. Instead, it can be used both as an expressive informal language for documenting business concerns, and as an ontological foundation to compare and develop formal declarative languages. In addition, the framework lays the ontological foundation of the techniques for declarative modeling and mining that are part of the subsequent chapters in this text. The framework adopts the design principles for declarative process modeling, outlined in Section 2.2 of the previous chapter. It is a *unifying* approach, not a unified approach, hereby not claiming to cover all concepts in every process modeling language.

This chapter is structured as follows. Section 3.1 provides an overview of the framework and points out its design principles. Section 3.2 presents the EM-

BrA^{2}CE Vocabulary. This vocabulary is defined as an extension of the SBVR and allows to declaratively refer to the state of a business process. In Section 3.3 an execution model is provided by modeling the dynamics of an activity life cycle through the use of Colored Petri Nets.

3.1 Overview

In this section, we give a brief overview of the EM-BrA^{2}CE Framework by describing its state space, and transition types. Furthermore, we introduce the "payment-after-shipment" order process that is used as a running example in the remainder of this chapter.

3.1.1 State and State Space

Within the framework the concepts 'activity type' and 'activity' have a central position. These concepts relate to each other on a type-instance basis. An activity type corresponds to a process model, whereas an activity corresponds to a process instance. This correspondence honors the design principle that stipulates: "coordination work is work". In particular, the framework does not distinguish between the act of coordinating a process, and the act of performing an atomic activity. Consequently, process instances are considered to be activities in their own right that in turn consist of sub-activities that are to be coordinated.

It is meaningful to associate a state space to an activity type and a particular state to an individual activity. At any moment, an activity is in a particular state that is composed of facts (propositions) about instance-level concepts such as activities, agents, business concepts, and events. These are defined as follows.

- **Activities** (or services) are individual concepts that represent a unit of (coordination) work to be performed by an agent.
- **Agents** (or service providers) are individual concepts that represent an actor or a group of actors who can perform activities.
- **Business concepts** are individual concepts of which the facts can be manipulated in the context of performing activities.
- **Events** are individual concepts that correspond to an instantaneous, discrete state change of a concept in the world.

The EM-BrA^{2}CE Vocabulary, defined in section 3.2, provides a detailed definition of these concepts, their concept types, and fact types.

3.1.2 Transition Types

An activity state transition brings about a state change by changing some of the facts about activities, agents, business concepts, and events of which the current state of the activity is composed. There are twelve activity state transitions: *create*, *schedule*, *assign*, *revoke*, *start*, *addFact*, *removeFact*, *updateFact*, *complete*, *abort*, *skip*, and *redo*. Each state transition type describes a life cycle event of an activity. The *create*, *schedule*, *assign*, and *revoke* transitions represent coordination work that is to be executed by a coordinator agent as part of constructing an execution plan. The *start*, *addFact*, *removeFact* and *updateFact* transitions represent the actual work that is to be executed by a worker agent. The *skip*, *abort* and *redo* transitions represent the coordination work related to exception handling. The framework associates two event type to each transition type: a positive event type that indicates that the transition has taken place, and a negative event type that indicates that a transition was prevented from taking place. The transition types are defined as follows:

- $create(A, AT, B, P, Coordinator)$: the request to create a new activity A of type AT with business identifiers B, parent activity P by an agent $Coordinator$. Activity event types: *created*, *createRejected*.
- $schedule(A, DueDate, Coordinator)$: the request to set the due date of activity A to $DueDate$ by an agent $Coordinator$. Activity event types: *scheduled*, *scheduleRejected*.
- $assign(A, Agent, Coordinator)$, $revoke(A, Agent, Coordinator)$: the request to assign or revoke the activity A to an agent $Agent$ by an agent $Coordinator$. Activity event types: *assigned*, *assignRejected*, *revoke*, *revokeRejected*.
- $start(A, Worker)$: requests an activity A to start by an agent $Worker$. Activity event types: *started*, *startRejected*.
- $addFact(A, F, Worker)$, $removeFact(A, F, Worker)$, $updateFact(A, F_1, F_2, Worker)$: the request to add, or remove business fact F or update a business fact F_1 by F_2 within the context of activity A by an agent $Worker$. Activity event types: *factAdded*, *factUpdated*, *factRemoved*, *addFactRejected*, *updateFactRejected*, *removeFactRejected*.
- $complete(A, Worker)$: the request to complete an activity A by an agent $Worker$. Activity event types: *completed*, *completeRejected*. Upon completion of an activity, all business fact manipulations are committed to change the globally visible business facts.
- $skip(A, Coordinator)$, $abort(A, Coordinator)$, $redo(A, Coordinator)$: the request to skip, abort or redo an activity A by an agent $Coordinator$. Activity event types: *skipped*, *skipRejected*, *aborted*, *abortRejected*, *redone*, *redoRejected*.

Because they are generic, these twelve activity state transitions provide a means of defining an **execution model**. This is done in section 3.3. The current state of an activity determines which state transitions can occur. These state transitions might be subject to business and non-business concerns. The Petri net of Figure 3.1 models the allowable sequences of transitions in the activity life cycle as imposed by non-business concerns.

The only way an agent (or service provider) can change the state of the world is by invoking one of these twelve state transitions on an activity (or service). An agent must be assigned to an activity, in order to be able to invoke a transition on this activity. Likewise, agents (or service providers) can only perceive facts when assigned to an activity. Only after starting an activity, the business facts that pertain to its state can be manipulated. Only after completing an activity, these manipulations are propagated to the outside world.

Business concerns can also constrain the possible trajectories within the state space of an activity type. Informally, it suffices to check prior to the occurrence of a state transition of a particular type whether a number of particular business concerns will be violated or not. When no business concern is violated, the state transition can take place. When, on the other hand, the transition would lead to an intolerable violation of a business concern, the state transition is prevented from taking place. At this point, a negative event might be recorded.

3.1.3 Running Example

To illustrate declarative process modeling, an order business process will be used as a running example throughout the text. The order process depicts a "payment-after-shipment" trade policy and is represented using the Business Process Modeling Notation (BPMN) Object Management Group (2006b) in Figure 3.2. The process can be declaratively modeled as follows:

- **state space**: the state space of the order-to-cash process is described by facts about
  - **roles**: buyer and seller.
  - **composite activity types**: coordinate purchase order, coordinate sales order.
  - **atomic activity types**: Coordinate purchase order *can consist of* place order and pay activities. Coordinate sales order *can consist of* accept order, reject order, and ship activities.
  - **activity event types**: created, assigned, started, completed
  - **event types**: timeout, obligation violated
  - **business concepts**: order, order line
  - **business fact types**: order *has* order line, order *is critical*, order *has* due date, order *has* discount, order *has* customer, customer *is* loyal customer, customer *is* corporate customer,... Place order can manipulate the business fact types 'order has order line' and 'order has due date',...
- **rules**:
  - "Initially a buyer has the permission to perform a place order activity."

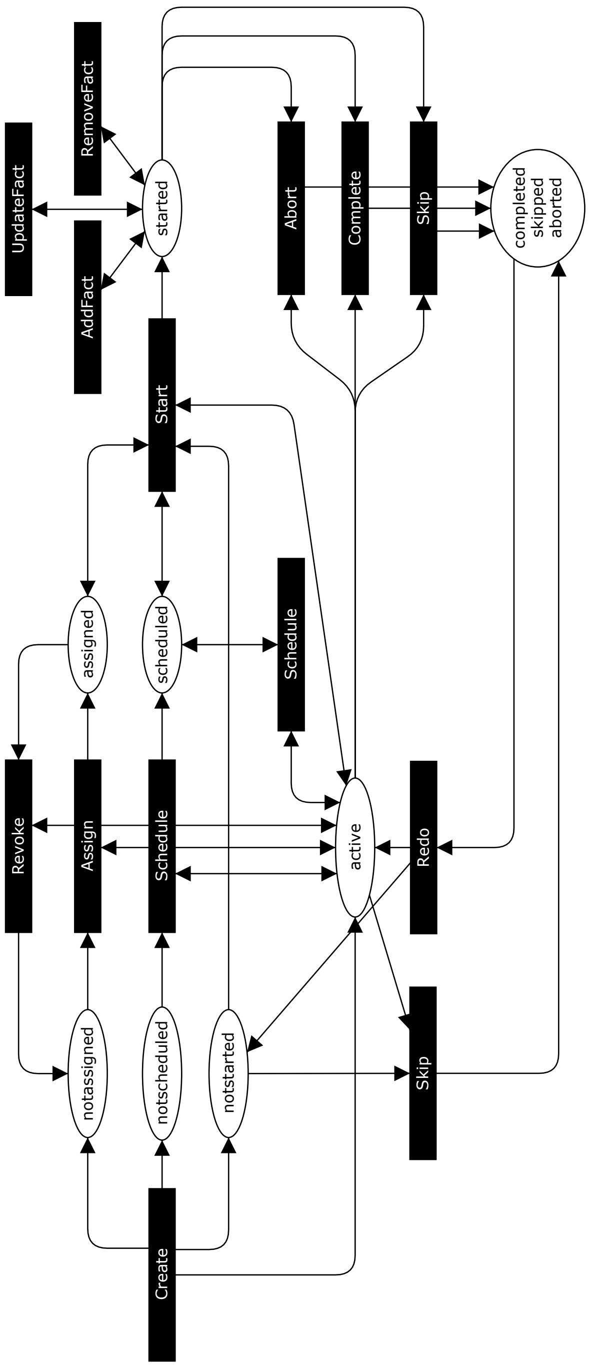


Figure 3.1: A CP-Net model of the the transition types in the EM-BrA^{2}CE execution model

- "When a buyer completes a place order activity, the seller has the obligation to perform a accept order activity or a reject order activity within 2 time units."
- "When the buyer completes a place order activity, the buyer has the obligation to perform a pay activity within 2 time units after the seller completes the ship activity."
- "When the seller completes a accept order activity, the seller has the obligation to perform a ship activity within 2 time units."
- " There exists exactly one place order activity that has parent a handle purchase order activity."
- "There exists at most one accept order activity that has parent a handle sales order activity."
- "Activities that have type place order, accept order, reject order and ship order must not be performed in parallel."
- "Accept order and reject order activities are mutually exclusive."
- "Accept order and ship activities are mutually inclusive."
- "After the start of a ship activity, the order lines of the order can no longer be changed."
- "Each order has at least one order line."
- "The agreed price of a sales item is less or equal to the standard price of the sales item."
- "An order has a 10 percent discount if the order is from a loyal customer."
- "An agent that has age less than 18 years can not perform a place order activity."
- "An agent that has function junior sales representative can not perform an accept order or reject order activity that is identified by an order that has an amount larger than 2000 euro."
- "Coordinate purchase order can make visible the business fact type 'order *has* rejection notice." "It is necessary that a rejection notice is only visible to an agent that is a corporate customer."
- "A buyer can subscribe to completed in the context of ship." "It is not possible that an agent that has role buyer perceives an event that is about a ship activity for an order that has a total amount of less than 2000 euro."

3.2 The EM-BrA^{2}CE Vocabulary

In this section, we define a unifying foundational vocabulary for declarative process modeling. Foundational vocabularies provide the conceptual building blocks upon which domain-specific vocabularies are built. In general, one can distinguish a layered hierarchy of modeling levels, such as the one displayed in Table 3.1. The displayed hierarchy distinguishes between real-world concepts, data instances, domain-specific vocabularies, foundational vocabularies, and ontology

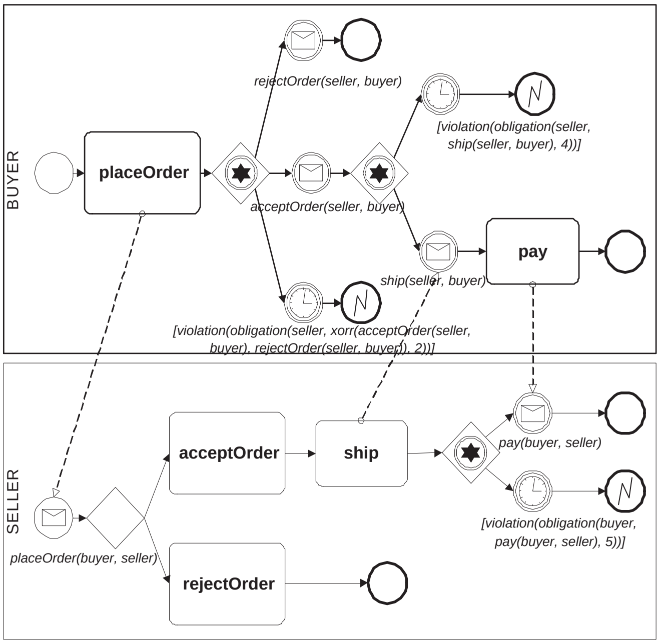

BUYER
placeOrder
rejectOrder(seller, buyer)
[violation(obligation(seller, ship(seller, buyer), 4))]
acceptOrder(seller, buyer)
pay
ship(seller, buyer)
[violation(obligation(seller, xorr(acceptOrder(seller, buyer), rejectOrder(seller, buyer)), 2))]
SELLER
acceptOrder
ship
pay(buyer, seller)
[violation(obligation(buyer, pay(buyer, seller), 5))]
placeOrder(buyer, seller)
rejectOrder

Figure 3.2: Payment-after-shipment

languages. In the top-four layers in this meta-modeling infrastructure there exists instantiation-classification relationships between the concepts of subsequent layers, although there can also exist instantiation-classification relationships within a single modeling layer (Atkinson and Kühne, 2003).

Table 3.1: Traditional layering of ontologies

modeling level	**definition and example**
meta-metamodel or ontology language	A language to represent ontologies. For example, OWL, MOF/UML, the SBVR Meaning and Representation Vocabulary.
metamodel or foundational vocabulary	The conceptual building blocks upon which domain-specific vocabularies are built. For example, the UML, the SBVR Vocabulary for Describing Business Vocabularies.
model or domain-specific vocabulary	A domain model of real-world concepts. For example a UML class diagram or an SBVR business vocabulary.
data or instance	A logical representation of a real-world concept. For example, purchase order data.
real-world concept	For example, a purchase order.

Foundational vocabularies for process modeling provide conceptual building blocks for modeling business processes. In the literature there are many foundational vocabularies for process modeling and enterprise modeling. These are all different, but they share a common tradition of conceptual modeling. In this section, we define a new foundational vocabulary for declarative process modeling that is called the EM-BrA^{2}CE Vocabulary. EM-BrA^{2}CE stands for 'Enterprise Modeling with Business Rules, Agents, Activities, Concepts and Events'. The acronym indicates the main building blocks of the language. To a large extent, these building blocks also appear as synonyms or homonyms in many other foundational ontologies for process modeling. In particular, the EM-BrA^{2}CE Vocabulary adopts a selection of concepts within the following foundational vocabularies:

- The reference model and workflow glossary of the Workflow Management Coalition (WfMC) (1995, 1999).
- The Semantics of Business Vocabulary and Rules (SBVR) initiative for conceptual modeling and business modeling issued as an Object Management Group (2008) standard.
- The Web Service Modeling Ontology (WSMO) (Roman et al., 2005) which is a foundational ontology for representing semantic web services developed by the Digital Enterprise Research Institute (DERI).
- The Role-Based Access Control (RBAC) which is a flexible access control model developed by Ferraiolo et al. (2001); Sandhu et al. (1996) and standardized with the InterNational Committee for Information Technology Standards (INCITS) (2004).
- The Agent-Object-Relationship (AOR) modeling approach of Wagner (2003) that distinguishes agents as entities that can have deontic assignments and that can consist of other agents.

In the definitions of the EM-BrA^{2}CE Vocabulary, we make reference to the origin of adopted concepts and indicate potential differences. The aim of the vocabulary is to have a maximal expressiveness while introducing only a minimal number of concepts. Therefore, we have attempted not to differentiate too much by introducing too many specializations of concepts.

This section is structured as follows. First, we motivate why the SBVR was chosen as ontology language. Subsequently, an introduction to the SBVR Meaning and Representation Vocabulary is provided. In the next section, the EM-BrA^{2}CE Vocabulary is defined that extends the SBVR Meaning and Representation Vocabulary. In particular, building blocks such as business concept, activity, state, agent, event, and deontic assignment are defined.

3.2.1 Candidate Ontology Languages

There exist several candidate ontology languages that can be used to define a foundational vocabulary for declarative process modeling. Among the most prominent candidates are first-order logic, the tandem Meta Object Facility(MOF) / Unified Modeling Language (UML) (Object Management Group, 2006c), and the Web Ontology Language (OWL). From these candidates we choose to model the meta-model in terms of the vocabularies provided by the Semantics of Business Rules and Vocabulary (SBVR) language (Chapin, 2005; Object Management Group, 2008). The SBVR was chosen because it possesses many desired properties:

- **Model granularity.** Information models can use different levels of granularity to represent concepts in the world. In the last decade two paradigms have emerged: object-level and fact-level granularity. Fact-orientation perceives the world in terms of facts rather than in terms of objects, attributes and relationships. Fact types have a finer granularity compared to object types. This facilitates the expression of business rules (Halpin, 2000) and postpones implementation decisions about grouping attribute and relationship types into object types (Halpin, 1991; Leung and Nijssen, 1988).

- **Local Closure.** In knowledge representation one has to deal with incomplete knowledge of the world. In SBVR it is possible to indicate the predicates (fact types) over which the model has complete knowledge. Such a construct is called *local closure* and it is possible to indicate local closure in SBVR. In general, two assumptions are possible: an open-world and a closed-word assumption (Brachman and Levesque, 2004). Under an open-world assumption (OWA) it is accepted that a model incompletely represents the world. Under a closed-world assumption (CWA) it is assumed that the model completely represents the world. Both assumptions lead to a different semantics, for instance when reasoning with negation (Wagner, 1991). In the SBVR, it is possible to indicate the concepts and fact types over which the model represents complete information, thus enabling local closure.

- **Business rules as natural language expressions.** As business rules are most often formulated as (natural) language statements, the SBVR contains an English vocabulary for describing vocabularies and stating rules and a vocabulary to express the meaning of (natural) language expressions in terms of formal logic. The processing of natural language pertains to the Artificial Intelligence domain of Natural Language Processing (NLP). It involves on the one hand the understanding of natural language statements in terms of semantic formulations (Baisley et al., 2005), and on the other hand the verbalization of semantic formulations into natural language statements. To date, two SBVR natural language parsers have been developed: the Unisys Rules Modeler (Baisley, 2005; Unisys, 2005) and SBeaVeR (Digital Business Ecosystem (DBE), 2007). These parsers analyze the meaning of natural language expressions in terms of semantic formulations.

- **Rule modality.** One of the characteristics of declarative process models is that they make a distinction between business rules that cannot be violated, rules that can be violated and guidelines. The current SBVR specification requires business rules to be either a necessity, an obligation, a prohibition or a possibility.

- **Reification.** The SBVR allows propositions to be treated as concepts in their own right. As such, propositions can be made about other propositions. This is called 'objectification' in the standard. Objectification is, for instance, useful to represent that a particular business fact has been asserted or retracted in the context of a given activity.

The SBVR has many features that make it an attractive languages for declarative process modeling. Nonetheless the SBVR does also have its shortcomings.

- **No temporal logic.** The semantics of SBVR expressions is underpinned by first-order logic, Simple Deontic Logic, restricted Higher-Order logic and reification. Although Structured English provides two linguistic techniques to express temporal relationships: objectification and intensional roles (Object Management Group, 2008, p. 47, p. 243), it lacks a temporal logic to represent and reason about temporal relationships. The inclusion of temporal logic is deferred to a later version of SBVR (Object Management Group, 2008, p. 108). In a dynamic world of business processes, such knowledge representation and reasoning mechanisms are required to reason about properties qualified in terms of time or the effect of activities on the state of the world.

- **No default logic.** Furthermore, the SBVR lacks the semantics to model business rules in terms of a number of general rules and exceptions. Such a means for representing and reasoning with default knowledge is, for instance, provided by defeasible logic (Antoniou et al., 2001; Nute, 1994). This way of knowledge representation is valuable, because it facilitates the incremental specification of business rules (Grosof et al., 1999): new rules can be added without the conditions of previous rules need to be reconsidered. Normal, non-defeasible rules, in contrast, require a complete, encyclopedic knowledge of all rules to be updated or decided upon.

- **Complexity of natural language formulations.** The current SBVR specification imposes a restricted grammar to natural language formulations that often turn out to be less understandable than a logical expression in a common first-order syntax.

3.2.2 SBVR as Ontology Language

The Semantics of Business Vocabulary and Business Rules (SBVR) provides a number of conceptual vocabularies for modeling any domain – including itself – in the form of a vocabulary and a set of rules. As the EM-BrA^{2}CE vocabulary extends the conceptual vocabularies of the SBVR, these vocabularies will be discussed in the remainder of this section.

In SBVR, meaning is kept separate from expression. As a consequence, the same meaning can be expressed in different ways. In real-life, meaning is more often expressed in textual form than in diagrams as statements provide more flexibility in defining vocabulary and expressing rules. For these reasons, the SBVR specification defines a structured, English vocabulary for describing vocabularies and verbalizing rules, called SBVR Structured English (Object Management Group, 2008, p. 237). One of the techniques used by SBVR structured English are font styles to designate statements with formal meaning. In particular,

- the term font is used to designate a noun concept.
- the name font designates an individual concept.
- the *verb* font is used for designation for a verb concept.
- the **keyword** font is used for linguistic particles that are used to construct statements.

The definitions and examples in the remainder of the text use these SBVR Structured English font styles.

In SBVR a vocabulary and a set of rules make up a so called conceptual schema. A conceptual schema with an additional set of facts that adheres to the schema is called a conceptual model. Figure 3.3 depicts the relationship of a conceptual schema and a conceptual model to some of the core building blocks in SBVR. These core building blocks are part of the SBVR Meaning and Representation Vocabulary. This vocabulary contains among others the following definitions (Object Management Group, 2008, p. 37):

> A conceptual schema *is* a combination of concepts and facts (with semantic formulations that define them) of what is possible, necessary, permissible, and obligatory in each possible world.
> A conceptual model or fact model *is* a combination of a conceptual schema and, for one possible world, a set of facts (defined by semantic formulations using only the concepts of the conceptual schema).

The facts in a conceptual model may cover any period of time. Changing the facts in a conceptual model creates a new and different conceptual model. In this way the SBVR gives conceptual models a monotonic semantics (Object Management Group, 2008, p. 91).

Informally speaking, the nouns and verbs that occur within a particular vocabulary can be related to noun concepts (or object types) and verb concepts (or fact types). In natural language, the grammar of a basic sentence can be seen as a subject-verb-object triple. Just as verbs can have the roles of subject and object in a sentence, verb concepts can have roles that refer to noun concepts playing a part, assuming a function or being used in some situation. In the SBVR Meaning and Representation Vocabulary, depicted in Figure 3.4, these concepts are formally defined as follows.

> A meaning represents what is meant by a word, sign, statement, or description; what someone intends to express or what someone understands.
> A concept *is* a meaning that represents a unit of knowledge created by a unique combination of characteristics.

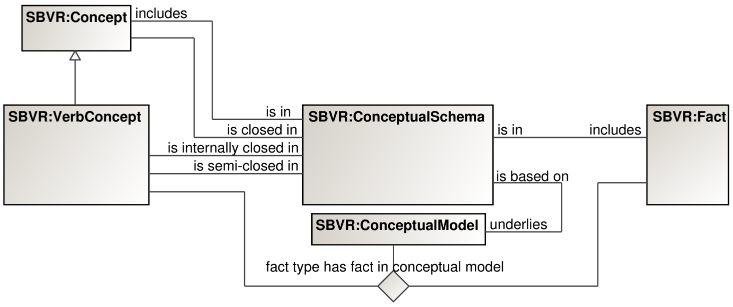


Figure 3.3: A MOF/UML representation of SBVR conceptual schema and model (Object Management Group, 2008)

> A verb concept or fact type *is* a concept whose instances are all actualities and that is a basis for atomic formulation, having at least one role. Concept type: concept type.
> A noun concept *is* a concept that is not a verb concept. Concept type: concept type.
> An individual concept *is* a concept that corresponds to only one thing. General concept: noun concept. Concept type: concept type.
> A role *is* a noun concept that corresponds to things based on their playing a part, assuming a function or being used in some situation. Necessity: each role *is of* at most one fact type. 'Verb concept *has* role' *is* an abstraction of a thing playing a part in instances of the fact type. Concept type: concept type.
> A concept type *is* a noun concept that *specializes* the concept 'concept'. 'Concept$_1$ *specializes* concept$_2$' the concept$_1$ incorporates each characteristic incorporated into the concept$_2$ plus at least one differentiator. This represents the specialization-generalization relationship.
> An SBVR:proposition *is* a meaning that is asserted when a sentence is uttered or inscribed and which is true or false.
> An SBVR:fact *is* a proposition that is taken as true.

Although concepts have a particular meaning, by themselves they do not constitute any statement about what is true, possible, necessary, permissible, and obligatory in a possible world. Such statements can be expressed by means of rules. The SBVR Vocabulary for Describing Business Rules, depicted in Figure 3.5, contains among others the following abbreviated definitions.

> A business policy *is* a directive that *is* not *actionable* whose purpose is to guide an enterprise.
> A rule *is* an *actionable* directive that introduces an obligation or a necessity.
> A business rule *is* a rule that is under business jurisdiction. 'business rule *is derived from* business policy' represents the business policy from which

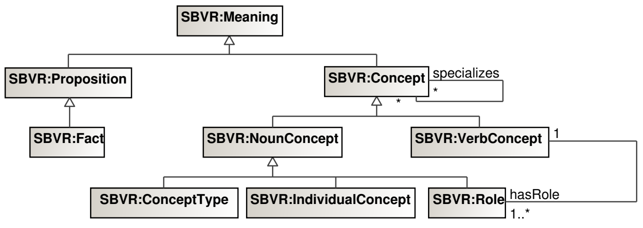


Figure 3.4: A MOF/UML representation of the SBVR Meaning and Representation Vocabulary

a business rule originates.
A structural (business) rule *is* a (business) rule that is intended as a definitional criterion. A structural rule expresses a necessity that cannot be violated.
An operative business rule *is* a business rule that is intended to produce an appropriate or designed effect. An operative business rule expresses an obligation that can be violated.
A level of enforcement is something that represents a position in a graded or ordered scale of values that specifies the severity of action imposed in order to put or keep an operative business rule in force. 'operative business rule *has* level of enforcement' the level of enforcement that a particular operative business rules has.

The SBVR defines a business rule as a rule under business jurisdiction that is derived from a business policy. This definition can be seen as too limited because very often rules are imposed on organizations by a third party. On the other hand, imposed rules always have to be internalized and in that regard the definition remains useful. A salient feature is to assign a level of enforcement to an operative business rule expressing an obligation or a prohibition. In this way a less crisp distinction can be made between strict business rules (hard constraints) and guidelines (soft constraints).

In SBVR, meaning remains separate from expression. The SBVR provides a vocabulary called the Logical Formulation of Semantics Vocabulary to describe the structure and the meaning of vocabulary and business rules in terms of formalized statements about the meaning. Such formalized statements are semantic formulations (Baisley et al., 2005). Besides these fundamental vocabularies, the SBVR provides a discussion of its semantics in terms of existing, well-established formal logics such as first-order logic, deontic logic, and higher-order logic.

3.2.3 The EM-BrA^{2}CE Vocabulary

Although the SBVR provides extensive vocabularies for expressing business vocabularies and business rules, the current SBVR specification (Object Manage-

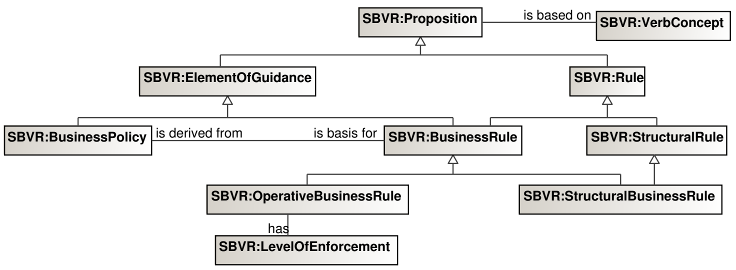


Figure 3.5: A MOF/UML representation of the SBVR Vocabulary for Describing Business Rules

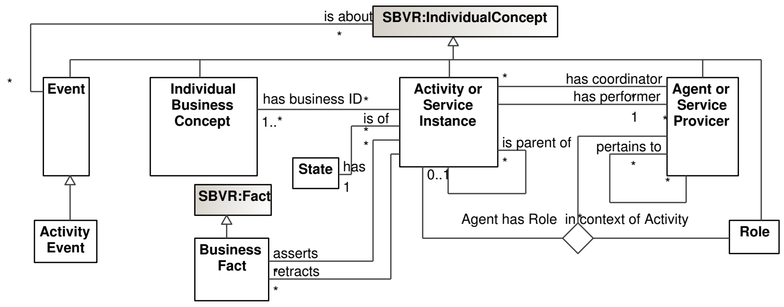
ment Group, 2008) does not have a built-in vocabulary for expressing process-related concepts such as agent, activity, event, or deontic assignment. These concepts are introduced by the EM-BrA^{2}CE Vocabulary.

The EM-BrA^{2}CE Vocabulary defines instance-level concepts that are meant for describing the state of a business process instance. In addition, it defines type-level concepts that are meant for describing the state space of a business process model. In conceptual modeling, it often occurs that instances of types that are types themselves (Atkinson and Kühne, 2003; Halpin, 2004). This paradigm is known as higher-order typing. In UML, higher-order typing can be obtained using the UML stereotype mechanism (Atkinson and Kühne, 2003) or using UML powertypes (Object Management Group, 2005). Figure 3.6(a) is a MOF/UML class diagram representation of the instance-level concepts in the vocabulary. Likewise Figure 3.6(b) represents the type-level concepts. Whereas all instance-level concepts extend SBVR:individual concept, all type-level concepts extend SBVR:concept type. To each instance-level individual concept a particular type-level concept type corresponds. In the following paragraphs these type-instance pairs are defined.

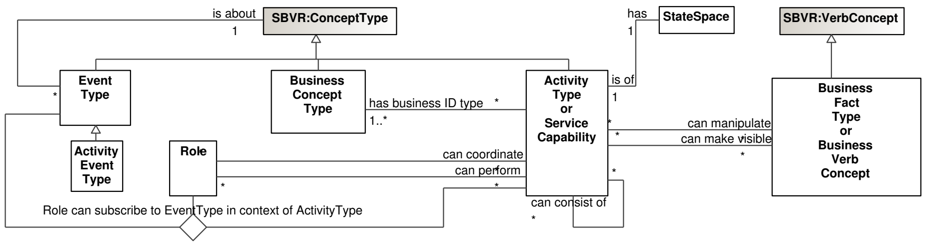
Business Concept – business concept type

The flexibility of declarative business process modeling comes, among others, from the under-specification of process models and the use of guidelines (soft constraints). It does, however, not come from run-time adaptability of the process model. Therefore, the vocabulary distinguishes fact types that can be manipulated in the context of an activity, called business fact types. The following definitions apply.

A business concept type *is* an SBVR:concept type that *specializes* the individual concept 'individual business concept' and that *classifies* an individual business concept. Example: the business concept type 'purchase order'.
An individual business concept *is* an SBVR:individual concept of which



(a) instance-level

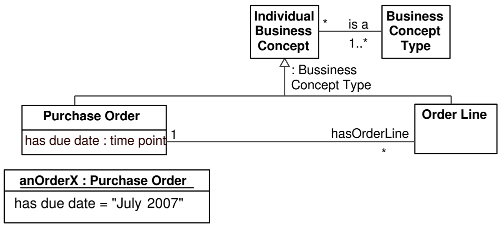


(b) type-level

Figure 3.6: A MOF/UML representation of the EM-BrA^{2}CE Vocabulary

the facts can be manipulated in the context of an activity. Concept type: business concept type.
'individual business concept *is a* business concept type' *is* an SBVR:assortment fact type that categorizes a business concept as being of a particular business concept type. Example: the individual business concept 'anOrderX', the fact 'anOrderX *is a* purchase order'.
A business fact type *is* an SBVR:fact type that has only business concept types as SBVR:role.
Example: the business fact type 'purchase order *has due date* time point'.
A business fact *is* an SBVR:fact that is the basis for an atomic formulation of which every role binding *is bound to* a business concept. Concept type: business fact type.
Example: the business fact 'anOrderX *has due date* July 2007'.

As mentioned earlier, the vocabulary contains both type-level concepts and instance-level concepts. A business concept type, for instance, is a type-level concept. It is a classification type of individual business concepts. This means that every instance of a business concept type is a sub-class of individual business concept. In the semantics of UML 2.0, business concept type can be thought of as a powertype. This is represented in Figure 3.7.



Individual Business Concept
*
is a
1..*
Business Concept Type
: Bussiness Concept Type
Purchase Order
has due date : time point
1
hasOrderLine
*
Order Line
anOrderX : Purchase Order
has due date = "July 2007"

Figure 3.7: A UML 2.0 powertype representation of business concept type

Activity – activity type

The pair activity – activity type represents two of the most central concepts in the vocabulary. The following definitions apply:

An activity type or service capability *is* an SBVR:concept type that *specializes* the individual concept 'activity' and that *classifies* an activity.
Example: the activity type 'place order'.
An activity or service instance *is* an SBVR:individual concept that represents a unit of (coordination) work to be performed by an agent. Example: the activity 'anActivityX'.
'activity *has type* activity type' *is* an SBVR:assortment fact type that categorizes an activity as being of a given activity type. Necessity: each

activity *has type* exactly one activity type. Example: anActivityX *has type* coordinate purchase order.

The design principle "coordination work is work" is recognized in the definitions of activity: an activity can either represent the act of performing an atomic unit of work or the act of coordinating a set of sub-activities. The former activity is called an atomic activity whereas the latter activity is called a composite activity. The fact types '*can consist of*' and '*is parent of*' indicate the activities a composite activity can consist of.

'Activity type$_1$ *can consist of* activity type$_2$' *is* an SBVR:partitive fact type that represents that an activity of activity type$_1$ involves the coordination of activities of activity type$_2$.
'Activity$_1$ *is parent of* activity$_2$' *is* an SBVR:partitive fact type that represents an activity$_2$ being composed of an activity$_1$. Example: the fact 'anActivityX *is parent of* anActivityY'.
A composite activity type *is* an activity type that describes a category of composite activities. Example: the composite activity type 'coordinate purchase order'. Necessity: A composite activity type *can consist of* at least one activity type.
An atomic activity type *is* an activity type that describes a category of atomic activities. Example: the atomic activity type 'place order', the fact type 'coordinate purchase order *can consist of* place order'.
A composite activity *is* an activity that represents the coordination of a number of activities.
An atomic activity *is* an activity that is not a composite activity and that represents an elementary unit of work. Necessity: an atomic activity *is* not *parent of* an activity.

When performing coordination work an agent can create an execution plan that consists of a number of sub-activities. In that case, the agent is identified as the coordinator of the created sub-activities. This is expressed by the *has coordinator* verb concept. The coordinator can schedule each activity in the execution plan for a particular due date, as expressed by the *has scheduled due date* fact type. The *has coordinator* fact type, is set by the coordinator when he assigns a given activity in the execution plan to a particular agent.

'Activity *has coordinator* agent' *is* an SBVR:associative fact type that represents an agent coordinating an activity.
'Activity *has scheduled due date* time point' is an SBVR:is-property-of fact type that represents the scheduled due date of an activity.
'Activity *has performer* agent' *is* an SBVR:associative fact type that represents an agent performing an activity. Necessity: an activity *has performer* exactly one agent. Note: The latter constraint is not restrictive, since agents can form (ad-hoc) groups that are also agents.

An activity is uniquely **identified** by a set of business concepts. For example, a business concept of business concept type purchase order uniquely identifies an activity of type coordinate purchase order. This is expressed by the *has business ID* fact type. Another way of looking at business identifies is that they are the object on which an agent performs an activity. Consequently, the *has object* fact type is a synonym for the *has business ID* fact type and sets the business context of a given activity.

> 'Activity type *has business ID type* business concept type' *is* an SBVR:associative fact type that represents the business concept types that can identify an activity type. Example: coordinate purchase order *has business ID type* purchase order.
> 'Activity *has business ID* business concept' *is* an SBVR:associative fact type that represents an activity being (partially) identified by the business concept. Synonym: 'Activity *has object* business concept type' Example: anActivityX *has business ID* anOrderX or anActivityX *has object* anOrderX.

When performing an activity of a particular activity type, an agent can manipulate business facts of particular business fact types. This is expressed by the 'activity type *can manipulate* business fact type' fact type. Additionally, agents can retrieve information about particular business fact types when performing activities. The business fact types that are visible are indicated by the 'Activity type *can make visible* business fact type' fact type.

> 'Activity type *can manipulate* business fact type' *is* an SBVR:associative fact type that represents that a business fact of type business fact type can be asserted or retracted during the performance of an activity of type activity type. Example: place order *can manipulate* the business fact type 'purchase order *has due date* time point'.
> 'Activity type *can make visible* business fact type' *is* an SBVR:associative fact type that represents the business fact types that can be made visible in the context of activities of activity type. Note: visibility can be restricted by a visibility constraint. Example: coordinate purchase order *can make visible* the business fact type 'purchase order *has due date* time point'.

Within the context of an activity, a worker can perceive and manipulate only those business facts in which the business ID has a role. When an agent does business fact manipulations during the performance of an activity, the result of these manipulatations is temporarily reflected by the *asserts* and *retracts* verb concepts. Only upon completion of the activity, the concept manipulations are committed to the entire system, i.e. the outside world. Section 3.3 explains the EM-BrA^{2}CE execution model and discusses this mechanism in detail.

> 'Activity *asserts* business fact' *is* an SBVR:associative fact type that represents a business fact has been asserted in the context of the activity. Example: anActivityY *asserts* the business fact 'anOrderX *has due date* Juli 2007'.

‘Activity *retracts* business fact’ *is* an SBVR:associative fact type that represents a business fact has been retracted in the context of the activity.

State – state space

An activity type (business process model) can be modeled by describing a state space and a set of business rules that constrain the possible transitions in this state space. Consequently, an activity (business process instance) has a particular state that corresponds to a specific set of facts that are true in this state.

A state space *is* an SBVR:conceptual schema that *includes* the SBVR:concepts that describe a set of discrete states of an activity type. Necessity: a state space can only contain concepts that are instances of the concepts defined in the EM-BrA^{2}CE Vocabulary.
‘activity type *has* state space’ *is* an SBVR:associative fact type that represents the state space of an activity. Necessity: an activity type *has* exactly one state space.
A state *is* an SBVR:conceptual model that *includes* facts about the concepts in the state space, that corresponds to a specific situation of an activity and that *is based on* the state space of an activity type.
‘activity *has* state’ *is* an SBVR:associative fact type that represents the state of an activity. Necessity: an activity *has* exactly one state.

State space is a specialization of an SBVR:conceptual schema, as depicted in Figure 3.8. Like a conceptual schema, a state space is described by the concepts, fact types and facts that adhere to the state space. As such, a state space describes a potentially infinite number of states. Likewise, state is a specialization of SBVR:conceptual model. Each state is based on a state space and contains a number of facts that adhere to the fact types in that state space.

In natural language, state is most often a relative notion that consists of a subgroup of states. For example, when defining the goal state of a business process, it is useful to consider the notion of an abstract state. Goal state is a concept that also occurs in WSMO.

An abstract state *is* a set of states that *conform to* the abstract state and that *is based on* the state space of an activity type.
‘state *conforms to* abstract state’ *is* an SBVR:associative fact type that a state corresponds to an abstract state.
‘state space *has goal state* abstract state’ *is* an SBVR:associative fact type that represents an abstract state being a goal or end state of a state space.
‘state space *has start state* abstract state’ is an SBVR:associative fact type that represents an abstract state being the start state of a state space.

An SBVR:business rule can be seen as a statement about an abstract state being either a necessity, an obligation, a prohibition or a possibility.

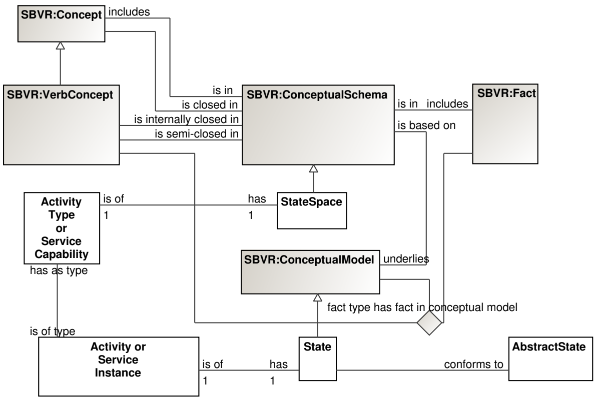


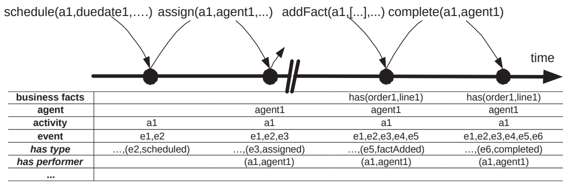
Figure 3.8: A MOF/UML representation of state and state space in the EM-BrA^{2}CE Vocabulary

A *logic system* is called **monotonic** when the set of ground facts and logical formula in the system can produce a set of consequences that monotonically increases, even when new logical axioms are added. Logics with this property, namely that a derived fact cannot be invalidated by the addition of a logical formula that is consistent with this fact, are called **monotonic logics**. Conversely, a logic is non-monotonic when the addition of a logical formula can produce a reduction of the set of consequences that can be derived from it (Brachman and Levesque, 2004). A classical example of a non-monotonic system is Prolog, as its negation-as-failure entails that the addition of a fact might entail falsity of a previously derived fact.

In the EM-BrA^{2}CE Framework (composite) activities represent the (coordination) work that occurs in the context of business processes. When an activity state transition occurs, a business process instance enters a new state and the transition is recorded by an activity event. Furthermore, agents can manipulate business facts in the context of an activity. Such a system that allows the manipulation of business facts could be interpreted as non-monotonic. However, the solution of the SBVR can be adopted that considers each conceptual model, consisting of a conceptual schema and collection of facts, as a logical system in its own right. Each time the facts in a conceptual model are changed, this creates a new and different conceptual model. In this way conceptual models are given a monotonic semantics (Object Management Group, 2008, p. 77). This solution also conserves monotonicity when using negation-as-failure.

Figure 3.9 illustrates a number of state transitions that occur to a place order activity a1. Each state transition results in a new set of concepts and ground facts, and thus a new state, that is partially represented in the columns of the figure. As each new activity state is considered to be a new SBVR:conceptual model, deductive reasoning can use a monotonic reasoning paradigm (Object

Management Group, 2008, p. 77).



business facts			has(order1,line1)	has(order1,line1)
agent		agent1	agent1	agent1
activity	a1	a1	a1	a1
event	e1,e2	e1,e2,e3	e1,e2,e3,e4,e5	e1,e2,e3,e4,e5,e6
has type	…,(e2,scheduled)	…,(e3,assigned)	…,(e5,factAdded)	…,(e6,completed)
has performer		(a1,agent1)	(a1,agent1)	(a1,agent1)
...				

Figure 3.9: An illustration of the state transitions of a place order activity a1

Agent – role

Business Process Management Systems (BPMSs) must support business processes in which both humans and machines perform (coordination) work. To this end it is useful make abstraction from the differences between humans and machines through the use of the **agent** metaphor. This agent metaphor is present in many other ontologies for business modeling (Guizzardi and Wagner, 2005; Wagner, 2003). In the vocabulary, the agent concept in the vocabulary does not only represent individual workers or machines, but also ad-hoc groups of agents, such as for instance an entire department or company. This is expressed with the *pertains to* fact type. This construct is grounded in the holonic agent metaphor of holonic muli-agent systems (holoMAS) (Weiss, 1999).

> An agent or service provider *is* an SBVR:individual concept that represents an actor or a group of actors who can perform activities. Example: the agents ‘workerX’, ‘purchase department’, ‘buyer inc.’.
> ‘Agent$_1$ *pertains to* agent$_2$’ *is* an SBVR:partitive fact type that represents organizational structure and ad-hoc groups of agents. Example: the facts ‘workerX *pertains to* purchase department’,‘purchase department *pertains to* buyer inc.’. Note: the ‘*pertains to*’ fact type is transitive.

In the context of a business process an agent can fulfill a particular role that represents an authorization to perform a number of activities. This conception of role is consistent with the Role Based Access Control (RBAC) standard (Ferraiolo et al., 2001; InterNational Committee for Information Technology Standards (INCITS), 2004; Sandhu et al., 1996). In the vocabulary the following definition applies:

> A role *is* an SBVR:individual concept that represents a set of authorizations with regard to the performance of activities of given activity types.

Agents that have a particular role in the context of a business process have the authorization to perform a particular activity. This authorization is expressed by the ‘Role *can perform* activity type’ fact type. When performing an activity of a particular activity type, an agent can manipulate business facts of particular business fact types. This is expressed by the ‘activity type *can manipulate* business fact type' fact type. Additionally, agents can retrieve information about particular business fact types when performing activities. The business fact types that are visible are indicated by the 'activity type *can make visible* business fact type' fact type. Business rules can constrain the ability of agents to perceive or manipulate business facts, in spite of their role assignments. This is discussed in section 4.1.

> 'Role *can perform* activity type' *is* an SBVR:associative fact type that *represents* that an agent that has a given role can perform an activity of a particular activity type.
> 'Role *can coordinate* activity type' *is* an SBVR:associative fact type that *represents* the authorization that an agent of a particular role can coordinate an activity of a particular activity type.
> 'Agent *can have role* role' *is* an SBVR:associative fact type that represents that an agent can assume a particular role.
> 'Agent *has role* role *in the context of* activity' *is* an SBVR:associative fact type that represents that an agent assumes a particular role in the context of an activity. Note: These authorizations can be restricted by an activity authorization constraint.

The EM-BrA^{2}CE execution model distinguishes activity state transitions related to coordination (*create*, *schedule*, *assign*, *revoke*) and state transitions related to performing actual work (*start*, *addFact*, *removeFact*, *updateFact*, *complete*). Consequently, the vocabulary makes a distinction between the coordinator and the performer of an activity. The activity hierarchy determines whether an agent can coordinate an activity. In particular, when an agent has the authorization to perform a particular composite activity, he has the authorization to coordinate the activities of which the composite activity is parent. This is expressed by the following business rules.

> It is necessary that a role *can coordinate* an activity type$_1$, if an activity type$_2$ *can consist of* the activity type$_1$ and role *can perform* activity type$_2$.
> It is necessary that an activity$_1$ *has coordinator* an agent, if the activity$_1$ *has parent* an activity$_2$ and activity$_2$ *has performer* the agent.

The activities that are performed by a subsidiary agent, are performed by the agents to which the agent pertains.

Event – event type

In the last decades, events have been actively investigated in research communities such as the Knowledge Representation domain, Active Database domain, the architecture description domain. But even within these domains there exist quite distinct notions of the concept 'event'. A substantial distinction is whether these events are considered volatile or non-volatile. **Volatile events** are perdurants that are immediately consumed (removed) after detection. In the Active Database community event definition languages and event detection prototypes such as for example SAMOS (Gatziu and Dittrich, 1993) and Snoop (Chakravarthy and Mishra, 1994) have this conception of event. **Non-volatile events**, on the other hand, are endurants that are never removed but are considered to persist. In the Event Calculus (Kowalski and Sergot, 1986), for instance, events are considered to persist. In active database systems, volatile events have been used to model reactive behavior. Each time when an event is detected, it is reacted upon and the event is removed from the model. The disadvantage of such an event removal policy, however, is that it does not allow for detecting so-called composite events. **Composite events** represent situations that correspond to the (non-)occurrence of several (atomic) events. To detect composite events, events need to non-volatile or they must at least be retained in the system during some time. Unlike atomic events, which occur at a particular point in time, composite events occur over a time interval that spans at least the occurrence times of each involved atomic event. Many event detection languages, among which SAMOS and Snoop, do not incorporate this interval logic and Galton and Augusto (2002) report on the unintended semantics of some composite event operators in these languages.

In the EM-BrA^{2}CE framework, the state of an activity (or service instance) includes the event history of the activity or its sub-activities. Consequently, events are given a non-volatile semantics. Although composite events are not considered explicitly by the vocabulary, composite events can still be included in business rules expressions.

An event *is* an SBVR:individual concept that corresponds to an instantaneous, discrete state change of a concept in the world.
A negative event *is* an SBVR:individual concept that records at at a given moment a particular state change was requested, but could not take place.
'Event *is about* SBVR:concept' *is* an SBVR:associative fact type that represents the concept whose state change is reported by the event.
An event type *is* an SBVR:concept type that *specializes* the individual concept 'event' and that *classifies* an event.
'event type *is type of* event' *is* an SBVR:assortment fact type that categorizes an event as being of a particular event type. Necessity: it is necessary that an event *has type* exactly one event type.
'event *occurs at* time' *is* an SBVR:is-property-of fact type that represents the time at which an event occurs.

Events report a state change of a concept in the world. Ferreira and Ferreira (2006) show that it also can be useful to keep track of negative events. Negative events report that a state change was requested, but that it was prevented from taking place. Negative events provide information about disallowed transitions, and are for instance useful for the purpose of process mining. For instance, when an agent is not authorized to be assigned to perform a particular activity, this can be recorded as a negative event of the event type assignRejected.

Unlike many ontologies for business modeling, such as for instance the Agent-Object-Relationship (AOR) (Wagner, 2003) or Unified Foundational Ontology

(UFO) (Guizzardi and Wagner, 2005), a distinction is made between activities and events. Activities are performed by agents and have a particular duration whereas events occur instantaneously and represent a state change in the world. Changes to the life cycle of an activity are reflected by means of activity events. The framework considers twelve generic activity state transitions that correspond to twelve activity life cycle events.

> An activity event type *is* an event type that describes a category of activity state changes. Example: the activity event types 'created', 'scheduled', 'assigned', 'revoked', 'started', 'factAdded', 'factRemoved', 'factUpdated', 'aborted', 'skipped', 'completed' and 'redone'.
> An activity event *is* an event that corresponds to the state change of an activity. Necessity: it is necessary that an activity event *is about* exactly one activity. Necessity: it is necessary that an activity event *has* exactly one activity event type. Example: anEventX, anEventX *has type* scheduled, anEventX *is about* anActivityX.
> A business fact event *is* an event that involves the state change of a business fact. Necessity: a business fact event *is about* exactly one business fact.

The distinction between activity and event allows for reactive behavior. At each point during execution the history of a business process instance might be inspected through the use of an event query language. When an external event is added to the current state of an activity, that activity enters a new state. In this new state, the activity can undergo an additional transition as a reaction to the external event. Because this second transition is also recorded as an activity event, the system keeps track of its own state, reflecting the external (composite) events that have been reacted upon. The latter prevents the system from reacting twice to the same event.

The fact type 'role *can subscribe to* event type *in context of* activity type' expresses the visibility of events to agents in the context of an activity. It does not express how agents are notified of the event, which can generally occur using either a pull, a push or a publish-subscribe mechanism (Bailey et al., 2005). Furthermore, it is possible that the visibility is constrained by so-called event subscription constraint business rules. All this is in accordance with the "declarative communication logic" design principle.

> 'role *can subscribe to* event type *in context of* activity type' *is* an SBVR:associative fact type that expresses that an agent with a particular role can subscribe to an event of event type in the context of an activity of activity type. Example: seller *can subscribe to* completed *in the context of* ship.
> 'agent *perceives* event' *is* an SBVR:associative fact type that expresses that an event is non-repudiable to a particular agent. Example: anAgentX *perceives* anEventY.

Deontic Assignment

In the literature, there are many languages to reason about the permissions, obligations, and prohibitions of agents with respect to performing particular activities (Goedertier and Vanthienen, 2006d; Governatori, 2005; Knottenbelt and Clark, 2004; Marín and Sartor, 1999; Paschke and Bichler, 2005; Yolum and Singh, 2004). Such deontic concepts are called deontic assignments in the vocabulary (Wagner, 2003). A deontic assignment represents among others the obligation or permission of an agent to perform a particular activity by a particular due date.

> A deontic assignment *is* an individual concept that represents an obligation, prohibition, permission, conditional obligation or conditional permission of an agent towards another agent (beneficiary) regarding the performance of an activity with respect to a given due date.
> ‘deontic assignment *has* due date’
> ‘deontic assignment *involves* activity’
> ‘deontic assignment *has performer* agent’
> ‘deontic assignment *has beneficiary* agent’
> An obligation *is* a deontic assignment that represents the obligation of an agent to perform a particular activity by a particular due date.
> A permission *is* a deontic assignment that represents the permission of an agent to perform a particular activity before a particular due date.

A deontic assignment can also be expressed conditionally. When an agent performs a given activity, a conditional deontic assignment may result from it. For instance, in the shipment-after-payment process model visualized in Figure 3.2 a buyer makes a conditional commitment when he places an order. In particular, a buyer has the conditional obligation to pay the seller if the seller accepts the order. If the seller rejects the order, no obligation results from it. The following definitions are included in the vocabulary:

> An conditional obligation *is* a conditional deontic assignment that represents the conditional obligation that rests on an agent to perform a particular activity before a given due date, after – and on the condition that – a particular agent has done a particular activity within a particular due date.
> An conditional permission *is* a deontic assignment that represents the conditional permission of an agent to perform a particular activity before a particular due date, after – and on the condition that – a particular agent has done a particular activity within a given due date.
> ‘conditional deontic assignment *has conditional due date* date’
> ‘conditional deontic assignment *involves conditional* activity’
> ‘conditional deontic assignment *has conditional performer* agent’
> ‘conditional deontic assignment *has conditional beneficiary* agent’

The existence of deontic assignments is entirely defined by temporal deontic rules and is dependent on the historic behavior of agents playing a particular role in the context of a composite activity. Deontic assignments should not be confused with the deontic propositions of the SBVR. The Deontic propositions in SBVR resemble those of Standard Deontic Logic (SDL) (Føllesdal and Hilpinen, 1971) and express that a particular state of affairs is permissible, necessary, obligatory or prohibited. Like SDL the SBVR expresses the obligation to bring about a certain proposition in an impersonal way: it cannot express the agent to whom a particular obligation or permission applies. Another difference with deontic assignments is that deontic propositions are static; they cannot represent deontic properties that come into effect and cease to hold because of timeouts on deadlines or other events. Finally, the SBVR is not able to express so called contrary-to-duty obligations (Governatori and Rotolo, 2002), reparative obligations that come into existence as the result of the violation of an obligation. For instance, after a due date on an obligation to pay has passed, a violation event occurs.

> A violation event is an event that occurs when an agent does not perform an obligation within the due date of that obligation.
> Necessity: Each violation event *is about* exactly one obligation.

Many deontic logics are closed such that, for instance, prohibition can be derived from the lack of either an obligation or a permission deontic assignment. It would however be unfair to assume that a process modeler must specify deontic assignment rules for each activity type that occurs within a process model. Therefore it is useful to indicate the activities for which explicit deontic assignments must be derived in order to perform them (Segerberg, 1982). This is expressed by the is-property-of fact type ‘activity type *is deontically closed in* state space’.

> ‘activity type *is deontically closed in* state space’ *is* a fact type that expresses that in each state based on the state space, the entire extension of every deontic assignment that involves an activity of the activity type is given in the facts included in the state.

When an activity type is deontically closed in a state space, prohibition is derived from the absence of permission or obligation. When, in contrast, this is not the case, no deontic assignment can be derived from the absence of information.

Non-functional, Quality-of-service Concerns

Given its origin in telecommunication, the term ‘quality of Service’ (QoS) at first sight has little ado with business modeling. However, in the academic research involving web services, the term quality of service refers to a number of non-functional quality requirements such as availability, robustness, scalability, security and trust information (Roman et al., 2005). QoS concerns are also business concerns that can be specified in a language that the business understands. The vocabulary considers the following QoS concerns.

> Spatial availability *is* a quality of service specification that determines the location from which activities of a given activity type can be performed

or that business facts of a given business fact type can be accessed.
Temporal availability *is* a quality of service specification that determines the amount of time during a time period that activities of a given activity type can be performed or that business facts of a given business fact type can be accessed.
Response time *is* a quality of service specification that determines the maximum time period it may take to perform a state transition on an activity of given activity type or on a business fact of a given business fact type.
Throughput *is* a quality of service specification that determines the ratio of activity state transitions or business fact accesses per unit of time.
Historic window *is* a quality of service specification that determines the time period during which historic information about activity events or business concept manipulations must be stored.
Latency *is* a quality of service specification that determines the maximum delay by which concept modifications are propagated.
Security *is* a quality of service specification that determines the identity, privacy, alteration and repudiation facets related to performing activities or consulting information.

Quality of service specifications can be imposed both on fact types (information) and activity types (processes). QoS concerns must be both information- and process-aware rather than exclusively information- or process-driven. This entails that Quality of Service (QoS) specifications on information access should contain information about the activity (or service) context in which information is retrieved. This is particularly important when the same information (or facts) is required in the context of different activities with different QoS requirements. For example, when verifying whether a customer is a high-volume customer, it is not so important to have zero latency on the historic sales records that are consulted. In contrast, when determining the total amount of outstanding debt with a customer, it is likely that sales records must be consulted without latency. Clearly the activity context in which information (facts) are retrieved is an important differentiator of QoS specifications. This is reflected in the vocabulary:

'Fact type *must have* temporal availability *in the context of* activity type'
'Fact type *must have* spatial availability *in the context of* activity type'
'Fact type *must have* response time *in the context of* activity type'
'Fact type *must have* throughput *in the context of* activity type'
'Fact type *must have* historic window *in the context of* activity type'
'Fact type *must have* latency *in the context of* activity type'
'Fact type *must have* security *in the context of* activity type'

QoS specifications on information (or fact types) must be process aware. This relation also holds in the opposite sense: QoS specification on processes (or activity types) must be information aware. The latter is particularly important when strict QoS specifications on business processes are disproportionate with less strict

QoS specifications on information. The fact types 'activity type *can manipulate* business fact type' and 'activity type *can make visible* business fact type' keep track of the business fact types that are accessed by activities of a given activity type. It can be used to determine whether activity type QoS specifications are aligned with fact type QoS specifications.

'Activity type *must have* spatial availability'
'Activity type *must have* temporal availability'
'Activity type *must have* response time'
'Activity type *must have* throughput'
'Activity type *must have* historic window'
'Activity type *must have* latency'
'Activity type *must have* security'

Cost and Time Concerns

Cost and time concerns affect the coordination of activities (or services). For example, in an order acceptation process, a sales representative will not include an expensive 'review creditworthiness' activity that is disproportionate with the insignificant amount of the order. Likewise, a sales representative would not schedule a slow, time-consuming shipment for a rush order of an important customer.

The performance of an activity (or service) inadvertently has financial implications. When activities are performed among agents of different organizations, the financial implication is called a price. When activities are performed among agents that pertain to the same organization, the financial implication is called a cost. O'Sullivan et al. (2002) discuss different techniques for agents (or service providers) to charge money for providing their services and to settle payment. Within organizations cost accounting techniques are usually put in place to determine the internally incurred cost of the activities (or services) that are performed and corresponding incentive-compatible cost allocation models. The EM-BrA^{2}CE Vocabulary does not provide a vocabulary to express charging styles, settlement models, or allocation schemes. Instead, it provides a single cost measure that informs the coordinator of an activity about the expected financial impact of having the activity performed.

Cost of performance *is* the cost that is incurred when performing a given activity.
'Activity *has an expected cost of* cost of performance' *is* an SBVR:is-property-of fact type that represents the cost of performance that is expected to be incurred prior to the start of the activity. Example: anActivityY *has an expected cost of* 4.5 euro.

Derivation rules can specify the fact type 'activity *has an expected cost of* cost of performance' based on the properties of the activity such as the activity type, the agent assigned to perform the activity, the object (or business id) of the activity and the scheduled due date of the activity.

The performance of an activity (or service) inadvertently takes time. When performing a coordination activity, a coordinating agent must take into account the scheduled due date of the coordination activity. In particular, all required sub-activities in the execution plan of the coordination activity must be completed prior to the completion of the coordination activity. For instance, when a sales representative coordinates the processing of a sales order, the order acceptation and shipment sub-activities must be completed before the due date imposed on the coordination activity. Many non-functional properties influence the time required for a service provider to perform an activity: capacity, throughput, arrival rates. The EM-BrA^{2}CE Vocabulary does not provide a vocabulary to express these concerns. Instead, it provides a time measure that informs the coordinator of an activity about the expected duration of performing an activity.

> Duration of performance *is* the duration that is required to perform an activity.
> 'Activity *has an expected duration of* duration of performance' *is* an SBVR:is-property-of fact type that represents the expected time needed to complete a particular activity. Example: anActivityY *has an expected duration of* three working days.

Derivation rules can define the fact type 'activity *has an expected duration of* duration of performance' based on the properties of the activity such as the agent assigned to perform the activity and the object (or business id) of the activity .

3.3 Execution Semantics

Albeit a unifying framework for declarative process modeling, the EM-BrA^{2}CE Framework can still benefit from a specific and formal execution model. A specific execution model further clarifies the meaning of the concepts defined in the vocabulary. Furthermore, given a formal execution model, it becomes possible to enact process models that have been modeled using concepts in the framework, potentially supplemented with specific business rules expressed in existing languages for declarative process modeling. In Section 4.3, for instance, we use the EM-BrA^{2}CE execution model for building two declarative simulation models. The execution model can be used to build a declarative workflow management system prototype, such as Pesic et al. (2007a) have done for DECLARE, which incorporates the ConDec language.

In the literature, there exist many process modeling languages that all have their specific execution semantics. It seems not possible to include the execution semantics of all these languages in one unified execution model without introducing contradictions. Every execution model necessarily makes design choices. Whilst avoiding being over-specific, the EM-BrA^{2}CE Framework aims at incorporating the activity life cycles of a number of important languages and paradigms that have appeared in the literature. In particular, the execution model is influenced by the activity life cycles involved in the case handling paradigm (van der

Aalst et al., 2005), the dynamic discovery and invocation of web services (W3C, 2004b), the choreography and orchestration of semantic web services, such as by the Web Service Execution Environment (WSMX) (M. Zaremba, 2005), and the state transitions considered by the Mining eXtensible Markup Language (MXML) format for event logs (van Dongen and van der Aalst, 2005a).

3.3.1 Colored Petri Nets

Günther and van der Aalst (2005) show how it is possible to represent the execution semantics of the case-handling paradigm in terms of a Colored Petri Net (CP-Net). This approach can be generalized towards the state space and state transition types considered by the the EM-BrA^{2}CE Framework.

There are several reasons for choosing CP-Nets. First of all, CP-Nets have a formal semantics (Jensen, 1993, 1996). Furthermore CP-Nets represent an expressive, high-level modeling language that portrays more modeling convenience compared to, for instance, classical Petri nets. Although each CP-net can be translated into a classical Petri net and vice versa, this does not guarantee the suitability of Petri Nets for modeling in practice (Jensen, 1993). In particular, it is difficult to model data manipulations with classical Petri nets, not allowing for token colors. Another reason for using CP-Nets is that CP-Net models can be simulated, making CP-Nets suitable for rapid prototyping process models and for generating artificial data sets of event logs that can later be used to evaluate the performance of process mining algorithms (Alves de Medeiros and Günther, 2005). Additionally, CP-Nets allow for formal state space analysis that would, in theory, allow for directly analyzing the state space of individual declarative business process models. However, the inclusion of fact-oriented case data and event history into the state space of process models can be expected to result in too large a state space for analyzing realistic models.

Jensen (1996) provides an extensive introduction to the semantics and analysis methods of CP-Nets. Throughout this section the semantics of CP-Nets in terms of their differences to classical Petri nets will be informally discussed whenever a new language construct is encountered.

3.3.2 Places and Color Sets

Just like in classical Petri nets, the **state** or **marking** of a CP-Net is represented by the tokens that reside in each of the **places** of the CP-Net at a particular moment. Unlike classical Petri nets, however, CP-Nets allow to associate a data type, called a **token color**, to each place in the CP-Net such that only tokens of an indicated token color may reside in that place. It is possible for places to contain tokens prior to the occurrence of any state transition in the net. Such a start state or **initial marking** can be defined in terms of initialization expressions for a particular place. The state space of an EM-BrA^{2}CE process model can be modeled using four places, depicted in Figure 3.10:

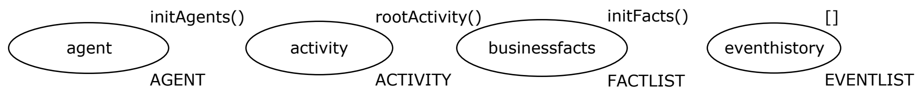


Figure 3.10: The CP-Net places that span the EM-BrA^{2}CE state space

- an `agent` place of token color `AGENT` of which the tokens represent the agents that can coordinate or perform activities. A domain specific function `init-Agents()` can be used to define the agents that are initially present. During the execution of a business process, agents can come into existence or cease to exist. Agents can also form ad hoc groups of agents (holonic agents) that perform an activity as a whole.
- an `activity` place of token color `ACTIVTY` of which the tokens represent the (composite) activities that are coordinated or performed by agents. The initial marking this place consists of a so-called `rootActivity`, that is parent to all other activity instances.
- a `businessfacts` place of token color `FACTLIST` that holds one list token representing a list of business facts that can be manipulated by performing activities. A domain specific function `initFacts()` can be used to define the initial business facts, such as properties of agents, that are present in the system.
- an `eventhistory` place of token color `EVENTLIST` that holds one list token representing an ordered list of historic events that have taken place throughout the life cycle of individual activity instances. This place contains the empty list `[]` as initial marking.

This representation depicts how state in the EM-BrA^{2}CE is constituted of a current state of affairs represented by the `agent`, `activity`, and `businessfacts` places and a history of past business events, represented by the `eventhistory` place. There is some redundancy in this conception of business process state: the current state of affairs can always be obtained by "replaying" the history of past business events. However, this redundancy can be introduced without loss of generality. Moreover, not having to calculate the current state of the process model facilitates process modeling and improves efficiency of simulations.

In the source code below this paragraph, the above token colors are defined in terms of Standard ML (Milner et al., 1990), a functional programming language with compile-time type checking and type inference. The fact-oriented metamodel of the EM-BrA^{2}CE Vocabulary is translated into the color sets `FACT`, `CONCEPT` and `AGENT`, which are here treated as synonyms. These color sets represent a quadruple consisting of a statement identifier (for reification purposes), a subject identifier, a predicate representing one of the fact types and a value. For instance, the fact that a worker `workerX` belongs to department `departmentY` is now represented as a quadruple `(statementZ,workerX,fromDepartment,departmentY)`. To express the existence of an individual concept `workerX` of concept type `agent` the following quadruple can be constructed: `(statementZZ,workerX,has as type,agent)`. This form of knowledge representations corresponds to RDF with reification (W3C, 2004a), one of the foundation languages of the Semantic Web. Notice treating `FACT` and `CONCEPT` as synonyms is a simplification of the SBVR ontology language. However, the simplification is only a limitation in the context of higher-order typing. The idea of representing agents as tokens in a CP-Net is, among other, present in van der Aalst's (1998) representation of workflow.

```
colset CONCEPTid = int with cL..cU;
colset VALUE = union nb:INT + st:STRING + id:CONCEPTid;
colset NOUNCONCEPTTYPE = subset STRING with [...];
colset VERBCONCEPTTYPE = subset STRING with [...];
colset FACTTYPE = union nount:NOUNCONCEPTTYPE + verbt:VERBCONCEPTTYPE;
colset FACT = product       (*the statement id*)        CONCEPTid *
                            (*the subject id*)          CONCEPTid *
                            (*the predicate*)           FACTTYPE *
                            (*the object/value*)        VALUE;
colset FACTLIST = list FACT;
colset CONCEPT = FACT;
colset AGENT = CONCEPT;
```

The `ACTIVITY` color set is a septuple composed of an integer that denotes the non-business activity identifier, an activity type, a business id that is a list of business concepts that uniquely identify the activity, an activity identifier that denotes the immediate parent activity, an agent identifier that denotes agent that is currently assigned as the coordinator or performer of the activity, a time indication that denotes the due date by which the activity is to be performed, an event list that keeps track of the concept manipulation events that have occurred within the context of the activity. Furthermore, activity is defined as a timed token. This allows to model time evolution. The idea of representing individual activities as tokens in a CP-Net is based on Günther and van der Aalst's (2005) representation of Case Handling in CP-Nets.

```
colset ACTIVITYid = int with aL..aU;
colset ACTIVITYTYPE = subset STRING with [...];
colset ACTIVITY = product   (*the activity id*)         ACTIVITYid *
                            (*the activity type*)       ACTIVITYTYPE *
                            (*the business id*)         FACTLIST *
                            (*the parent id*)           ACTIVITYid *
                            (*the coordinator/worker*)  AGENTid *
                            (*the due date*)            TIMESTAMP *
                            (*transaction events*)      EVENTLIST
                            (*timed token*)             timed;
```

The `EVENT` color represents the activity events that have occurred. It is a septuple represented as the cardinal product of an activity identifier, an activity type, a business id that is a list of business concepts that uniquely identify an activity, an event type denoting the nature of the activity state transition, an agent identifier that denotes the agent who has brought about the state transition, a list of facts that specify the event and a time indication that denotes the time at which the state transition has occurred. The idea of incorporating history into an event history token stems from van Hee et al. (2006), Although the authors do not propose to incorporate activity events but rather propose to incorporate the consumption and production of tokens in each input and output place as events.

```
colset EVENTTYPE = subset STRING with [ "created","createRejected",
                                        "scheduled","scheduleRejected",
                                        "assigned","assignRejected",
                                        ...,
                                        "completed","completeRejected"];
colset EVENT = product  (*the activity id*)          ACTIVITYid *
                        (*the activity type*)        ACTIVITYTYPE *
                        (*the business id*)          FACTLIST *
                        (*the event type*)           EVENTTYPE *
                        (*the coordinator/worker*)   AGENTid *
                        (*event parameters*)         FACTLIST *
                        (*the time of occurrence*)   TIMESTAMP;
colset EVENTLIST =list EVENT;
```

Because each place in a CP-Net can contain multiple tokens, possibly of the same token color, the content of a place can be represented as a **multi-set**. This might raise the question why both the `businessfacts` and the `eventhistory` place consist of exactly one **list token** containing an ordered list of tokens. The reason for this modeling feature, is that it cannot be foreseen at design-time how many tokens will actually be required in order to fire a transition. For instance, it is not possible to foresee which and how many tokens representing business concepts, business facts and events actually need to be inspected when deciding upon assigning an agent to a particular activity. In addition, to trigger a transition upon the non-presence of a token without using inhibitor arcs also requires this modeling feature (Mulyar and van der Aalst, 2005). To overcome this problem, it is required to take all business concept, business fact and event tokens into consideration by removing a list with all tokens from both places, querying this list and returning a potentially updated list of tokens upon termination of a transition.

3.3.3 Transitions

As in classical Petri nets, the dynamics of CP-Nets come from **transitions** (represented as rectangles). Places in CP-Nets are linked to transitions via input and output arcs. **Input arcs** indicate that a particular transition may remove (consume) tokens from a particular place, whereas a **output arcs** indicate that a

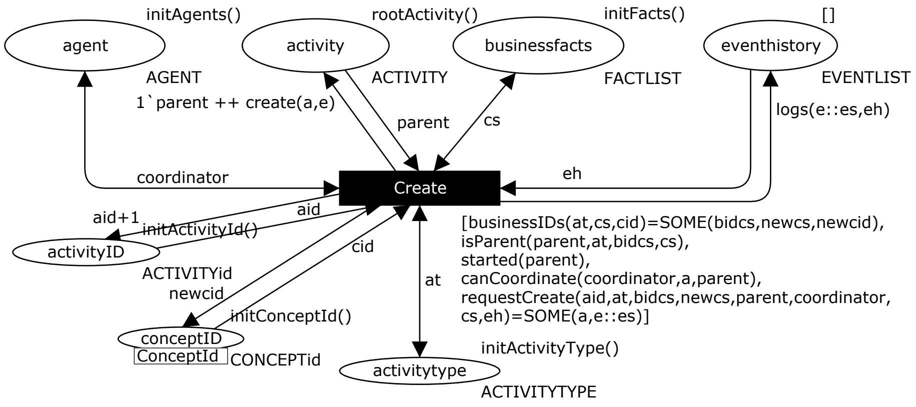


Figure 3.11: A CP-Net model of the Create transition

transition may add (produce) tokens to a particular place. Informally, a CP-Net transition may fire (**fire rule**) when on each input place a required number of tokens can be found that together satisfy the **guard condition** of that transition. As a result of firing, a transition consumes a number of tokens on each input place and produces a number of tokens on each output place. The value and amount of tokens consumed and produced in place can be manipulated through the use of **arc expressions**.

The Create Transition

The `Create` transition, depicted in Figure 3.11, is one of the most complex transitions in an activity life cycle as it involves many aspects:

- the determination of the activity type
- the determination of the activity identifiers
- the determination of the parent activity
- the determination of the agent who coordinates the activity
- establishing whether the activity can be created
- logging the creation of an activity as an event in the event history

In the remainder of this section these aspects are discussed consecutively.

The determination of the **activity type** of the activity that is about to be created can be modeled by consuming and producing an `ACTIVITYTYPE` token from the `activitytype` place and binding it to the variable `at`. As the same token is consumed and produced by the `create` transition, the incoming and outgoing arc are replaced by a **bidirectional arc**, that is both an input and an output arc.

Because the `ACTIVITYTYPE` token is not timed, it can be consumed at the same modeling time for the creation of other activities. This pattern applies to the whole CP-Net, such that it allows for the desired behavior that activity state transitions can occur concurrently.

In the EM-BrA^2CE Vocabulary activities have two identifiers: a non-business identifier and a business identifier. The determination of a **non-business identifier** is modeled using the "ID Manager" Pattern (Mulyar and van der Aalst, 2005). Initially the `activityID` place stores one token with an integer value determined by `initActivityID()`. Each time the `Create` transition fires, the token is consumed and used as an identifier for the activity to be created. In addition, the incremented integer value is produced on the output arc. By incrementing and memorizing the last identifier value, the uniqueness of the non-business identifier can be guaranteed.

The determination of an activity **business identifier** is more complex as it involves real-world business concepts. In some cases, the creation of a activity will (in part) involve the creation of a new business concept whilst in other cases the creation of an activity only involves the identification of existing business concepts. For instance, when a customer applies for credit, the activity `applyForCredit` might be identified by a new business concept of type `creditApplication` representing the new credit application. In contrast, when a customer wants to modify the requested duration of the credit, the activity `requestChange` might be identified by the already existing `creditApplication` business concept. This complexity is encapsulated within the guard condition `businessIDs(at,cs,cid)= SOME(bidcs,newcs,newcid)`. The function `businessIDs (at,cs,cid)` takes as input the variable `at`, a list of existing business concepts, bound to the variable `cs` and a unique identifier for new business concepts to be created, bound to variable `cid`. On its output the `businessIDs` either returns `NONE` when it fails to determine suitable business identifiers or a triple `SOME(bidcs,newcs,newcid)` with respectively the existing and newly generated business concepts and `newcid` with a properly incremented value of `cid`.

Some activities can exist only within the life cycle of a (composite) **parent activity**, whereas other activities can also be created independently from a parent activity. The latter activities have `rootActivity` as their immediate parent. The logic of determining a proper parent for an activity to be created is encapsulated in the `isParent(parent,at,bidcs,cs)` guard condition. The input variable `parent` is bound to an `ACTIVITY` token from the `activity` place. Additionally, for a child activity to be created, it required that the coordinating parent activity has already started. This is expressed with the `started(parent)` guard condition.

Upon creation of an activity is is unclear who will eventually perform the activity until an activity is assigned to a particular agent. Until that period, the accountability for an activity can be attributed to the **coordinating agent** (human or system) who has created the activity. In order to determine the coordinating agent an `AGENT` token is consumed and produced from the `agent` place and bound to the `coordinator` variable. When activities are created within the context of a parent activity, the agent assigned to the parent activity is also the coordinator of the new child activity. For instance, when `agentX` has been assigned to the parent activity `handleCreditApplication` he is also identified as the coordinator of a newly created child activity `reviewCredit`. When, in contrast, an activity has the `rootActivity` as parent, an agent can only create an activity when it has to role of an agent who can coordinate the particular activity. This logic is concealed in the `canCoordinate(coordinator,a,parent)` guard condition.

Additionally, a set of domain-specific business rules might determine whether the specified activity can be **created** or not. The guard condition

```
requestCreate(aid,at,bidcs,newcs,parent,coordinator,cs,eh)=SOME(a,e::es)
```

is a hook that allows for the evaluation of business rules to check whether an activity with the specified features can be created. The creation of an activity might depend on the evaluation of strict business rules (hard constraints) and upon the freedom of choice of agents not to follow general guidelines (soft constraints) with regard to the creation of a particular activity. When no activity is to be created the function returns either `NONE` resulting in a failure of the guard condition or `SOME(a,e::[])` for which `e` is bound to an event of type `createRejected`. When on the contrary an event must be created the function returns `SOME(a,e::es)`. In this case `a` is bound to a new activity token that is produced in place `activity`. In addition, `e` is bound to an event of type `created` and `es` represents a list of events that represents the newly created business concept identifiers (if any). The arc expression `1‘parent ++ create(a,e)` returns the parent token and, in the case of a successful `created` event, a new activity token `a`. This activity token also receives a token time that indicates the earliest time at which another activity state transition can occur. In this way, time can be incorporated into the model. The events `es` are added as transaction events to the newly created activity. Only when the activity completes these transaction events are committed and affect the list of business concepts and facts `cs`.

Via the arc expression `logs(e::es,eh)` multiple events related to the `Create` transition are incorporated in the **event history**, bound to the variable `eh`. The event `e` is of the event type `created` or `createRejected`. Additionally, the events `es`, related to the newly created identifying business concepts also need to be added to the event history.

The Schedule Transition

Time aspects are often an important aspect in the coordination of activities. An important timing aspect is the time point at which an activity is expected to be completed. In the EM-BrA^{2}CE Vocabulary this moment is called the **due date** of an activity. Due dates on activities originate from (informal) service-level agreements, legal requirements or in-house timing policies and strategic plans. When it becomes clear that an activity is not going to be fulfilled before due date, coordinating agents must make alternative arrangements to speed up the processing of the activity or the minimize the consequences of tardiness. Making these arrangements is called **deadline-based escalation** by van der Aalst et al.

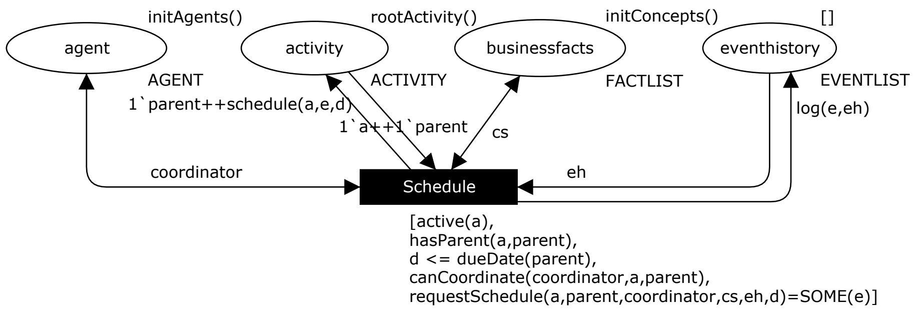


Figure 3.12: A CP-Net model of the Schedule transition

(2007b). The authors classify a number of escalation mechanisms that can be put in place and evaluate these mechanisms by means of simulation.

Figure 3.12 depicts the `Schedule` transition. In this transition the following aspects are contained:

- the identification of the parent activity
- the identification of an agent who can coordinate the activity
- establishing whether the specified schedule does not violate any business constraints or guidelines
- logging the scheduling of an activity as an event in the event history

In the remainder of this section these aspects are discussed consecutively.

In general, the life cycle of child activities is contained within the life cycle of a **parent** activity. This means that activities are created only when their immediate parent activity has already started. Furthermore, a child activity must be completed prior to the completion of its parent activity. This conception of composite activities has an important consequence for scheduling activities: an activity cannot be scheduled with a due date later than the due date of its parent. Both the activity to be scheduled `a` and its parent activity `parent` are consumed from the `activity` place. Guard condition `hasParent(a,parent)` ensures that `parent` is bound to the parent activity of `a`. Furthermore, guard condition `d <= dueDate(parent)` ensures that the chosen due date `d` occurs before the due date of the parent activity.

The agent who will schedule the activity, the **coordinator**, is identified via the guard condition `canCoordinate(coordinator,a,parent)`.

To check whether the chosen due date `d` does not violate any hard **business constraints** and to model the freedom of choice with respect to observing soft **business guidelines** regarding deadlines, the `Schedule` transition has a `requestSchedule(a,parent,coordinator,cs,eh,d) = SOME(e)` guard condition. Only when the function `requestSchedule(a,parent,agent,cs,eh,d)` returns a `scheduled` event, the arc expression `1‘parent++schedule(a,e,d)` will produce a parent token and an activity token with updated due date. When the function returns a `scheduleRejected` event, the activity token `a` will not be updated. When the function returns `NONE`, the transition will not take place.

When the state transition occurs, either a `scheduled` or a `scheduleRejected` event is to be incorporated within the event history. For optimization purposes, it can still be possible to leave out some of the activities in the event history. These considerations are however, not relevant to the discussion of the semantics of EM-BrA^{2}CE.

The Assign and Revoke Transition

The coordination of activities also requires a coordinating agent to assign the activity to an agent who will perform the activity. Conversely, when an agent can no longer perform an activity as planned, the assignment must be withdrawn. This is modeled in the CP-Net by means of the `Assign` and, its counterpart, the `Revoke` transition depicted in Figure 3.13. The following aspects are contained:

- the identification of the coordinating agent and the agent who will be assigned to the activity
- checking whether the assignment or revocation does not violate any business constraints or guidelines
- logging the assignment of an activity as an event in the event history

In the remainder of this section these aspects are discussed consecutively.

For an agent to be assigned to an activity, the activity must not yet been assigned. Conversely, for an assignment to be revoked, the activity must have been assigned to an agent. This is respectively expressed by the `not(assigned(a))` and `assigned(a)` guard conditions.

To identify an agent that can coordinate the assignment or revocation of a worker to a activity, an `AGENT` token must be bound to the `coordinator` variable that satisfies the guard condition `canCoordinate(coordinator,a,parent)`. To identify a worker agent, an `AGENT` token must be bound to the `worker` variable in the assign transition.

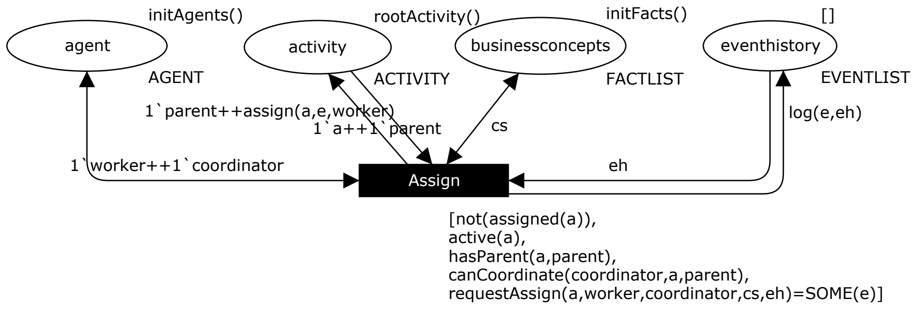
To check whether an assignment or revocation does not violate any business constraints or guidelines, respectively the

```
requestAssign(a,worker,coordinator,cs,eh)=SOME(e)
```

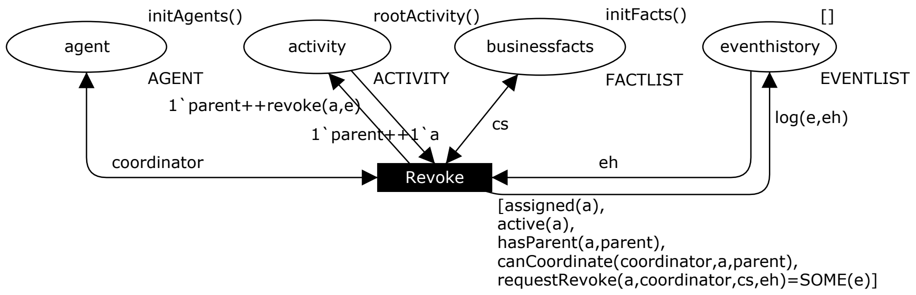
and

```
requestRevoke(a,coordinator,cs,eh)=SOME(e)
```

have been conceived. The variable `e` is bound to an event of type `assign`, `assign-Rejected` or a `revoked`, `revokeRejected` respectively. In function of the event type, the `assign(a,e,worker)` and `revoke(a,e)` arc expressions update the state of the activity bound to `a` or make no activity state update.



(a) the Assign transition



(b) the Revoke transition

Figure 3.13: A CP-Net model of the Assign and Revoke transition

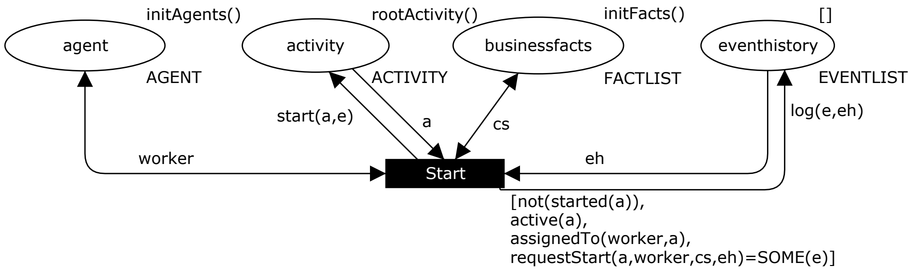


Figure 3.14: A CP-Net model of the Start transition

The Start Transition

Once an activity has been assigned to a worker agent, the activity can finally start. This start of an activity marks the moment from which a worker can actually perform operations that affect the environment or that collect information about the environment. Figure 3.14 represents the logic of the `Start` transition. As expressed by the `not(started(a))` guard condition, an activity can only start if it has not yet started. Moreover, the worker who triggers the start transition must be a worker that has been previously assigned to to the activity, as expressed by the `assignedTo(worker,a)` guard condition. The guard condition `requestStart(a,worker,cs,eh)=SOME(e)` is once again used to check whether the starting of an activity violates any existing business rules.

The Fact Manipulation Transitions

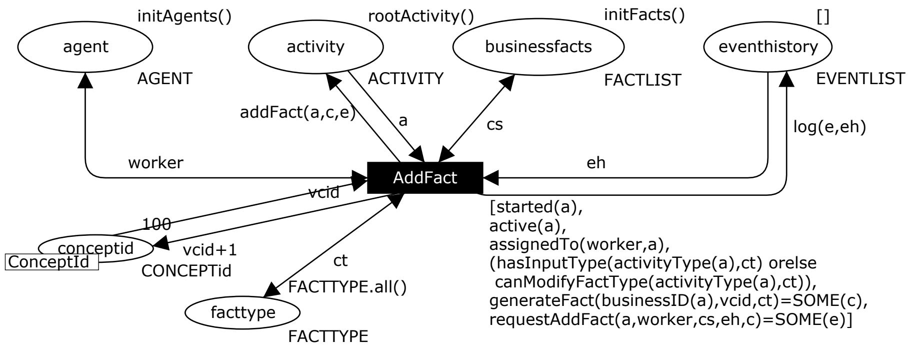
During the performance of an activity a worker can bring about some changes in the environment (the physical world) or retrieve some information from the environment. These changes are reflected in the manipulation (the **addition, removal or update**) of business concepts and facts that pertain to the state space of an activity. Business concepts and facts, however, can be shared among multiple activities that reside within the (information) system. Therefore such manipulations can be treated in two different ways: a stateless and a stateful approach. In a **stateless approach** each manipulation is immediately proliferated to the entire system to alter the state of all other concurrent activities. Because the state of activities does not differ from the state of the information system, this is called a stateless approach. In a **stateful approach** manipulations immediately affect the state of the activity, but manipulations are only carried through (committed) once the activity has completed. Case handling, for instance, uses a stateless approach (van der Aalst et al., 2005), whereas BPEL4WS uses a stateful approach (Curbera et al., 2003). Although both approaches are equally meaningful, only the more complex stateful approach is modeled in the CP-Net model of the data manipulation transitions.

Many different transaction handling mechanisms can be put in place to guarantee the integrity of data manipulations, for example expressed in terms of "ACID properties" (however it is not always clear whether each ACID property is meaningful in a collaborative environment (Dumas et al., 2005)). A solution based on locking, for instance, is the distributed **Two-Phase Commit** (2PC) protocol (Bernstein and Goodman, 1981). Case handling tools such a FLOWer implement a 2PC transaction handling protocol. However, the required locking of business facts during the long-running active part of an activity life cycle is often seen as too restricting, inhibiting the concurrency of activities within business processes. A solution to the concurrency problem is, for instance, offered by the **Tentative Hold Protocol** (Roberts and et al., 2001). This protocol allows for tentative, non-blocking holds or reservations to be requested when starting an activity. When a worker has manipulated a business concept or fact in the contexts of performing an activity, other workers that have taken reservations on this business concept are signaled that their reservations do no longer hold and arisen conflicts are solved. Although transaction handling is a necessary requirement, for reasons of clarity it has been left out of the CP-Net model of the EM-BrA^{2}CE activity life cycle.

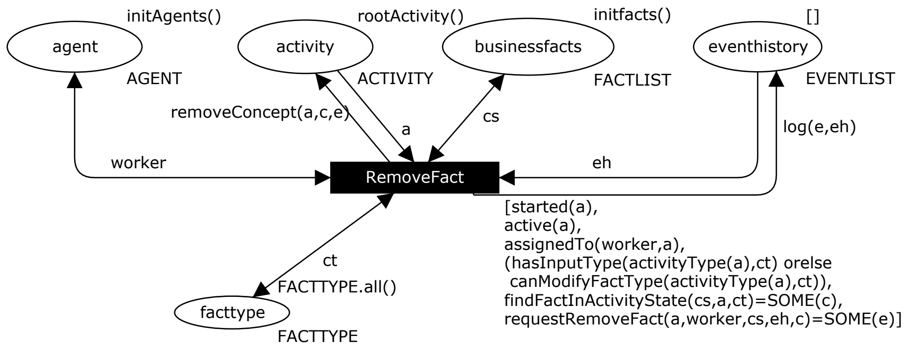
Figure 3.15 represents a CP-Net model of the `AddFact`, `RemoveFact` and `UpdateFact` state transitions. The following aspects are contained in the model and are consecutively discussed in the remaining paragraphs of this section:

- determining a unique concept identifier (statement id) in the case of addition
- identifying the business concept whose property is being manipulated (subject id)
- identifying the concept type of the fact to be manipulated (predicate)
- identifying the new value of the concept to be manipulated (concept value)
- identifying the worker as the worker that is assigned to the activity
- checking whether the manipulation does not violate any business rules
- logging the manipulation event and supplementing it to the activity transaction list (stateful approach)

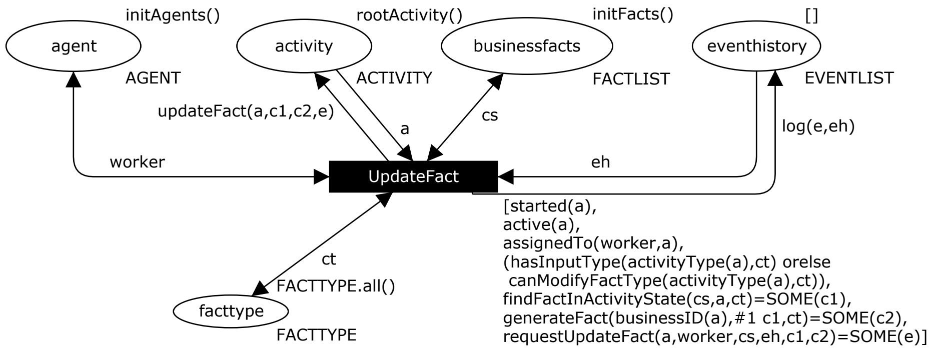
When a new concept or fact is added during the performance of an activity, this concept or fact must be asserted to the system using a globally **unique identifier** identifying the concept or fact as a reified statement (a statement id). The determination of such a unique concept identifier is modeled using the "ID Manager" Pattern (Mulyar and van der Aalst, 2005) and involves place `conceptid`, that forms a **fusion place** that shares the same tokens with the previously discussed `businessID` place. In principle another globally unique identifier is required when adding a noun concept. However, without loss of generality, it can be assumed that a noun concept identifier is equivalent to the identifier of the statement that asserts its existence.



(a) the AddFact transition



(b) the RemoveFact transition



(c) the UpdateFact transition

Figure 3.15: A CP-Net model of the fact manipulation transitions

The addition, removal or update of a business concept or fact requires the identification of the **fact type** of the fact that is being manipulated. This is modeled with the `facttype` place, from which a `FACTTYPE` token is taken and bound to the `ct` variable. In the EM-BrA^{2}CE Vocabulary, facts can only be manipulated by an activity of a particular activity type when activity is of an activity type that can manipulate the fact types of the fact. These concerns are captured by the `canModifyFactType(activityType(a),ct)` guard conditions.

When adding or updating a concept or fact, a new **value** is to be generated. Such a value of type `VALUE` can either be a number, a string or an identifier. Two non-business requirements can be formulated for concept values. First, both string and number values must pertain to the domain of the concept type. In addition, values that refer to the business id (subject id) of other business concepts must guarantee referential integrity. These concerns are present within the `generateFact(businessID:CONCEPTLIST,vcid:CONCEPTid,ct:FACTTYPE)` function.

The manipulation of business facts might lead to the violation of hard and soft business constraints that are specified on these business facts. For instance, during a `requestChange` activity, a registered customer might remove earlier provided collateral information involving an open `creditApplication`. This might trigger a warning message (soft or hard business constraint) that a `creditApplication` business concept is incomplete without collateral information. By implementing functions such as `requestAddFact(a,worker,cs,eh,c)`, `requestRemoveFact(a,worker,cs, eh,c)`, and `requestUpdateFact(a,worker,cs,eh,c1,c2)` compliance to such business rules can be enforced.

The arc expression `log(e,eh)` **logs** the data manipulations as activity events in the event history. Nonetheless, data manipulations are not yet forwarded to affect the business facts and concepts visible in the context of other activities. Instead data manipulations are only carried through once the involved activity has properly completed (stateful approach). Meanwhile every data manipulation event `e` is added to the transaction list of an activity `a` by the functions `addFact(a,c,e)`, `removeFact(a,c,e)` and `updateFact(a,c1,c2,e)`. This transaction list is modeled as a list of data manipulation events and is included in the color set of the `ACTIVITY` token. Upon completion of the activity, the transaction list is committed to affect the globally visible business concepts and facts.

The Complete Transition

When a worker completes an activity, the work represented by the activity is considered to be completed and all data manipulations are proliferated to the entire system. After completion, it is no longer possible for a worker to do supplemental business fact or concept manipulations without reopening the activity instance (modeled as a `Redo` transition). Figure 3.16 represents a CP-Net model of the `Complete` transition. The following aspects are addressed:

- identifying the worker as the worker that is assigned to the activity

- verifying that the activity has no active child activities
- verifying whether the completion of the activity does not violate any hard or soft business constraints
- committing the business fact manipulation transaction list to update the system's state accordingly
- logging the completion event and removing any activity events from the event history that are not able to affect the life cycle of other activities (event garbage collection).

An activity can only complete if it has been previously started. This is expressed with the `started(a)` guard condition. The worker who decides upon completing an activity must be a worker that has been previously assigned to to the activity, as expressed by the `assignedTo(worker,a)` guard condition. The guard condition `requestComplete(a,worker,cs,eh)= SOME(e)` is once again used to check whether the starting of an activity violates any existing business rules.

The work of coordinating a number of activities within a business process is modeled as a (composite) **parent** activity. Consequently, for a child activity to be created, it required that the coordinating parent activity has already started. Conversely, for an activity to complete it is required that the coordinating parent activity is still active. The latter requirement is verified with the `noActiveChildren(a,eh)` guard condition.

As discussed in the previous section, the CP-Net model contained in this text models the stateful approach with respect to business fact manipulation. During the execution of an activity each business fact manipulation is added to a transaction list, modeled as a list of data manipulation events included in the color set of the `ACTIVITY` token. Only upon completion of an activity the transaction list is **committed** to affect the globally visible business facts. This operation is modeled with the `commit(cs,a,e)` arc inscription.

The completion of an activity can trigger the creation or start of subsequent activities that are temporally dependent upon the activity. In that case the completion of the activity is a valuable event that needs to be retained in the event history. Other activity events (such as events of type `created` or `scheduled`) are perhaps less valuable, because it is detected that there exist no business rules that involve these activity events in relationship to life cycle events of other activity types. As such events are not able to affect the life cycle of other activities it is possible to remove them from the event history without side effects. This is called **event garbage collection**. These concerns are concealed in the `log(e,eh)` arc inscription.

The Abort, Skip and Redo Transitions

In addition to creating, scheduling and assigning an activity, coordinating an activity could also involve the canceling, the imperfect (incomplete) termination

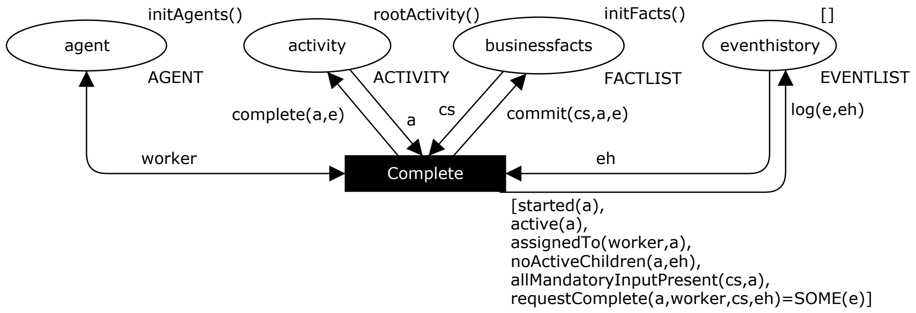
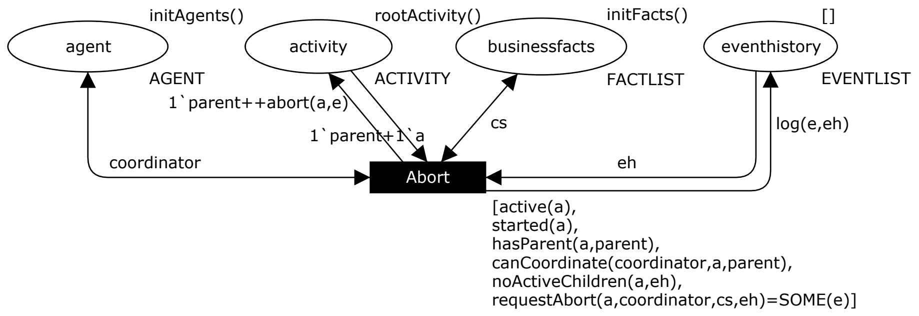


Figure 3.16: A CP-Net model of the Complete transition

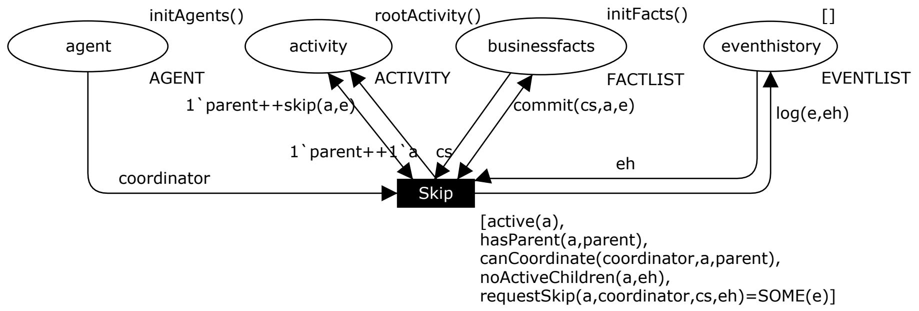
and reopening of an activity. The latter activity life cycle events are respectively modeled as `Abort`, `Skip` and `Redo` transitions, depicted in Figure 3.17.

Sometimes it is necessary for a coordinating agent to cancel or **abort** an active activity or even an entire group of activities. Such cancelation might be part of natural behavior of activities in other cases it might be required to abort an activity when an unforeseen exception occurs (Russell et al., 2006). For instance, in a purchase process it might be required to request a price quote with a minimum number of suppliers. When a required number of proposals are received from suppliers a purchase decision is made. At that moment any remaining active request for quote activities can naturally be canceled out. When, on the other hand, an external event would render the ongoing purchase unwanted, an exception has occurred and all activities within the purchase process must be aborted. An activity can only be canceled when it is in an active state, as modeled with the `active(a)` guard condition. Furthermore, a parent (composite) activity cannot be canceled when it still has active children, as modeled with the `noActiveChildren(a,eh)` guard condition. The semantics of a "cascading abort", in which all active children of a composite activity are aborted, cannot be directly modeled within the current CP-Net, as it cannot be foreseen how many activity tokens (children) need to be consumed from the `activity` place. However, such a "cascading abort" can be seen as the occurrence of multiple abort transitions within the CP-Net. Likewise, the CP-Net does not model so-called compensating activities. Compensation can be obtained by a combination of an `Abort` and `Create` transition. Unlike the `Complete` transition, the `Abort` transition does not commit the business fact manipulation events that pertain to the activity.

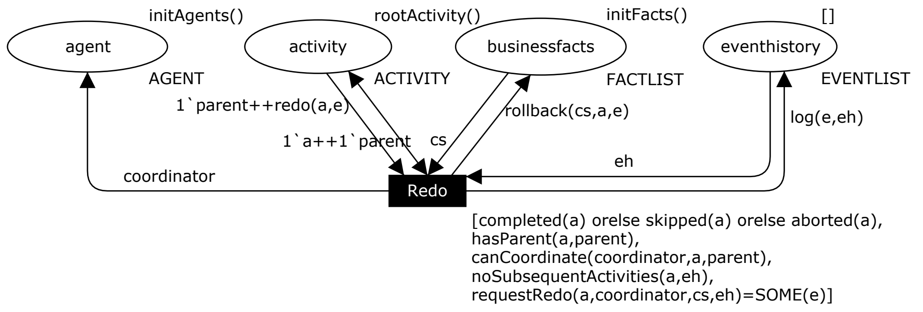
Skipping an activity is a form of imperfect completion of an activity, as it is not required that all postconditions are fulfilled when skipping an activity. Unlike the `Abort` transition, the `Skip` transition does commit every business fact manipulation event that has occurred during the life cycle of an activity. This is modeled with the `commit(cs,a,e)` arc inscription. The possibility of skipping an activity stems from the case handling paradigm (van der Aalst et al., 2005) and allows a coordinator to by-pass an activity when it is no longer deemed



(a) the Abort transition



(b) the Skip transition



(c) the Redo transition

Figure 3.17: A CP-Net model of the Abort, Skip and Redo transition

required. In the case handling case, however, skipping means stepping over the entire activity without any case data being manipulated. Here skipping an activity can still involve the partial manipulation of business facts. Skipping provides a lot of flexibility as it enables a business process to bypass standard behavior as required without such a possibility being explicitly defined in the process model. For instance, when a customer requests information and preliminary, non-binding about a credit rate, it might be useful to perform a `makeProposal` activity without the customer even having identified himself. Such functionality could be provided, when a bank clerk can skip over a number of activities in the credit approval process.

After completing, aborting of skipping an activity, an activity can under some conditions be reactivated by performing a `Redo` operation. This transition has the effect that any previously committed business events are undone, this is modeled by the `rollback(cs,a,e)` arc expression. All previously committed business fact manipulation event that pertains to an activity's transaction list are retained. In this way, a worker can decide which data manipulation can be retained from previous executions and which require alteration. After alteration, a new list of business fact manipulation events can be committed upon completion of the activity. In the case handling paradigm redoing an activity changes the state of added case data from 'defined' to 'unconfirmed'. Although differently encoded, the semantics of the `Redo` transition closely resembles redoing an activity within the case handling paradigm. In addition to business rules constraining the redoing of an activity, an activity cannot be redone when any subsequent related activities have been using information that is being retracted from the system via the rollback operation. This is expressed in the `noSubsequentActivities(a,eh)` guard condition. When such subsequent activities are present, all of them need to be redone before the activity in question can be redone.

3.3.4 Unspecified semantics

Although the proposed CP-Net models a large proportion of the intended semantics of EM-BrA^{2}CE, not all aspects have been fully specified. For reasons of generality, or due to inherent limitations of CP-Nets a number of aspects have been consciously omitted. In this section these omitted aspects are briefly discussed.

Central–Decentral Coordination

The execution semantics do not specify whether the coordination of an activity happens centrally, or decentrally. A central coordination presupposes a central coordinator that is aware of all underlying business and transition constraints, and controls whether a particular transition can take place or not. Typical examples of central coordination are Workflow Management Systems (WfMS). A decentral coordination allows the coordination of activities (services) by different agents (service providers). The logic of the underlying business concerns is distributed and resides with multiple agents (service providers). A typical example are distributed, independently interacting web services. Whereas decentral coordination corresponds to the architectural requirements of a networked, service-based economy (Goethals et al., 2007), it also requires the distributed agents (service providers) the synchronize about whether a state transition can take place or not. Lemahieu et al. (2003) and Snoeck et al. (2004) describe a two-phase coordination protocol, to determine whether a state transition can take place or not in a distributed setting. These concerns are purposefully left out of consideration in the framework. Choosing for a particular coordination setting, would not in keeping of the framework to be a unifying framework for declarative process modeling. Notice that this coordination concern is orthogonal to the design principle of declarative process modeling, to model business concerns from a third-person perspective, as identified in section 2.2 of the previous chapter.

Direct–Indirect Communication

In general, coordination can take place through direct or indirect communication. Direct communication involves the actors in a business process to directly communicate with one another. Indirect communication can be understood as communication by the manipulation of an artifact in the environment. For instance, when an invoice is distributed to the desk of an employee this might trigger a process of verifying, paying and booking the invoice in the general ledger in which the invoice becomes the artifact of indirect communication. Another example of indirect communication is, for instance, the KanBan system for automobile production that was invented by Toyota (Ohno, 1988). The KanBan system is a signaling protocol that signals demand for goods in the supply chain. In the research domain of Multi-Agent Systems (Hadeli et al., 2004) this form of indirect coordination is called **stigmergy**. An important design principle of declarative process modeling approaches, as identified in section 2.2 of the previous chapter, is that they are communication-agnostic. This entails that it is modeled what events and facts about business concepts agents can perceive, not how this is communicated.

Reactive Behavior

A disadvantage of CP-Nets is that it lacks the **reactive behavior** of an open system (Eshuis and Dehnert, 2003). In a CP-Net every state transition occurs within the model, whereas in reality state transitions occur on the initiative of an agent that resides in the environment of the modeled system. To synchronize state information both system and environment communicate by means of exchanging events. In reality, it is possible for reactive systems not being able to respond to their environment in a timely fashion. Since an information system is also a reactive system that runs in parallel with its environment, it also risks to loose synchrony with its environment.

The consequences of the lacking reactive behavior of CP-Nets should however be put into the right perspective. First of all the possibility that an information system can go out of sync with its environment is a possibility that can be left out of consideration in specifying a semantics for EM-BrA^{2}CE. It is acceptable to include a number of events that would normally take place in the environment (such as the actions of external agents) within the boundaries of the CP-Net. Nonetheless, **timeout events**, an important category of events, cannot be incorporated within the closed system of a timed CP-Net. The reason is that model time is exogenous to the CP-Net and it cannot be used to conditionally fire a transition without provoking undesired side effects. Time-triggered activity life cycle operations such as deadline escalation can therefore not be explicitly included in the execution semantics.

Transaction Handling

Evidently, activities can take place concurrently in the execution semantics. Like any other concurrent system, this entails that a mechanism must be put in place to maintain the ACID properties of state (composite) transitions. Such transaction handling can be added by keeping track of the locks or reservations that have been requested on particular (business) facts. Although it is possible to include a **transaction handling** mechanism such as a Two-Phase Commit or Tentative Hold protocol, such a mechanism is not included. It can be seen as a separate concern.

Composite State Transitions

Another part of intended semantics that has been left unspecified in the CP-Net is the presence of **composite state transitions**. Composite state transitions are state transitions that occur for a particular group of activities at the same time. “cascaded abort” and “cascaded redo” transition, for instance, are composite state transitions that represent the automatic aborting or redoing of all child activities when a composite activity is is being aborted or redone. A “delegate” transition, for instance, is the combination of a revoke and assign transition and represent the coordinating activity of a worker assigning an activity to another worker (Li et al., 2003; Wainer et al., 2007; Zhang et al., 2003). These transitions, however, can be omitted from the CP-Net without loss of meaning as they can be simulated by performing a number of subsequent atomic state transitions at the same model time.

3.4 Conclusion

The EM-BrA^{2}CE Framework is a unifying approach, that integrates many works on enterprise modeling, declarative process modeling, service modeling, semantic web services, process mining, and access control. It is a concise synthesis, that aims at maximal expressiveness while introducing only a minimum number of concepts. It has been devised in accordance to the design principles for declarative process modeling outlined in the previous chapter. The framework is intended to be a foundation in integrating and developing existing and new forms of declarative business process modeling. In the remainder of this text, the definitions and execution semantics of the framework serve as an ontological foundation for a number of declarative techniques for modeling and mining business processes.

CHAPTER 4

Modeling and Mining within the EM-BrA^{2}CE Framework

In the previous chapter, we have presented the EM-BrA^{2}CE Framework as a unifying framework for declarative process modeling. It consists of a foundational vocabulary and a specific execution semantics. It can be seen as an attempt to bring structure to a number of foundational vocabularies and execution models that exist in the literature.

In this chapter, we introduce a number of declarative techniques for modeling and mining business processes within this framework. In Section 4.1 we show how business concerns relate to business rules. To this end, we propose a classification of sixteen business rule types and discuss related work. In Section 4.2, we indicate how one such business rule type, namely temporal deontic rules, can be formalized. To this end, we introduce a formal language for expressing and reasoning about temporal deontic rules. We also indicate how it is possible to visualize and verify a set of rules in this language. In Section 4.3, we provide a proof-of-concept of the EM-BrA^{2}CE execution model, by including two simulation models. Section 4.4 enumerates a number of process mining tasks that can be identified within the framework and introduces the concept of declarative process mining.

4.1 Documenting Business Concerns

In this section we show how the framework can be used for declarative process modeling. In particular, we define sixteen business rule types. They refer to several business concerns that can be identified when modeling business processes: data concerns, control flow concerns, and access control concerns. For each business rule type we provide a definition, related work, and indicate how it can be declarative enforced within the execution semantics of the framework. However, in this section we do not define formal languages that would allow to make inferences with these business rule types, nor do we provide verification or visualization mechanisms. Nonetheless, we claim this section of the text to be valuable from an end-user perspective. Already being capable of documenting business concerns in declarative process models yields many advantages.

Table 4.1 gives an overview of the rule types that are defined in this section and indicates the state transition types, already defined in Section 3.1.2, at which these business rule types are relevant. At each given state, an agent might request a particular state transition to occur. Business rules constrain the transitions in a state space. Informally, it suffices to check prior to the occurrence of a state transition whether relevant business rules will be violated or not. When no business rule is violated, the state transition can take place. When, on the other hand, the transition would lead to an intolerable violation of a business rule, the state transition is prevented from taking place.

Table 4.1: Relating transition types to business rule types

	create	*schedule*	*assign*	*revoke*	*start*	*addFact*	*removeFact*	*updateFact*	*complete*	*skip*	*abort*	*redo*
Temporal deontic rule		x			x							x
Activity precondition					x							x
Activity postcondition									x			
Dynamic integrity					x	x	x	x				x
Activity cardinality	x				x					x		x
Serial activity constraint		x			x							x
Activity order		x			x							x
Activity exclusion	x				x							x
Activity inclusion									x			
Reaction rule	x				x							x
Static integrity						x	x	x				
Derivation rule						x		x				
Activity authorization			x									
Activity allocation rule			x									
Visibility constraint												
Event subscription												

4.1.1 Data Concerns

For many organizations data quality is an important business concern. This can be put down to the fact that organizations in a service-oriented economy use information as primary resource for doing business. Organizations like insurance companies, banks, or manufacturers cannot afford mistakes because of bad data quality. An extensive automation of processes such as payroll administration, insurance claim processing, or repairs, has left little possibility for human intervention to detect and correct errors due to incorrect data or miscalculations. Technologies for ensuring data quality have been existing for decades. Such technologies range from client-side input validation, over data bases integrity constraints, to production rule engines for making complex calculations. In spite of these technologies, data quality is still an issue.

One of the root causes of poor data quality can be attributed to a strictly procedural modeling of data concerns. When data concerns are modeled procedurally, it is documented when and how a particular data concern should be verified and enforced. Such a procedural approach to modeling data concerns goes against the "declarative rule-enforcement" design principle identified in Section 2. When data concerns are declaratively modeled, they become *process-aware* in that the processes in which they are relevant can be identified. They are however not *process-driven* in that their definition is strongly connected to one process in particular. We suggest the following methodology for documenting data concerns.

1. Define or choose a domain specific **business vocabulary**. In terms of the EM-BrA^{2}CE Vocabulary, a business vocabulary consists of business concept types and business fact types. According the the SBVR standard, a business vocabulary can be shared by different business communities within and between organizations. For this reason, business vocabularies should be well-defined, domain-specific vocabularies, that must become part of an organization's every-day communication. At the same time, an organization must recognize that business vocabularies will always remain context-dependent, and thus can be inconsistent with other vocabularies. Mediation strategies are a still a dire necessity (Fensel and Bussler, 2002).

2. Define integrity constraints and derivation rules. The actual data quality concerns of a company can be verbalized as integrity constraints and derivation rules. This specification takes place prior to any implementation. Either a natural language or a formal language can be used.

3. Decide upon the **modality** of integrity and derivation rules. In correspondence to the "differentiation by modality" design principle of declarative process modeling, derivation rules and integrity constraints can be given different modalities, such as necessity, obligation, permission, prohibition, or advice. These modalities are provided by SBVR.

In the remainder of this section, we define the business rule types integrity constraint and derivation rule, indicate related work, and indicate how they relate to the declarative execution semantics of the EM-BrA^{2}CE framework.

Data aspect: static integrity constraint

The performer of an activity can perform particular manipulations (addition, removal or update) of business facts. These state transitions are among others subject to particular integrity constraints. Integrity constraints involve cardinality constraints, domain constraints, and the like.

A static integrity constraint *is* a business rule that constrains the domain over which business facts can range by expressing a logical assertion that

can, cannot, must or must not remain true (Wagner, 2003).
Example:
Necessity: each order *has* at least one order line.
Advice: the agreed price of a sales item is less or equal to the standard price of the sales item.

Integrity constraints can be operationalized by verifying whether the manipulation of a business fact (addition, removal and update) would lead to a violation of the integrity constraint. Not every integrity constraint needs to be evaluated. For instance, only those integrity constraints that range over a fact type that is currently being manipulated. In the database literature efficient algorithms have been proposed for static constraint verification (Gupta et al., 1994).

In the activity life cycle of the EM-BrA^{2}CE Framework, a static integrity constraint constrains *addFact*, *removeFact* and *updateFact* activity state transitions.

Data aspect: derivation rule

Almost any knowledge representation language allows expressing so-called derivation or deduction rules.

A derivation rule *is* a business rule that defines a business fact in terms of existing business facts (Wagner, 2003).
Example:
Advice: an order *has* a 10 percent discount if the order *is from* a loyal customer.

A derivation rule can have a deontic instead of an alethic nature. In other words, derivation rules can be SBVR:structural business rules or SBVR:operative business rules. These rule types require a different execution semantics. For instance, a business rule might recommend a particular price, but leave a salesperson with the freedom of choice of determining custom-tailored prices.

In the monotonic semantics given to SBVR conceptual models, each time facts of a fact type that is defined by a derivation rule are required (for instance in the context of evaluating another business rule) the derivation rule is consulted. Consequently, derivation rules augment the fact base (SBVR:conceptual model) during the processing of state transition requests. Reasoning paradigms such as backward chaining automatically support this execution semantics.

In the activity life cycle of the EM-BrA^{2}CE Framework, a derivation rule constrains *addFact* and *updateFact* activity state transitions.

4.1.2 Control-flow

Control-flow concerns deal with order and timing concerns that apply to the activities in business processes (Jablonski and Bussler, 1996). These sequence and timing constraints are an important aspect of business process flexibility and compliance. To date, these constraints are most often implicitly transcribed into control-flow-based process models. This implicit representation of constraints, however, complicates the verification of compliance and restrains flexibility in business process design. The reason is that procedural process models are – to quote Schmidt and Simone (1996) – a "pre-computation of activity dependencies". Without knowing the underlying business concerns, it is as difficult to make changes to processes at design-time as it is to adapt processes at execution-time.

We suggest the following methodology for declaratively documenting control-flow concerns concerns.

1. Identify the processes (or services) within or between organizations. In the vocabulary, processes are composite activity types, whereas atomic activities represent the lowest level of granularity of work that is considered to be an activity (or service). Fact types: 'activity type$_1$ *can consist of* activity type$_2$'.
2. Identify the business fact types that can be made visible or manipulated by activities (services) of a particular activity type (service capability). Fact types: 'activity type *can make visible* business fact type', 'activity type *can manipulate* business fact type'.
3. Identify sequence and timing concerns as business rules.
4. Determine the **modality** of each modeled rule.

In the remainder of this section, we enumerate a number of business rule types that can be used to verbalize sequence and timing constraints, point out related work, and indicate how they can be operationalized in the declarative execution semantics of the framework.

Control-flow: temporal deontic rule

Business policy and regulations contain a lot of implicit order and timing information. In a trade community, for instance, different business protocols might exist for engaging in a business interaction. Such business protocols lay down the obligations and permissions of all business partners in an interaction and can be expressed in the form of temporal deontic rules.

> A temporal deontic rule *is* a structural business rule that defines when deontic assignments come into existence or cease to exist based on the (non-)occurrence of events.

The rules describe behavior from a third-person perspective and can, for instance, be expressed in the PENELOPE language (Goedertier and Vanthienen, 2006d). This language is introduced in Section 4.2.

The temporal deontic rules displayed underneath categorize the external business regulation payment-after-shipment that visualized in the BPMN diagram of Figure 4.1(a). Assuming that the agents in a business interaction do not intend to violate these deontic assignment rules, the resulting permissions and obligations impose partial order constraints on the activities in a business process.

> Necessity: initially a buyer *has* the permission to perform a place order activity.
> Necessity: when a buyer completes a place order activity, the seller *has* the obligation to perform a accept order activity or a reject order activity within 2 time units.
> Necessity: when the buyer completes a place order activity, the buyer *has* the obligation to perform a pay activity within 2 time units after the seller completes the ship activity.
> Necessity: when the seller completes a accept order activity, the seller *has* the obligation to perform a ship activity within 2 time units.

In the activity life cycle of the EM-BrA^{2}CE Framework, a temporal deontic rule constrains *schedule*, *start* and *redo* activity state transitions. Temporal deontic rules indirectly affect the sequence and timing of activities. An agent who coordinates an activity will try to observe the deontic assignments that result from performing the activities. In addition, the coordinator will take into account that other agents potentially could violate the deontic assignments that are imposed on them. In particular, a coordinator will schedule the activities such that he does not violate any permissions and that the deadlines on obligations are observed. Additionally, he will take appropriate action when other agents violate the deadlines that are imposed on them.

Control-flow: activity precondition

Although any activity state transition can be constrained using preconditions, we only consider preconditions imposed on the *start* and *complete* activity transitions to be business rules. A precondition on the *start* transition is called an activity precondition:

> An activity precondition *is* a business rule that defines the conditions that are required to start an activity of a given activity type.
> Example:
> Necessity: To start an accept order activity, a place order activity has completed and no accept order or reject order activity has started.
> Necessity: To start a reject order activity, a place order activity has completed and no accept order or reject order activity has started.
> Necessity: To start a ship activity, a accept order activity has completed and no ship order activity has started.
> Necessity: To start a pay activity, a accept order activity has completed, a ship order activity has completed and no pay activity has been started.

In the Web Service Modeling Ontology (WSMO) (Roman et al., 2005), it is possible to assign a precondition to a service capability. However, preconditions can only be expressed in terms of (business) concepts. In particular, it is not possible to include event conditions or to query the properties of activities. This limitation potentially has the disadvantage that it is required to add artificial business concepts to a business vocabulary. For instance, instead of stating that an order has been accepted, it is required to refer to a potentially artificial business concept 'acceptation notice'. The same activity preconditions, expressed in terms of business concepts only, would then be formulated as:

> Necessity: To start an accept order activity, it is necessary that the order exists and does not *have* an acceptation notice or a rejection notice.
> Necessity: To start a reject order activity, it is necessary that the order exists and does not *have* an acceptation notice or a rejection notice.
> Necessity: To start a ship activity, it is necessary that the order the order *has* an acceptation notice and does not *have* a shipping order.
> Necessity: To start a pay activity, it is necessary that the order the order *has* an acceptation notice and the order *has* a proof of delivery and the order has not yet been paid.

Unlike the EM-BrA^{2}CE Framework, WSMO makes a distinction between the state of the information space and the state of the world. Because the EM-BrA^{2}CE Framework is more situated on the conceptual modeling level, no such distinction between the world and the information system is made.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity precondition constrains *start* and *redo* activity state transitions.

Control-flow: activity postcondition

A precondition on the *complete* transition is called an activity postcondition:

> A business fact postcondition *is* a business rule that specifies the abstract state of an activity of a particular activity type upon its *completion.*
> Example:
> Necessity: To complete an activity that *has type* place order, it is necessary that the order exists.
> Necessity: To complete an activity that *has type* accept order, it is necessary that the order *has* an acceptation notice.
> Necessity: To complete an activity that *has type* reject order, it is necessary that the order *has* a rejection notice.
> Necessity: To complete an activity that *has type* ship, it is necessary that the order *has* a proof of delivery.
> Necessity: To complete an activity that *has type* pay, it is necessary that there exists a proof of payment.

This constraint subsumes a *mandatory* constraint in the case handling paradigm and is similar to a *postcondition* in WSMO. The case handling paradigm allows specifying which case data types are free, mandatory or restricted with respect to performing an activity of a particular activity type (van der Aalst et al., 2005).

A **free** business fact type can be manipulated in every sub-activity. A business fact type is **mandatory** for a particular activity type, when a fact of this fact type is required for the *completion* of the particular activity. A business fact type is **restricted** to a particular (or a number of) activity type, when a fact of this fact type can only be manipulated in the context of an activity of this type. van der Aalst et al. (2005) provide a means in which these constraints can be operationalized by considering them as postconditions on the completion of a particular activity. In WSMO, it is possible to assign a postcondition to a service capability. However, the same restriction applies as with respect to preconditions.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity postcondition constrains *complete* activity state transitions.

Control-flow: reaction rule

> A reaction rule or event-condition-action (ECA) rule *is* a business rule that expresses the activities that are to be undertaken, given the (non-)occurrence of certain events and a particular condition being fulfilled.

In spite of their apparent simplicity, business processes using only reaction rules cannot be classified as being declarative process models. The reason is that process models that are composed of reaction rules only constitute an explicit execution scenario that risks to be over-specified and can be regarded to be as procedural as control-flow based models. What is needed is a hybrid approach, in which the freedom of choice that is left by other business rules is filled in – when required – by a small set of reaction rules. Therefore, reaction rules are considered among other business rules to specify behavior, but constitute by themselves no means for declarative process modeling.

The properties of relevant abstract states in the execution model of a business process can be used to describe decision points. For example, the above discussed payment-after-shipment temporal deontic rules, state that a seller has the obligation to either accept or reject an order, when a buyer places an order. Although possible from a modeling perspective, the protocol does not stipulate what the buyer must do in case the seller, for instance, rejects the order. This freedom of choice can be represented as an abstract state or decision point that is described by the following abstract state expression:

> An agent of role seller has the obligation either to accept or reject an order.

From this abstract state a number of mutually exclusive abstract states can be derived: the seller has accepted the order, the seller has rejected the order or the order times out. Reaction rules can impose a suitable reaction to each of these states:

> When an order is rejected and if the order *is critical*, then notify a purchase representative.
> When an order is rejected and if the order *is* not *critical*, then reorder

> with a different seller.
> When an order times out and if the order *is critical*, then notify a purchase representative.
> When an order times out and if the order *is not critical*, then reorder with the same seller.

In the activity life cycle of the EM-BrA^{2}CE Framework, a reaction rule defines *start* activity state transitions.

Control-flow: dynamic integrity constraint

Within the context of an activity, agents can manipulate business facts that are related to the activity. There are, however, conditions on the state change of business facts that could prevent an activity from taking place. Wagner (2003) calls such conditions dynamic integrity constraints.

> A dynamic integrity constraint *is* a business rule that defines the admissible state changes of a business fact.
> Example: After the start of a ship activity, the order lines of the order can no longer be changed.

The activities via which the manipulations occur always remain implicit in these rules as the fact type 'activity type *can manipulate* business fact type' already relates these activity types to the appropriate business fact types.

In the activity life cycle of the EM-BrA^{2}CE Framework, a dynamic integrity constraint constrains *start*, *redo*, *addFact*, *removeFact* and *updateFact* activity state transitions.

Control-flow: activity cardinality constraint

An agent that performs a composite activity, actually constructs an execution plan. This coordination work involves, among others, the creation of a number of activities. Although the composites of a coordination plan are defined by the 'Activity type$_1$ *can consist of* activity type$_2$' fact type, the fact type does not impose any restrictions on the number of such activities that may be included in the execution plan. For instance, an execution plan in the context of a handle purchase order activity might involve two separate ship activities, but it may only contain one accept order activity. Such cardinality restrictions are imposed by activity cardinality constraints.

> A activity cardinality constraint *is* a business rule that limits the number of activities of a particular activity type that occur within the context of a same parent (coordination) activity.
> Example:
> There exists exactly one place order activity$_1$ that *has parent* a handle purchase order activity$_2$.
> There exists at most one accept order activity$_1$ that *has parent* a handle sales order activity$_2$.

This business rule type can be expressed in the ConDec language with so-called existence constraints.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity cardinality constraint constrains *create*, *start* and *redo* activity state transitions.

Control-flow: serial activity constraint

When coordinating a composite activity, it is also relevant to know whether two activities can be performed concurrently. This is expressed by a serial activity constraint.

> A serial activity constraint *is* a business rule that imposes that activities belonging to a particular set of activity types must not be performed in parallel within the context of a same parent (coordination) activity.
> Example:
> Activities that *have type* place order, accept order, reject order and ship order must not be performed in parallel.

Sadiq et al. (2005), for instance, include serial activity constraints in their constraint specification framework.

A serial activity constraint constrains *schedule*, *start* and *redo* activity state transitions.

Control-flow: activity order constraint

An order constraint on two activities is a stronger condition than a seriality constraint.

> An activity order constraint *is* a business rule that imposes that activities of particular activity types must be performed in a specified order within the context of a same parent (coordination) activity.
> Example:
> An accept order activity$_1$ can only start after a place order activity$_2$ has completed.

This can, for instance, be expressed with an order constraint in the constraint specification framework of Sadiq et al. (2005) or with different types of relation constraints in the ConDec language.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity order constraint constrains *schedule*, *start* and *redo* activity state transitions.

Control-flow: activity exclusion constraint

> An activity exclusion constraint *is* a business rule that imposes that two activities of a particular activity type are mutually exclusive within the context of a same parent (coordination) activity.
> Example:
> Necessity: accept order and reject order activities are mutually exclusive.

This can, for instance, be expressed with an exclusion constraint in the constraint specification framework of Sadiq et al. (2005) or with different types of negation constraints in the ConDec language.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity exclusion constraint constrains *create*, *start* and *redo* activity state transitions.

Control-flow: activity inclusion constraint

> An activity inclusion constraint *is* a business rule that imposes that two activities of a particular activity type are mutually inclusive within the context of a same parent (coordination) activity.
> Example: Obligation: accept order and ship activities are mutually inclusive.

This can, for instance, be expressed with an inclusion constraint in the constraint specification framework of Sadiq et al. (2005) or as a co-existence constraint in the ConDec language.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity inclusion constraint constrains *complete* activity state transitions of activities that can consist of the activities mentioned in the constraint.

4.1.3 Access Control

Access control is the ability to permit or deny access to physical or informational resources. It is an organization's first line of defence against unlawful or unwanted acts. Recently, access control has become a key aspect of regulatory compliance. Access control within the EM-BrA^{2}CE framework is based on the existing standard for Role-Based Access Control (Ferraiolo et al., 2001; InterNational Committee for Information Technology Standards (INCITS), 2004; Sandhu et al., 1996). The existing RBAC standard allows specifying dynamic separation of duty (SoD) constraints. Such constraints impose that within a single session a user cannot assume two roles on which a dynamic SoD constraint applies. Strembeck and Neumann (2004) discuss an extension to the RBAC standard by allowing to express dynamic, process-related, access rules . The proposed EM-BrA^{2}CE Vocabulary goes further in that it allows expressing process-aware access control policies that can declaratively relate to the current state of a business process and to a history of related business events. The proposed vocabulary and business rules allow to specify process-aware access control in four steps:

1. Define an **access control policy**. An access control policy is a non-actionable directive that identifies security threads and safety concerns and motivates access control implementation. Metamodels such as the OMG's Business Motivation Model Object Management Group (2006a) provide the required structure to formulate access control policies.

2. Identify the access control **roles**. Roles are permissions involving the performance of activities or the involvement in activities that pertain to meaningful groups of activity types. Roles provide *stability* because role definitions do not change as often as agent-activity type assignments would. Fact types: 'role *can perform* activity type', 'activity type *can manipulate* business fact type', 'activity type *can make visible* business fact type' and 'role *can subscribe to* event type *in context of* activity type'.

3. **Specify access constraints**. Access constraints refine the role-based access control policy to take into account issues that are beyond the scope of user-role assignment. Access constraints give an access control model *precision*, because they constrain the role-based access according to the properties of the agent, the activity and the business process event history.

4. Make **agent-role assignments**. Agent-role assignment is the provisioning of agents with roles that represent access rights. Agent-role assignments can be defined by derivation rules, but for reasons of compliance and flexibility, agent-role assignments are often hard-coded. Fact type: 'agent *can have role* role'.

Access control: activity authorization constraint

An activity authorization constraint allows constraining the agent-role assignments that can be granted to an agent. For instance, the fact 'sales representative *can perform* accept order' is constrained by the rule that sales orders larger than 2000 euro cannot be reviewed by junior sales representatives.

> An activity authorization constraint *is* a structural business rule that dynamically constrains the activities that can be assigned to an agent on the basis of the properties of the activity, the business facts in its state space and the properties of the agent.
> Example: Necessity: an agent that *has age* less than 18 years can not perform a place order activity.
> Necessity: an agent that *has function* junior sales representative can not perform an accept order or reject order activity that *is identified by* an order that *has* an amount larger than 2000 euro.

A similar kind of rule has been described by Strembeck and Neumann (2004) in the context of the Role-based access control (RBAC) standard Ferraiolo et al. (2001); InterNational Committee for Information Technology Standards (INCITS) (2004); Sandhu et al. (1996).

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity authorization constraint constrains *assign* activity state transitions.

Access control specifications adhering to the role-based access control (RBAC) have a non-monotonic semantics that can be expressed in defeasible logic (Antoniou et al., 2001; Nute, 1994). Defeasible logic is a means to formulate knowledge in terms of general rules and exceptions. To this end, defeasible logic allows for rules of which the conclusions can be defeated (defeasible rules) by contrary evidence provided by strict rules, other defeasible rules and defeaters. A superiority relation between rules indicates which potentially conflicting conclusions can be overridden. In EM-BrA^{2}CE the 'agent *can perform* activity' fact type is defined using a rule set of two generic defeasible rules:

$$\begin{aligned} &r_1 : true \Rightarrow \neg CanPerform(G, A), \\ &r_2 : HasRole(G, R), HasType(A, At), CanPerform(R, At) \\ &. \qquad \Rightarrow CanPerform(G, A) \end{aligned}$$

and a number of domain-specific activity authorization constraints, that translated to the following defeasible rules:

$$\begin{aligned} &r_3 : A(3) \Rightarrow \neg CanPerform(G, A), \\ &... \\ &r_i : \mathrm{A(i)} \Rightarrow \neg CanPerform(G, A), \\ &... \\ &r_n : \mathrm{A(n)} \Rightarrow \neg CanPerform(G, A). \end{aligned}$$

The following priority relationship applies between the generic and domain-specific defeasible rules:

$$r_1 < r_2, r_2 < r_3, r_2 < r_4, ...r_2 < r_n.$$

Because activity authorization constraints all potentially defeat the 'agent *can perform* activity' conclusion, they cannot contradict each other. As a consequence, no activity authorization constraint can be specified that can invalidate the outcome of another constraint. This is a valuable result, because it facilitates the incremental specification of access control policies (Grosof et al., 1999): new rules can be added without the conditions of previous rules need to be reconsidered. Ordinary rules, in contrast, require a complete, encyclopedic knowledge of all rules to be updated or decided upon. In addition to their enhanced representational capability, efficient implementations of defeasible logic have been proposed (Maher et al., 2001) together with visualization techniques to render the added complexity of default reasoning comprehensible to end users (Bassiliades et al., 2005; Kontopoulos et al., 2006). Although defeasible logic is naturally present in the role-based access control model, the current SBVR specification does not allow for the specification of defeasible rules.

Access Control: visibility constraint

The fact type 'Activity type *can make visible* business fact type' indicates the business fact types that can be made visible in the context of an activity. It is possible to constrain the visibility of facts with visibility constraints.

> A visibility constraint *is* a structural business rule that dynamically constrains the visibility of business facts within an activity according to the properties of the activity, the business facts in its state space and the

agent that has been assigned to the activity.
Example: Coordinate purchase order *can make visible* the business fact type 'order *has* rejection notice'
Necessity: a rejection notice is only visible to an agent that *is* a corporate customer.

Organization: event subscription constraint

The fact type 'role *can subscribe to* event type *in context of* activity type' expresses the visibility of events to agents in the context of an activity. With event subscription constraints is it possible to conditionally limit the visibility of events.

A event subscription constraint *is* a structural business rule that constrains the conditions under which agents who have a particular role in the context of an activity can perceive the occurrence of an activity event.
Example: A seller *can subscribe to* completed *in the context of* ship.
A buyer *can subscribe to* completed *in the context of* ship.
Impossibility: an agent that *has role* buyer *perceives* an event that *is about* a ship activity for an order that *has* a total amount of less than 2000 euro.

When an event occurs, each agent who has a particular role in the context of the activity and whose role is subscribed to the event type and for whom no subscription constraints apply, can perceive the event. Consequently, the event is non-repudiable to external agents such that any legal obligation that results from the event can be enforced.

Organization: activity allocation rule

It is possible that agents have the authorization to perform a particular activity, but that they are not the primary designated performers of a task. Activity allocation rules are guidelines that indicate to what extent assigning an activity to an agent is desirable.

An activity allocation rule *is* an operative business rule that indicates the assigning of an activity to a particular agent as an obligation or a prohibition. In both cases a level of enforcement indicates the degree to which such an assignment is desirable.
Example: Advice: an agent that *has function* purchase department head should not be assigned to an activity that *has type* archive document.

Whereas activity authorization constraints deal with authorization, activity allocation rules deal with the fair distribution of work.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity authorization constraint constrains *assign* activity state transitions.

4.1.4 Conclusion

Documenting business concerns already yields many advantages. By making the business concerns that govern business processes explicit as business rules, it becomes possible for human end-users to assess the impact of changes to business processes at design-time. In this way, it is possible to manually verify at design-time whether business processes do not violate particular business concerns, Sadiq et al. (2007) call this "compliance by design". Knowing the hard and soft constraints that govern business processes, it furthermore becomes possible to get an idea about the degrees of freedom within which business processes can take place. In addition, changes to business policies and business regulations become traceable as their impact on business processes has been documented. This can only be to the benefit of design-time flexibility. In this section we have identified sixteen business rule types that relate to data concerns, control flow concerns, and access control. For each of these concerns, we have indicated how these can be verbalized by means of the vocabulary and business rule types of the framework and how they can be declaratively enforced within its execution semantics.

However, to realize the execution-time advantages of declarative process models, it is required that each of the identified business rules is expressed in a formal language. Formal languages allow to make inferences, and can be used for the execution, visualization, and verification of declarative process models. In the next section, we discuss a formal language for expressing temporal deontic rules.

4.2 PENELOPE

In Section 4.1.2 we have defined temporal deontic rules to be structural business rules that define when deontic assignments come into existence or cease to exist base on the (non-)occurrence of events. In this section, we introduce PENELOPE (**P**rocess **EN**tailment from the **EL**icitation of **O**bligations and **PE**rmissions), a language to express temporal deontic rules about the obligations and permissions of both internal and external agents in a business interaction, and an algorithm to generate compliant sequence-flow-based process models that can be used to visualize and validate them. Furthermore, we indicate how abductive reasoning over the event calculus allows verifying a set of temporal rules.

This section is structured as follows. First, we discus the relevant literature on applications of deontic logic. Next, we formally introduce PENELOPE. Finally, we define and illustrate the algorithm to generate compliant, control-flow based process models from a rule set of obligations and permissions.

4.2.1 Preliminaries: the Event Calculus

The Event Calculus, introduced by Kowalski and Sergot (1986), is a logic programming formalism to represent and reason about the effect of *events* and the state of the system expressed in terms of *fluents*. The Event Calculus is appealing

Table 4.2: The event Calculus axioms (Shanahan, 1997)

term	**meaning**
$\mathit{Initiates}(\epsilon, \phi, \tau)$	event ϵ initiates fluent ϕ at time τ
$\mathit{Terminates}(\epsilon, \phi, \tau)$	event ϵ terminates fluent ϕ at time τ
$\mathit{Initially}_p(\phi)$	fluent ϕ holds from time 0
$\mathit{Happens}(\epsilon, \tau)$	event ϵ happens at time τ
$\mathit{HoldsAt}(\phi, \tau)$	fluent ϕ holds at time τ
$\mathit{Clipped}(\tau_1, \phi, \tau_2)$	fluent ϕ is terminated between times τ_1 and τ_2
$\mathit{HoldsAt}(\phi, \tau) \leftarrow \mathit{Initially}_p(\phi), \neg \mathit{Clipped}(0, \phi, \tau)$	
$\mathit{HoldsAt}(\phi, \tau_2) \leftarrow \mathit{Initiates}(\epsilon, \phi, \tau_1) \wedge \tau_1 < \tau_2 \wedge \neg \mathit{Clipped}(\tau_1, \phi, \tau_2)$	
$\mathit{Clipped}(\tau_1, \phi, \tau_2) \leftrightarrow \exists \epsilon, \tau : \mathit{Happens}(\epsilon, \tau) \wedge \mathit{Terminates}(\epsilon, \phi, \tau) \wedge \tau_1 < \tau < \tau_2$	

for several reasons. For instance, the Event Calculus builds on a first-order predicate logic framework, for which efficient reasoning algorithms exist. In addition the Event Calculus has the ability to reason about time, in which fluents come to existence or cease to holds dynamically. In addition the Event Calculus not only has the ability to deductively reason about the effects of the occurrence of events events (leading to the coming into existence of fluents or the ceasing to hold), most importantly, it also has the ability of reasoning **abductively**. Abductive reasoning over the event calculus is akin to planning in artificial intelligence. In particular, abductive reasoning produces a sequence of transitions (denoted by events) that must happen for a particular fluent to hold in the future (Eshghi, 1988; Shanahan, 1997; Van Nuffelen and Kakas, 2001). For these reasons, the Event Calculus is a suitable language both for specifying the semantics of state transitions in the EM-BrA^{2}CE framework and as a planning mechanism.

Shanahan (1997) provides suitable axiomatizations of the Event Calculus, in which the Frame Problem is solved through circumscription. Table 4.2 represents the predicates and axioms we have adopted to represent and reason about obligation and permission. We have operationalized the Event Calculus in CLP(fd), a sub-language of Prolog, which allowed us to experiment with protocol verification and the transformation to decentralized business process descriptions. Like other Prolog-implementations of the Event Calculus, the frame problem is inexistent, because of the closed-world assumption attached to Prolog's negation-as-failure.

4.2.2 The PENELOPE Language

Deontic logic is a logic for representing and reasoning about deontic concepts such as obligation, permission, prohibition and waived obligation. Various axiomatizations of deontic logic have been proposed with considerable extensions to Føllesdal and Hilpinen's Standard Deontic logic (SDL) (Føllesdal and Hilpinen, 1971). Broersen et al. (2004) use computational tree logic (CTL) (Clarke et al., 1993) to express the notion of deadline obligations. Several authors have built a Deontic Logic Knottenbelt and Clark (2004); Marín and Sartor (1999); Paschke and Bichler (2005); Yolum and Singh (2004) using the Event Calculus formalism, for which Shanahan (1997) provides suitable axiomatizations. In these works deontic properties are represented as fluents, such that it is possible to represent

Table 4.3: The PENELOPE axioms

term	**meaning**
$Xor(\alpha_1, \alpha_2)$	compound activity $alpha_1$ XOR $alpha_2$
$Or(\alpha_1, \alpha_2)$	compound activity $alpha_1$ OR $alpha_2$
$And(\alpha_1, \alpha_2)$	compound activity $alpha_1$ AND $alpha_2$
$Oblig(\pi, \alpha, \delta)$	agent π must do activity α by due date δ
$Perm(\pi, \alpha, \delta)$	agent π can do activity α prior to due date δ
$CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2)$	agent π must do activity α_2 by due date δ_2 after activity α_1 is performed prior to due date δ_1
(1) $Terminates(\alpha, Oblig(\pi, \alpha, \delta), \tau) \leftarrow \tau \leq \delta$	
(2) $Terminates(\alpha, Perm(\pi, \alpha, \delta), \tau) \leftarrow \tau \leq \delta$	
(3) $Happens(Violation(Oblig(\pi, \alpha, \delta)), \delta) \leftarrow$ $HoldsAt((Oblig(\pi, \alpha, \delta)), \delta) \wedge \backsim Happens(\alpha, \delta)$	
(4) $Initiates(\alpha_1, Oblig(\pi, \alpha_2, \delta_2), \tau) \leftarrow$ $\tau \leq \delta_1 \wedge HoldsAt(CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2)), \tau)$	
(5) $Terminates(\alpha_1, CC(\pi, \alpha_1, \delta_1, \alpha_2, \delta_2), \tau) \leftarrow \tau \leq \delta_1$	

and reason about the effects of activities on the obligations and permissions of actors.

Table 4.3 enumerates the deontic fluents and axioms of the PENELOPE language. In order to distinguish necessity from possibility in business policy and regulations, the PENELOPE language considers the deontic modalities of obligation, conditional commitment and permission, whereas other languages only consider commitments and conditional commitments (Knottenbelt and Clark, 2004; Yolum and Singh, 2004). PENELOPE does not consider prohibition or waived obligation. Prohibition is assumed, however, if neither permission nor obligation can be derived. The exclusion of prohibition and waiver prevents a lot of anomalies that often occur when several deontic rules lead to conflicting inferences. Some implementations of Deontic logic interpret deontic assignments as the obligation to bring about a certain proposition, others see it as the obligation to perform a certain activity. Because PENELOPE aims at entailing process models from deontic assignments, activities rather than propositions are the object of deontic assignments. This also allows us to model compound activities such as $Xor(AcceptOrder, RejectOrder)$. Unlike other languages, PENELOPE allows to explicitly define deadlines on the performance of activities in terms of the performance of previous activities. When an agent performs an obligation or permission within due time, the permission or obligation ceases to exist. This is expressed in axioms (1) and (2). Conversely, not performing an obligation within due date leads to a violation, as described in axiom (3). Business policies or regulations might provide so-called reparation, or contrary-to-duty, obligations to deal with violations (Governatori, 2005). Because we want to capture the semantics of both external regulations and internal business policy, we need to express deontic assignments to both external and internal agents involved in a business interaction. Internal agents in business policies are subordinate to the external agent they represent. Activities performed by an internal agent resort in the same deontic fluents as if they were performed by the representative external agent.

In Table 4.4, we give an example to demonstrate the intuition behind PENE-

Table 4.4: Payment-after-shipment

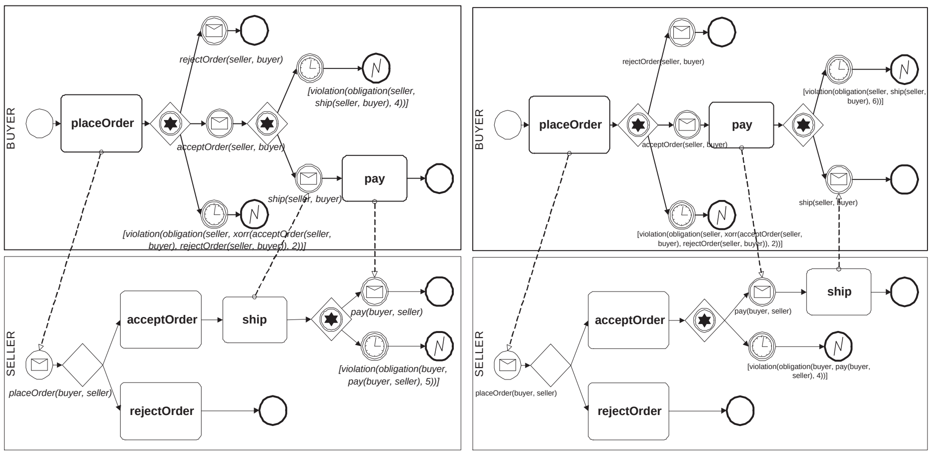
natural and formal expression
(1) Initially the buyer has the permission to place an order.
$\mathit{Initially}_p(\mathit{Perm}(\mathit{Buyer}, \mathit{PlaceOrder}(\mathit{Buyer}, \mathit{Seller})))$
(2) When the seller accepts the order, the buyer is committed to pay the seller, one time unit after the seller ships.
$\mathit{Initiates}(\mathit{AcceptOrder}(\mathit{Seller}, \mathit{Buyer}), \mathit{CC}(\mathit{Buyer}, \mathit{Ship}(\mathit{Seller}, \mathit{Buyer}), \delta_s, \mathit{Pay}(\mathit{Buyer}, \mathit{Seller}), \delta_s + 1), \tau)$
(3) When the buyer places an order, the seller must either accept or reject it within one time period
$\mathit{Initiates}(\mathit{PlaceOrder}(\mathit{Buyer}, \mathit{Seller}), \mathit{Oblig}(\mathit{Seller}, \mathit{Xor}((\mathit{AcceptOrder}(\mathit{Seller}, \mathit{Buyer}), \mathit{RejectOrder}(\mathit{Seller}, \mathit{Buyer})), \tau + 1), \tau)$
(4) When the seller accepts the order, the seller must ship within two time units.
$\mathit{Initiates}(\mathit{AcceptOrder}(\mathit{Buyer}, \mathit{Seller}), \mathit{Oblig}(\mathit{Seller}, \mathit{Ship}(\mathit{Seller}, \mathit{Buyer}), \tau + 2), \tau)$
(5) Only when sales accepts the order, dispatch may ship the order
$\mathit{Initiates}(\mathit{AcceptOrder}(\mathit{Sales}, \mathit{Buyer}), \mathit{Perm}(\mathit{Dispatch}, \mathit{Ship}(\mathit{Dispatch}, \mathit{Buyer}), \delta), \tau)$

LOPE. In an order-to-cash business process the external roles of $Buyer$ and $Seller$ may be distinguished. A seller can have, among others, the internal roles of $Sales$, and $Dispatch$. In addition, the following externally visible activity types could exist: *PlaceOrder*, *AcceptOrder*, *RejectOrder*, *Pay* and *Ship*. For these roles and activities a number of temporal deontic rules are displayed in the table. Assignments 1 to 4 categorize the external business regulation payment-after-shipment, specifying that payment takes place after shipment. Assignment 5, however, is an internal business policy, specifying that no order may be shipped without previously being accepted. These permissions and obligations impose partial order constraints on the activities in a business processes. This set of deontic assignments leads to the process model for both seller and buyer that is displayed in Figure 4.1(a).

4.2.3 Validation by Visualization

Declarative process modeling allows including a lot of functional and non-function aspects of business processes that is missing in procedural process models. However, a declarative process model in the form of a state description and a set of business rules can also be more difficult to understand than the graphically appealing process notations of the Business Process Modeling Notation (BPMN) (Object Management Group, 2006b) and UML Activity Diagrams (Object Management Group, 2005). In this section we introduce an algorithm to generate the state space of a set of temporal deontic rules. This state space can be used for verification of anomalies. In addition, this state space can be mapped to control-flow-based process models for each of the business partners in the interaction. Generated process models are not intended for process execution, but can rather be used by the process designer for validation and allow to identify the decision points and possible violations of obligations that can occur.

To generate the process model for a role in a business interaction, one must analyze the temporal obligations and permissions that hold at certain points in


BUYER
placeOrder
rejectOrder(seller, buyer)
acceptOrder(seller, buyer)
[violation(obligation(seller, ship(seller, buyer), 4))]
ship(seller, buyer)
pay
[violation(obligation(seller, xorr(acceptOrder(seller, buyer), rejectOrder(seller, buyer)), 2))]
SELLER
acceptOrder
ship
rejectOrder
placeOrder(buyer, seller)
pay(buyer, seller)
[violation(obligation(buyer, pay(buyer, seller), 5))]
[violation(obligation(seller, ship(seller, buyer), 6))]
[violation(obligation(buyer, pay(buyer, seller), 4))]

(a) payment-after-shipment (b) shipment-after-payment

Figure 4.1: Two process models visualized by PENELOPE

time, given certain narratives of activity performances N. To this end, the expressions below define sets of obligations $O(\tau)$ and permissions $P(\tau)$ that hold at state τ, sets of obligations $Od(\delta)$ and permissions $Pd(\delta)$ that are due at state δ and a set of violations without reparation $VWR(\tau)$ that happen at state τ. Notice that a narrative of activity performances N is implicitly assumed in each of these expressions.

$$\begin{aligned}
O(\tau) &\in \tau \rightarrow && \{\alpha : \mathit{HoldsAt}(\mathit{Oblig}(\pi, \alpha, \delta), \tau)\} && (4.1)\\
P(\tau) &\in \tau \rightarrow && \{\alpha : \mathit{HoldsAt}(\mathit{Perm}(\pi, \alpha, \delta), \tau)\} && (4.2)\\
Od(\delta) &\in \delta \rightarrow && \{\alpha : \mathit{HoldsAt}(\mathit{Oblig}(\pi, \alpha, \delta), \delta)\} && (4.3)\\
Pd(\delta) &\in \delta \rightarrow && \{\alpha : \mathit{HoldsAt}(\mathit{Perm}(\pi, \alpha, \delta), \delta)\} && (4.4)\\
VWR(\tau) &\in \tau \rightarrow && \{\alpha : \mathit{Happens}(\mathit{Violation}(\mathit{Oblig}(\pi, \alpha, \delta)), \tau), && \\
& && \neg\exists \mathit{Initiates}(\mathit{Violation}(\mathit{Oblig}(\pi, \alpha, \delta)), o, \tau)\} && (4.5)
\end{aligned}$$

A state τ in our state space corresponds to a set of obligations $O(\tau)$, permissions $P(\tau)$ and conditional commitments $CC(\tau)$ that hold this state. State transitions are defined differently in PENELOPE than in the commitment space defined by Yolum and Singh (2004). In PENELOPE a business interaction can move from a state τ_1 to a state τ_2 if there exists a narrative N of permissible activity performances, between states τ_1 and τ_2, such that the performance of the activities makes the same deontic fluents hold at state τ_2 that are contained by state τ_2. Under the assumption that no cycles can occur in the interaction, the state space can be represented as a directed acyclic graph. To efficiently enumerate a state space beyond a state τ, it suffices to perform all different combinations

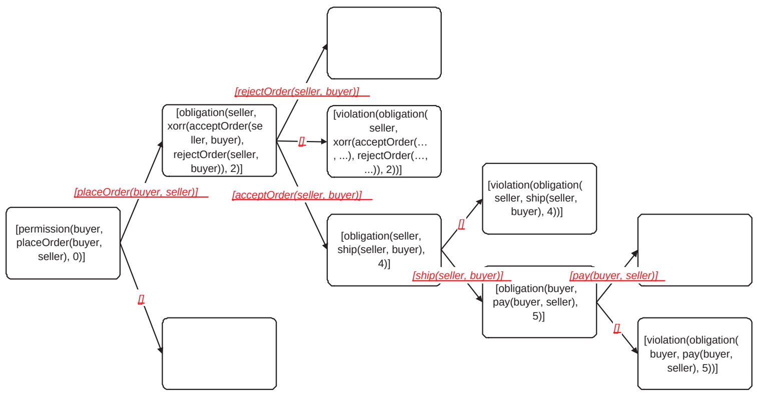

[rejectOrder(seller, buyer)]
[obligation(seller, xorr(acceptOrder(seller, buyer), rejectOrder(seller, buyer)), 2)]
[]
[violation(obligation(seller, xorr(acceptOrder(…, …), rejectOrder(…, …)), 2))]
[placeOrder(buyer, seller)]
[acceptOrder(seller, buyer)]
[permission(buyer, placeOrder(buyer, seller), 0)]
[]
[violation(obligation(seller, ship(seller, buyer), 4))]
[]
[obligation(seller, ship(seller, buyer), 4)]
[ship(seller, buyer)]
[pay(buyer, seller)]
[obligation(buyer, pay(buyer, seller), 5)]
[]
[]
[violation(obligation(buyer, pay(buyer, seller), 5))]

Figure 4.2: The state space of the example deontic assignments

of permissible activity performances that are due at the earliest due date of the obligations and permissions that hold in state τ. Figure 4.2 enumerates the state space of the deontic assignments of the order-to-cash business process in Table 4.4. A state τ is an end state if no obligations or permissions hold at τ or if there exist violations without reparation.

$$endState(\tau) \Leftrightarrow O(\tau) \cup P(\tau) = \emptyset \vee VWR(\tau) \neq \emptyset \tag{4.6}$$

Temporal deontic assignments to internal and external agents in a business interaction impose *partial* order constraints on the activities that are carried out. The problem of generation the control-flow for a particular role in a set of temporal deontic rules can either be *under specified*, *even specified* or *over specified*. A problem is under specified if no unique sequence flow can be entailed. For even and over specified problems, a unique sequence flow can be derived, provided that the deontic assignments contain no anomalies such as livelocks, deadlocks or contradictions. Given rules 1 to 4 in the example of Table 4.4, the generation problem is even specified. Adding rule 5 makes the problem over specified, but introduces no contradictions.

We have implemented the PENELOPE language in CLP(fd). In addition we have constructed an algorithm in Prolog to generate a proprietary XML file with the BPMN process model for all external roles in a set of deontic assignments. From this XML file a Microsoft Visio Add-in was written to draw the generated model. We have chosen the BPMN because its visualizations allow us to model external events and exceptions in control-flow. We make use of well-understood and general control flow constructs such as sequence, XOR-split, and AND-split.

The algorithm is limited to acyclic state spaces. A summary of the algorithm is displayed in pseudo-code below. It progressively enumerates all states in the state space and draws the BPMN model for role π. Whenever during state transitions the role π performs activities, this is modeled as a task. Whenever another role performs an activity of which π is a recipient, this is modeled as a message event.

The drawing logic of the algorithm is represented by a large number of IF-THEN rules. In the algorithm the obligations and permissions of role π that are due at state δ are contained by the sets $Od(\pi, \delta)$ and $Pd(\pi, \delta)$. A generated process model for a particular role π must not violate the obligations for which no violation is allowed. Therefore, whenever the set $PO(\pi, \delta)$ contains obligations to fulfil, these are drawn as tasks in BPMN. By way of precaution, a generated process model for a particular role π must foresee the possibility that other business partners violate obligations. For instance, when a buyer places an order, he must foresee never to receive a rejection or acceptance from the seller. Violations of obligations can only be detected if the due dates on obligations are timed during process enactment. This is represented in the BPMN model using intermediate timeout events. The obligations and permissions towards role π due at time δ are contained in the sets $OTP(\pi, \delta)$ and $PTM(\pi, \delta)$. In a state in which a violation occurs for which no reparation exists, an error end event is drawn. Notice that process design can only identify exceptions, it is up to the process modeler to properly deal with them. In some cases, the deontic conflicts between business partners might be resolved through human interaction.

```
$PO(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Oblig(\pi, \alpha, \delta), \delta)\}$
$PP(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Perm(\pi, \alpha, \delta), \delta)\}$
$OTP(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Oblig(\phi, \alpha, \delta), \delta), recipient(\alpha) = \pi\}$
$PTP(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Perm(\phi, \alpha, \delta), \delta), recipient(\alpha) = \pi\}$
$OO(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Oblig(\phi, \alpha, \delta), \delta), \phi \neq \pi\}$
$OP(\pi, \delta) \in \pi \times \delta \rightarrow \{\alpha : HoldsAt(Perm(\phi, \alpha, \delta), \delta), \phi \neq \pi\}$
**drawControlFlow**$(\pi, \tau)$
**if** $\neg endState(S(\tau))$ **then**
  $\delta := earliestDueDate(\tau)$
  **if** $\{\alpha : \alpha \in PO(\pi, \delta), atomic(\alpha)\} \neq \emptyset$ **then** draw tasks in sequence
  **if** $\{and(\alpha_1, \alpha_2) : and(\alpha_1, \alpha_2) \in PO(\pi, \delta)\} \neq \emptyset$ **then** draw tasks in parallel
  **if** $\exists xor(\alpha_1, \alpha_2) \in PO(\pi, \delta)$ **or** $PP(\pi, \delta) \neq \emptyset$ **then** draw XOR gateway
  $ACs := allCombinations(OO(\pi, \delta) \cup OP(\pi, \delta) \cup PP(\pi, \delta))$
  **forall** $AC \in ACs$
    $As := AC \cup PO(\pi, \delta)$
    **if** $\exists \alpha : \alpha \in As, \alpha \in xor(\alpha_1, \alpha_2), xor(\alpha_1, \alpha_2) \in PO(\pi, \delta)$ **then** draw task $\alpha$
    **if** $\exists \alpha : \alpha \in As, atomic(\alpha), \alpha \in PP(\pi, \delta)$ **then** draw (start event and) task $\alpha$
    **if** $\exists \alpha : \alpha \in As, \alpha \in xor(\alpha_1, \alpha_2), xor(\alpha_1, \alpha_2) \in PP(\pi, \delta)$
      **then** draw (start event and) task $\alpha$
    **if** $\exists \alpha_1, \alpha_2 : \alpha_1 \in As, \alpha_2 \in As, and(\alpha_1, \alpha_2) \in PP(\pi, \delta)$
      **then** draw (start event and) tasks $\alpha_1, \alpha_2$ in parallel
    **if** $OTP(\pi, \delta) \cup PTP(\pi, \delta) \neq \emptyset$ **then** (draw event gateway)
    **if** $\exists \alpha : \alpha \in OTP(\pi, \delta), \alpha \in As$ **then** draw start/intermediate event $\alpha$
    **if** $\exists \alpha : \alpha \in OTP(\pi, \delta), \alpha \notin As$ **then** draw intermediate timeout event $\alpha$
    **if** $\exists \alpha : \alpha \in PTP(\pi, \delta), \alpha \in As$ **then** draw start/intermediate event $\alpha$
```

```
      perform activities $As$
      drawControlFlow($\pi, \delta$)
      revoke activities $As$
    **end forall**
  **else**
    **if** $\{\nu : \nu \in VTM((\pi, \delta)\} \neq \emptyset$ **then** draw error end event
    **if** $\neg\exists\nu : \nu \in VTM((\pi, \delta)$ **then** draw end event
  **end if**
```

4.2.4 Verification

Temporal deontic assignments lay down the rules of interaction either between business partners or among the internal agents of a business partner in particular. Because temporal deontic rules are the starting point for the design of a business' private business processes they must be verified and validated. The rich semantics and the availability of efficient reasoning procedures present new opportunities for verification and validation. Without going into detail, we can highlight deadlock, livelock, deontic conflict, temporal conflict and trust conflict verification issues.

In a business interaction each legal scenario must lead to **termination**, a state in which no obligations or permissions exist. In a **deadlocks** situation, no permissible activity performance can carry the business interaction forward such that a new state of permissions, obligations and conditional commitments exist. Such a scenario might consist of two business partners having conditional commitments towards each other, but the conditional activity performance to turn at least one of these conditions into a base-level obligation is not permitted. For example, the buyer has made the conditional commitment to pay upon delivery, whereas the seller has made the conditional commitment to deliver upon payment. In a **livelock** situation, the protocol state is trapped in an infinite loop. Notice that it is not the occurrence of a loop that defines the livelock, but the occurrence of loops without a permissible activity performance that leads to a deontic state outside the loop.

Deontic conflicts arise when there are protocol states in which a business partner has both the permission and the prohibition to an activity performance or when he has both an obligation and obligation waiver to an activity performance. Note, however, that it is not possible to have deontic conflicts in PENELOPE, because it does not make use of prohibition and waiver modalities. **Temporal conflicts** occur when two deontic assignments at the same time initiate and terminate a permission, obligation or conditional commitment. Such deontic conflicts could be detected through state space enumeration, as discussed in the previous section. A declarative approach to verification could consist of abductive reasoning in the Event Calculus. The ASystem, a system for abductive reasoning within the framework of Abductive Constraint Logic Programming (Van Nuffelen and Kakas, 2001), could for instance be used for detecting conflicting states. The issue here is to ask the reasoner if it can come up with a narrative of activity performances such that a protocol state arises in which such a deontic conflict exist.

In a business interaction **trust conflicts** can also occur. This happens when a business interaction puts the business in a position were it has direct obligations towards non-trusted business partners that involve sensitive activities such as payment or the shipment of goods, that are not neutralized by preceding activity performances of the opposite party.

4.2.5 Conclusion

The sequence and timing concerns on the activities in business processes, known as control flow, are an important aspect of compliance to business policy and regulations. In this section, we have presented the PENELOPE language that is capable of *declaratively* capturing these concerns, with the purpose of (re)using them in business process design.

The PENELOPE language is a declarative process modeling language, because it adheres to the design principles for declarative process languages outlined in Section 2.2. In particular, PENELOPE allows to make the sequence and timing concerns on the activities within business processes explicit, without specifying how and when these concerns must be enforced. Furthermore, PENELOPE is not concerned with the details of how the performance of activities is communicated to other process participants. As a declarative process modeling language, PENELOPE also adopts a third-person perspective on the modeling of business concerns. Rather than modeling precedence relations for one business partner in a particular process, PENELOPE focuses on what *can* or *must* be done at certain points in time, by *all* business partners, in order to achieve their business goals, without considering one business process model in particular. This third-person perspective on the modeling of sequence and timing constraints makes it possible for external deontic assignments to be shared among process designers of different organizations. Furthermore PENELOPE has an activity-level granularity rather than a process-level granularity. In particular, temporal deontic rules are autonomous units of business logic that hold in general rather than for one particular business process. A such, changes to sequence and timing aspects in business policy and regulations are translated into changes to particular temporal deontic rules that potentially constrain multiple business process models.

From a specification of temporal deontic rules, we have indicated how it is possible to generate process models that visualize the control flow aspects of individual roles within a business interaction. The generated process models are not intended as a means for process execution, but can rather be used by the process designer for the purpose of validation. Moreover, this explicit generation of control flow can be used to identify the *freedom of choice* that is left by the sequence and timing constraints. It is up to the designer of the process to decide whether this freedom of choice is to be filled in at design-time or at runtime. In addition, the automatic generation of control flow contains an enumeration of all possible violations of obligations by other agents that allows the process designer to anticipate exceptions in current business process design.

4.3 Declarative Simulation Models

Alves de Medeiros and Günther (2005) show how CPN Tools (Jensen, 1993, 1996) can be used for generating artificial event logs. Using this idea, Alves de Medeiros (2006) has generated a large amount of artificial event logs from CPN Tools simulation models of reference problems in the domain of process discovery. In the latter work, all these reference problems have been encoded as pure colored Petri net (CP-Net) models. However, CPN Tools can also be used to simulate declarative process models. This has first been demonstrated by Günther and van der Aalst (2005) who model the semantics of the case handling paradigm. In Section 3.3 of the previous chapter, we have generalized this approach and specified the execution semantics of the EM-BrA^{2}CE Framework in terms of a CP-Net.

Having defined the execution semantics of the EM-BrA^{2}CE Framework in terms of a CP-Net, it now is possible to define a simulation model of a declarative process model, only by specifying the fact types that span the state space and transition constraints that apply in this state space. In this section, we briefly discuss the declarative simulation models of two fictitious processes: a credit application process and a driver's license application process. The models contain control flow concerns as well as access control concerns and serve as a proof-of-concept of the EM-BrA^{2}CE Framework. First, we discuss the concept types and fact types that span the state space. Then we show how the control flow and access control concerns can be declaratively defined as ML functions. Finally, we briefly discuss the structure of the event logs that are obtained by running the simulation models. These event logs are used in the remainder of this text for the purpose of re-discovering the modeled business concerns.

4.3.1 State Space

The state space of a declarative process model can be specified by defining its roles, activity types, and business concept types. The **credit application** process is a fictitious process of a customer applying for a loan with a financial institution. Below this paragraph, we indicate the roles, activity types, and business concept types have been included for this process. The `handleCreditApplication` activity type is a composite activity type, which represents the coordination of the entire credit application process. To bootstrap the simulation model, it starts from an already active activity of type `root`.

```
colset ACTIVITYTYPE = subset STRING with ["handleCreditApplication",
"applyForCredit", "checkDebt", "checkIncome", "reviewCredit",
"makeProposal", "rejectProposal", "acceptProposal", "reviewCollateral",
"changeApplication", ..., "closeApplication", "root"];
colset NOUNCONCEPTTYPE = subset STRING with ["agent","creditApplication"];
colset VERBCONCEPTTYPE = subset STRING with ["beneficiary","applicant",
"collateralType","loanType","duration","amount","debt","income",...,"risk"];
colset FACTTYPE = union nount:NOUNCONCEPTTYPE + verbt:VERBCONCEPTTYPE;
colset ROLE = subset STRING with ["customer", "employee", "expert", "teller"];
```

As displayed in Figure 4.3, these types appear as tokens in the generic simulation model.

The **Driver's License** process is a fictitious process of a future driver applying for a license. It is an extended version of the Driver's License simulation of Alves de Medeiros (2006). In particular, the process has been extended with a loop construct and an activity type `obtainSpecialInsurance`. Activities of the later activity type can occur in concurrently with activities of the type `doTheoreticalExam`. Because we only want to include control flow concerns in this example, no business concept types have been included.

```
colset ACTIVITYTYPE = subset STRING with ["start","applyForLicense",
"obtainSpecialInsurance","attendClassesCars","attendClassesMotorBikes",
"doTheoreticalExam","doPracticalExamCars","doPracticalExamMotorBikes",
"getResult","receiveLicense","end","root"];
```

4.3.2 Control Flow Concerns

Control flow concerns can, among others, be modeled as activity preconditions and activity postconditions. These business rule types mainly affect the `start` and `complete` transition types. Noteworthy is the way activity preconditions are encoded. In particular we use a special event operator that queries the event history. This operator is called the "completed-without-sequel" event operator *NS*. This is worthwhile mentioning, because this event operator can be used to logically represent the semantics of classical Petri nets. It is used in this section for the simulation of classical Petri nets in a declarative simulation model, and later in Chapter 5 as part of the language bias of our process discovery algorithm to rediscover classical Petri nets from event logs. In CPN ML, this event operator can be defined as follows.

```
fun NS(
    at1:ACTIVITYTYPE, (*the activity type that must be completed... *)
    bidcs1:FACTLIST,      (*... and its business identifier*)
    at2:ACTIVITYTYPE, (*the activity type that must not have occurred...*)
    bidcs2:FACTLIST,      (*... and its business identifier*)
    eh:EVENTLIST           (*the event history*)
):BOOL =
let
val completed =
eventsAfter(eh,bidcs1,at1,["completed"],bidcs1,at1,["aborted",...]) <> []
val withoutsequel =
eventsAfter(eh,bidcs2,at2,["started"],bidcs1,at1,["completed"]) = []
in
completed andalso withoutsequel
end
```

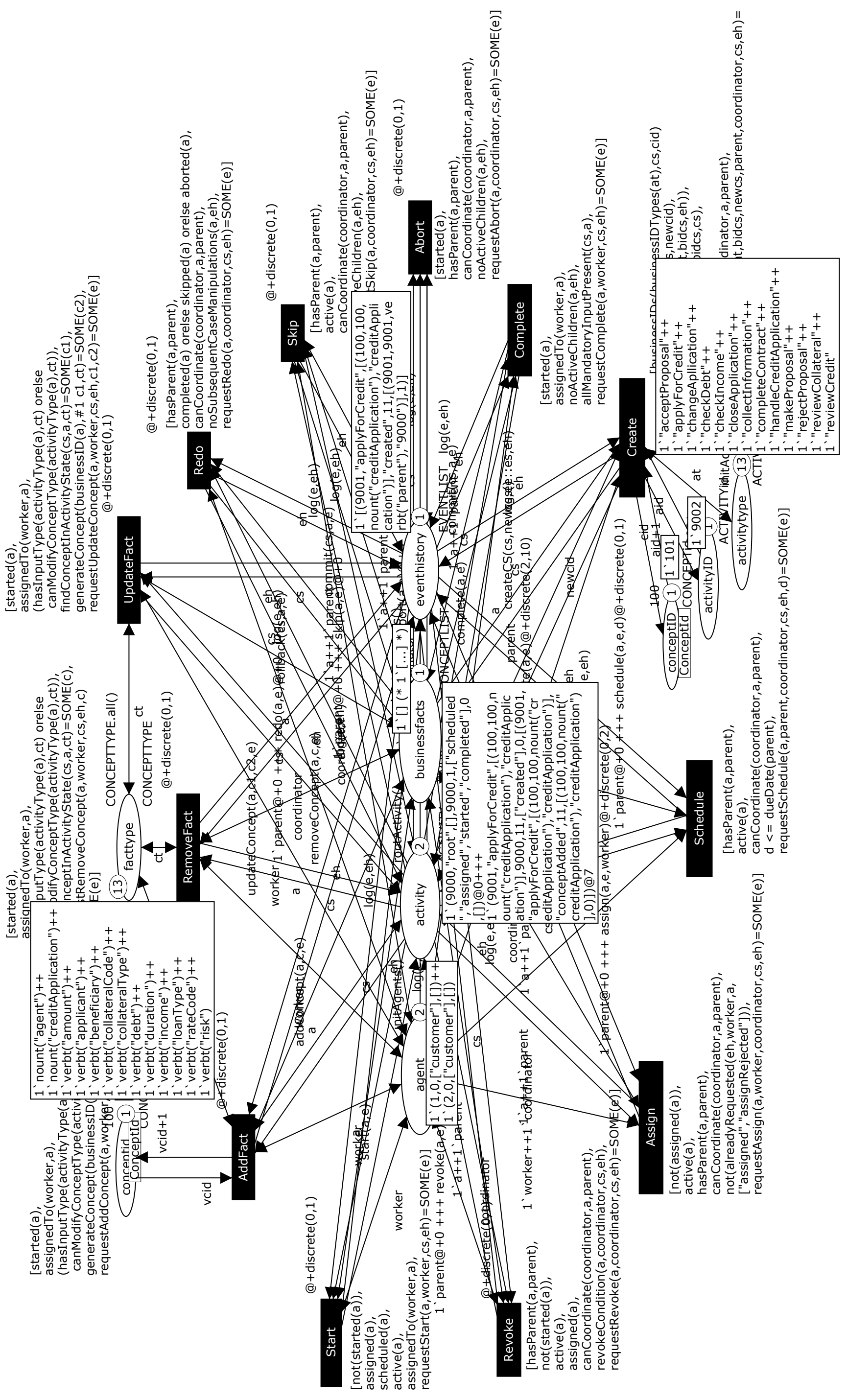


Figure 4.3: A running CP-Net simulation of the Credit Application example

The *NS* operator expresses that in the event history a `completed` event has been recorded for an activity of type `at1` with business identifiers `bidcs1`. Furthermore, it expresses that after this event, there is no event in the event history that indicates that an activity of type `at2` with business identifiers `bidcs2` has already started.

The *NS* operator can be used the express the preconditions of an activity type. The following boolean function, for instance, determines the conditions under which an activity of type `applyForLicense` can start. Notice that the function consists of a combination of *NS* literals. In Figure 5.8 shows a Petri net representation of the Driver's License example, and the logical representation in terms of *NS* literals.

```
fun startApplyForLicense(
    bidcs:FACTLIST,          (*the business id*)
    agent:AGENT,               (*the coordinator*)
    cs:FACTLIST,               (*the current fact base*)
    eh:EVENTLIST             (*the event history*)
):BOOL =
nbApplicationsDriver(bidcs,eh) < 3 andalso
(    NS("start",bidcs,"applyForLicense",bidcs,eh)
orelse
( NS("getResult",bidcs,"receiveLicense",bidcs,eh) andalso
NS("getResult",bidcs,"applyForLicense",bidcs,eh) andalso
NS("getResult",bidcs,"end",bidcs,eh)  )     )
```

Figure 4.4: Driver's license – simulated activity precondition

4.3.3 Access Control Concerns

Access control concerns affect the *assign* transition type. An example of such an access rule for the credit application can be verbalized as follows:

> "Agents that have a seniority greater than or equal to five years have a role 'senior' and this role can perform the activity 'review credit' ".
> "Agents from the department risk control have a role 'risk advisor' and this role can perform the activity 'review credit' ".
> "Within the context of a credit application, an agent cannot perform the activity 'review credit' when the agent also has performed the activity 'check debt' ".
> "Within the context of a credit application, an agent cannot perform the activity 'review credit' when the agent is the applicant of the credit application."

The boolean function `authorizationreviewCredit` in Figure 4.5 specifies this authorization logic. Notice that the defeasible semantics of role-based access control that are contained in the above rules were translated into one boolean function.

```
fun authorizationreviewCredit(
    agent:AGENT,          (*the worker to be authorized*)
    bidcs:FACTLIST,      (*the business identifier of the activity*)
    cs:FACTLIST,          (* the current business fact base*)
    eh:EVENTLIST         (* the current event history*)
):BOOL =
if (seniority(agent) >= 5 orelse fromDepartment(agent,"risk_control"))
then (
case (applicantID(bidcs,cs),applicant(bidcs,cs)) of
(SOME(applicantID),SOME(applicant)) =>
    (applicantID <> #1 agent) andalso
    not(hasDone(agent,[applicant],"checkDebt",eh))
| _ => false
)
else false
```

Figure 4.5: Credit application – simulated authorization rule

4.3.4 The Obtained Event Logs

CPN Tools simulations can be used to generate event logs (Alves de Medeiros and Günther, 2005). In the sample event logs in Table 4.5, the activity life cycle of two activities within a credit application process is represented. The column 'fact list' represents a list of facts that have been modified by the occurrence of the state transition the event reports about. For instance, in the first row the fact list '[fact(a91,parent,a90)]' reflects that with the *created* event the activity a91 was created and has the activity a90 as parent. In addition to the positive event *assigned*, the simulation log also contains negative events of the event type *assignRejected*. The availability of both positive and negative events allows to learn the transition constraints of each transition type. This is discussed in the next section.

Table 4.5: Credit application – event log

activity	activity type	business ID	event type	agent	fact list	time
a91	applyForCredit	c100	created	ag0	[fact(a91,parent,a90)]	0
a91	applyForCredit	c100	factAdded	ag0	[fact(c100,type,application)]	0
a91	applyForCredit	c100	assigned	aga0	[fact(a91,assignedTo,ag1)]	12
			...			
a91	applyForCredit	c100	factAdded	ag1	[fact(c100,beneficiary,ag1)]	14
a91	applyForCredit	c100	factAdded	ag1	[fact(c100,applicant,ag1)]	15
a91	applyForCredit	c100	factAdded	ag1	[fact(c100,loanType,bullet)]	15
a91	applyForCredit	c100	completed	ag1	[]	17
			...			
a96	reviewCredit	c100	created	ag0	[fact(a96,parent,a91)]	56
			...			
a96	reviewCredit	c100	assignRejected	ag0	[fact(a96,assignedTo,ag10)]	71
a96	reviewCredit	c100	assigned	ag0	[fact(a96,assignedTo,ag8)]	78
a96	reviewCredit	c100	factAdded	ag8	[fact(c100,risk,low)]	80
a96	reviewCredit	c100	completed	ag8	[]	81

4.4 Declarative Process Mining

Process mining is the automated construction of process models from information system event logs (Agrawal et al., 1998; van der Aalst et al., 2003). Whereas interviews provide a subjective, qualitative source of information, event logs are an objective and quantitative source of information. The analysis of event logs can reveal how business processes *actually* have taken place. It is a new and promising way of acquiring insights into business processes. Process mining is particularly useful in the context of human-centric processes that are supported, but not fully controlled by information systems (van der Aalst, 2007). Many organizations have information systems that keep track of event logs. This omni-presence of event logs, makes process mining and process analysis techniques almost as widely applicable as process modeling.

Table 4.6 explorers the process analysis space along two dimensions and positions the subsequent work in this text within this space. Currently, many algorithms have been developed to describe or predict control-flow, data, or resource-related aspects of processes from event logs . An important but difficult learning task is the discovery of sequence constraints from event logs. This task is called process discovery (Alves de Medeiros et al., 2007; van der Aalst et al., 2004). Other process learning tasks involve, among others, learning allocation policies (Ly et al., 2005) and revealing social networks (van der Aalst et al., 2005; van der Aalst and Song, 2004) from event logs. All of the aforementioned references describe new algorithms that have been implemented in the ProM Framework (van Dongen et al., 2005). In analogy with the WEKA toolset for data mining (Witten and Frank, 2000), the ProM Framework consists of a large number of plugins for the analysis of event logs (Process Mining Group, TU/Eindhoven, 2008).

In the remainder of this section, we position different process mining tasks within the EM-BrA^{2}CE Framework and characterize declarative process mining.

Table 4.6: An Overview of Process Mining Space (van der Aalst et al., 2003; van der Aalst and Weijters, 2004)

	control flow	data flow	resource flow
discovery	Chapter 5	-	Section 5.9
conformance	Section 6.1	-	-
extension	-	-	-

4.4.1 Process Mining Tasks

In this section, different process mining tasks are represented as classification problems that model the conditions under which a transition can take place (a positive event) or not (a negative event). In general, **classification learning** is learning how to assign an instance to a predefined class or group according to its known characteristics. The result of classification learning is a model that makes it possible to classify future instances based on a set of specific characteristics in an automated way. Classification techniques are often used for credit scoring (Baesens et al., 2003; Martens et al., 2007a), and medical diagnosis (Pazzani et al., 2001).

Ferreira and Ferreira (2006) show how process discovery can be formulated as a classification learning problem that determines the conditions under which an activity can take place (a positive event) or not (a negative event). This idea can be generalized towards the state transition types that are considered by the EM-BrA^{2}CE Framework or by the transactional model of the MXML format for event logs (van Dongen and van der Aalst, 2005a). A **negative event** reports that a state transition could not take place at a particular moment in the event log history. For each positive activity event type one can think of a negative one. For instance, for the event types 'created' and 'assigned' the event types 'createRejected' and 'assignRejected' can be conceived. Learning the classification rules that predict whether, given the state of a process instance, a particular state transition can occur, then boils down to learning the classification rule that predicts when either a positive or a negative event occurs. In this way, the following process mining tasks can be identified within the framework:

- *create*: classifying whether at a given moment, an activity can be created or not (*createRejected*). The obtained classification model might reveal information about business concerns such as **activity cardinality**, and **activity exclusion**.
- *schedule*: learning to discriminate whether a proposed due date or performance date will be accepted or rejected. The obtained classification model might obtain information about scheduling policies, and due dates.
- *assign*: learning the transition constraints that determine the conditions under which an agent can be assigned to an activity (*assigned*) or not (*assignRejected*). The obtained classification model might reveal information about **activity authorization** and **activity allocation** policies.
- *start*: learning the conditions that discriminate between a started or a *startRejected* event. This learning task is know in the literature as **process discovery**.
- *addFact*, *removeFact*, *updateFact*: learning the conditions that determine whether a *factAdded* or *addFactRejected* event takes place. This learning task can reveal information about static and dynamic data **integrity constraints**.
- *complete*: classifying whether a *completed* or *completeRejected* event takes place. From the learned classification model, one can obtain information about the intended effect of an activity in terms of an **activity postcondition** or an **activity inclusion** constraint.
- *skip*: classifying the conditions that determine whether an activity can be skipped.

- *abort*: classifying the conditions that determine whether an activity can be aborted.
- *redo*: classifying the conditions that determine whether an activity can be redone.

Not every process mining tasks is as useful or as widely applicable as others. Some learning tasks have not yet been addressed in the literature.

4.4.2 Declarative Mining Techniques

In order to be useful in practice, process mining techniques must be capable of learning process models that not only accurately describe the patterns in event logs, but that are also comprehensible to human end-users, and that take into account the specific domain knowledge and requirements of the end user. In his PhD, Martens (2008) brings forward three general requirements that make an induced classification model acceptable: accuracy, comprehensibility, and justifiability.

- **Accuracy** refers to the extent to which the induced model fits the behavior in the event log and can be generalized towards unseen behavior.
- **Comprehensibility** refers to the extent to which an induced model is comprehensible to end-users.
- **Justifiability** refers to the extent to which a model is aligned with the existing domain knowledge of the end-user.

Many process mining techniques primarily focus on the first criterion: accuracy. This is is a necessary, but insufficient requirement for process mining. An induced model that accurately describes the observed behavior, but that is completely incomprehensible to end users, or inconsistent with the knowledge of end users is likely to be less useful than a slightly less accurate model that is comprehensible and justifiable.

Declarative process mining techniques also target the comprehensibility and justifiability of the induced models. A process mining algorithm is said to be **declarative**, when it possesses the following learning features.

- **A declarative language bias.** The language bias of a learner is a specification of possible language constructs that a learner can use to construct hypotheses that explain the observed behavior in the event log. The language bias determines the nature and number of the hypotheses that can possibly be induced. For instance, the language bias of the α algorithm covers the class of Structured Workflow Nets (SWF-nets) (van der Aalst et al., 2004). A learner has a *declarative language bias*, when it uses a declarative process modeling language to represent the discovered patterns. In some cases, learners with a declarative language bias potentially discover more comprehensible process models. In particular, declarative languages can be expected to have a more condense representation when describing loosely-structured processes, whereas procedural languages are more concise when describing highly-structured processes. Pesic et al. (2007a), for instance, make this contrast when comparing the declarative ConDec (Pesic and van der Aalst, 2006) and the procedural YAWL (van der Aalst and ter Hofstede, 2005) process languages.

- **A configurable language bias.** An end user is likely to have an idea of the language constructs that are likely to be most successful in describing the behavior that is contained within an event log. Therefore, it potentially is very useful that an end-user can manipulate the langue bias of a learning algorithm. By configuring the language bias of a learner, a learner might produce more accurate or more condense patterns. An example of this, can be found with the decision point analysis technique of Rozinat and van der Aalst (2006), who discuss the use of uni-relational classification for the purpose of decision mining. In decision mining so-called decision points are identified in process logs, and the classification problem consists of determining which case data properties lead to certain paths being taken in the process. The algorithm has a configurable language bias in that allows specifying whether the time point at which a case data point is being entered should occur prior to the decision point or not. Sometimes a relevant case data attribute has not yet been entered, whereas it already affects the routing choices in the event log.

- **A configurable inductive bias.** When learning patterns from a sample data set, learners inevitably must make a so-called "inductive leap". This inductive leap expresses the uncertainty about the learned patterns being reasonable inferences that are generalizable towards the entire population. For instance, when summarizing a scatter plot of two continuous variables in terms of a linear function, we implicitly assume that the underlying population portrays a linear relationship between these to variables. The inductive bias of a learner is what is required for making an inductive leap; it is the additional set of assumptions that need to be added to the inductive system such that the predictions of the learner would follow deductively instead of inductively (Mitchell, 1997). Without an inductive bias, a learner can only memorize the observed data instances. For some algorithms, the inductive bias can be elegantly expressed in logic, whereas for other, black-box learning techniques, it is not possible to bring its inductive bias under words. The language bias is only part of the inductive bias of a learner: it is the assumption that the population can be described in terms of the language bias; that the correct hypothesis that is part of the hypothesis space. Another aspect of the inductive bias, is how a learner discriminates between good and bad hypotheses. In process mining, it can be interesting to have a configurable inductive bias beyond a configurable language bias. By configuring the inductive bias of a learner, one would in theory be capable of manipulating the learner's ability to generalize from the observed behavior. For example, many process discovery algorithms make a completeness assumption about the observed event sequences to strike an appropriate balance between precision and recall. By making this completeness assumption configurable, a user could manipulate the amount of unobserved behavior he or she is willing to accept in the model to be learned.

- **Account for prior knowledge.** In general, the problem of consolidating the knowledge extracted from the data with the knowledge representing the experience of domain experts, is called the knowledge fusion problem (Dybowski et al., 2003). Providing a proper solution for this problem is a key success factor for any learner. Prior knowledge constrains the considered hypothesis space to a subset that is consistent with the prior knowledge. An example of this might be the prior knowledge that, all other things being equal, a manager has at least the same access rights as his subordinates. A learner that is capable of taking this prior knowledge into account would refrain from inferring access control hypotheses that under the same circumstances would grant more access rights to a subordinate than to a manager.

With this characterization of declarative process mining techniques, no dichotomy is implied: some learners will have more declarative features than others, but no crisp distinction can be made.

Process mining algorithms are not solely considered to be declarative because of a language bias that allows the induction of declarative process models. Declarative process models make the business concerns that govern business processes explicit. However, it would in general be very difficult to directly identify business concerns from an event log only using machine learning techniques. For instance, process discovery techniques might reveal a dependency relationship between two activity types, but this does not necessarily capture the underlying business concern. For this reason, models obtained from process mining are most often subjected to the interpretation of a human-expert, rather than automatically being taken into production. Human-experts are capable of identifying the underlying business concerns both from procedural and declarative process models. After interpretation, the experts can close the loop in the BPM life cycle, by documenting and modeling the discovered knowledge, preferably using declarative languages. What also matters is that the learning algorithms are configurable in a declarative manner. The user of a process mining algorithm, very often has specific knowledge and requirements for a particular learning problem. Mining techniques that are capable of taking this into account can be expected to be more useful. Therefore process mining algorithms must also provide means of including a-priori knowledge from the end-user, his or her perception about the language constructs that are most likely to be able to describe the patterns in the event log (the language bias), and his or her perception about how a learner can best generalize beyond the observed patterns (the inductive bias).

4.5 Conclusion

Albeit a unifying approach, the EM-BrA^{2}CE Framework, introduced in the previous chapter, remains a framework and thus purposefully leaves many gaps to fill in. In this chapter, we have filled in a small number of these gaps, by indicating how the framework allows for the application of declarative modeling, simulation, and mining techniques.

First, we have indicated how many business concerns related to business processes can be represented as business rule types. To this end, we have identified sixteen business rule types that can be useful in verbalizing various business concerns. For these business rule types we have indicated related work, and briefly discussed how they can be declaratively enforced within the execution semantics of the framework. Furthermore, we have introduced a new language for reasoning about obligations, permissions, and due dates, called PENELOPE, and indicated how a set of rules specified in this language can be visualized and verified. To conclude the modeling aspect of this text, we have briefly discussed two simulation models that serve as a proof-of-concept of the execution semantics.

In this chapter, we have also made the transition from process modeling towards process mining. We have identified twelve data mining tasks that can be formulated as classification learning problems on event logs with negative events. Furthermore, we have provided a characterization of declarative process mining techniques. In particular, a process mining technique is declarative when it allows the user to have some control over learning features such as prior knowledge, language bias, and inductive bias. To illustrate declarative process mining, we have indicated how access rules can be discovered from event logs using an existing, multi-relational classification learner. Learning transition constraints such as access rules is relatively easy when an event log by itself contains negative events. There are, however, learning problems for which event logs generally contain no negative events. This is, for instance, often the case for process discovery. In the next chapter, we will focuss on this learning problem.

CHAPTER 5

Declarative Process Discovery with Artificial Negative Events

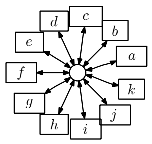
Declarative process discovery techniques focuss on comprehensibility and justifiability of the discovered process models, in addition to their accuracy. Inductive Logic Programming (ILP) is a machine learning technique that is particularly suited for raising comprehensibility and justifiability. Ferreira and Ferreira (2006) show that process discovery can be formulated as an ILP classification learning problem on event logs with both positive and negative events. In reality, event logs often do not contain negative examples. Without negative examples, it is a challenge to strike the right balance between generality and specificity, while dealing with problems such as expressiveness, noise, incomplete event logs, or the inclusion of prior knowledge. In this chapter, we present a declarative technique that deals with these challenges by representing process discovery as a multi-relational classification problem on event logs supplemented with Artificially Generated Negative Events (AGNEs). The AGNEs technique is implemented as a mining plugin in the ProM Framework (Process Mining Group, TU/Eindhoven, 2008).

5.1 Process Discovery

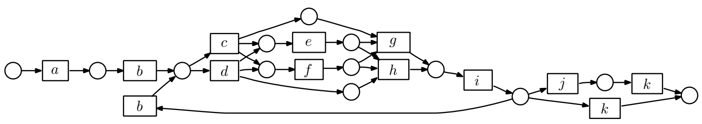
Within the analysis of event logs, *process discovery* is an important learning task. Process discovery aims at correctly summarizing an event log and describing how processes have *actually* taken place. Processes occur in a more or less structured fashion, containing structures such as or-joins, or-splits, and-joins, and-splits, and loops. The learning task can be formulated as follows: given an event log that contains the events about a finite number of process instances, find a model that correctly summarizes the behavior in the event log, striking the right balance between generality (allowing enough behavior) and specificity (not allowing too much behavior). For the purpose of process discovery, processes are often represented as workflow nets, a special subclass of Petri nets. Figure 5.1 illustrates the learning problem for a Driver's License application process. Given the event log in Figure 5.1(a), different process models can be conceived that portray similar behavior as the event log. The Petri net in Figure 5.1(b) is capable of parsing every sequence in the event log. However, it can be considered to be overly general as it allows any activity to occur in any order. In contrast, the Petri net in Figure 5.1(d) is overly specific, as it provides a mere enumeration of the different sequences in the event log. The Petri net in Figure 5.1(c) is likely to be the more suitable process model. It is well-structured, and strikes a reasonable balance between specificity and generality, allowing for instance an unseen sequence *abcefgik*, but disallowing random behavior.

σ_1	abcefgijk
σ_2	abdfehijk
σ_3	abdefhijbdfehijk
σ_4	abcfegik
...	...

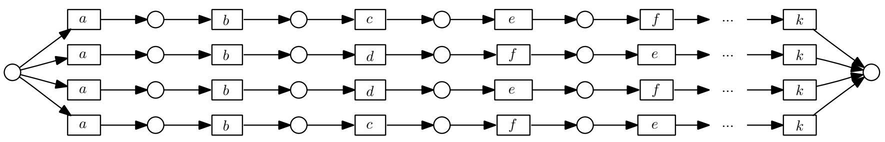
(a) event log



(b) an over-general model



(c) a specific and general model



(d) an over-specific model

Figure 5.1: DriversLicensel – discovery of a driver's license application process – (the transitions correspond to activity types that have the following meaning: *a* start, *b* apply for license, *c* attend classes cars, *d* attend classes motor bikes, *e* obtain insurance, *f* theoretical exam, *g* practical exam cars, *h* practical exam motor bikes, *i* get result, *j* receive license, and *k* end.)

An important aspect of process discovery is the visualization of the discovered process model in graphs. In the spirit of Chapter 2, such graph-based process models can be seen as *procedural process models*, because they consist of a "precomputation of activity dependencies" (Schmidt and Simone, 1996). In contrast, *declarative process models* reflect the underlying business concerns that govern business processes and that lead to activity dependencies. In this chapter, we nonetheless focus on the discovery of graph-based, procedural process models from event logs. We believe that, in general, it would be very difficult to identify the underlying business concerns from an event log using machine learning techniques. Instead, the approach of summarizing an event log in a graph-based, procedural model, and having a human expert determine the underlying business concerns using a declarative modeling technique is a valid and productive one. Graph-based models are also useful for performance analysis techniques such as simulation, and critical path analysis.

An inherent difficulty of process discovery is that it is often limited to a setting of unsupervised learning. Event logs rarely contain negative information about state transitions that were prevented from taking place. Such negative information can be useful to discover the discriminating properties of the underlying process. In the absence of negative information, it is necessary to provide a learner with a particular *inductive bias*, to strike the right balance between generality and specificity, while at the same time dealing with challenging problems such as expressiveness, noise, incomplete event logs, and the inclusion of prior knowledge (van der Aalst and Weijters, 2004):

- **expressiveness**: expressiveness relates to the ability to summarize an event log using a rich palette of **structures** such as sequences, splits, joins, and loops. Additionally, learners must be capable of detecting **history-dependent behavior**. Human-centric processes portray behavior that is dependent on the non-immediate history. An example is the occurrence of history-based joins in the control flow of a process or so-called non-local, non-free choice constructs (Wen et al., 2007). Another challenge is the detection of **duplicate activities**. These are activities that are identically labeled but that are used in different execution contexts within the process. A further challenge is the **inclusion of case data**. The routing choices that are made in processes can also be dependent on the value of its data properties. The inclusion of case data in discovered process models is therefore potentially useful.

- **noise**: Human-centric processes are prone to exceptions and logging errors. This causes additional low-frequent behavior to be present in the event log that is unwanted in the process model to be learned. Consequently, process discovery algorithms face the challenge of not overfitting this noise.

- **incomplete logs**: Incomplete event logs do not contain the complete set of sequences that occur according to the underlying, real-life process. Process discovery algorithms must be capable of generalizing beyond the observed behavior in the incomplete event logs. This is particularly the case for processes that portray a large amount of concurrent and recurrent behavior. For these process models, the completeness assumption that all behavior of interest is present in the event log quickly becomes invalid.

- **prior knowledge**: In the context of process discovery, prior knowledge might refer to knowledge about concurrency (parallelism), locality or exclusivity of activities. When a learner produces a process model that is not in line with the prior knowledge of a domain expert, the expert might refuse using the discovered process model. For instance, a domain expert might refuse a process model in which a pair of activities cannot take place concurrently, whereas in reality such parallelism is actually allowed. For this reason, process discovery should be capable of taking into account prior knowledge.

In this chapter, all above-mentioned challenges are addressed by representing process mining as first-order classification learning on event logs supplemented with artificially generated negative events (AGNEs). The AGNEs technique is capable of constructing Petri net models from event logs and has been implemented as a mining plugin in the ProM framework. In the next chapter, the performance of our technique is evaluated. Benchmark experiments with 34 artificial event logs and comparison to four state-of-the-art process discovery algorithms indicate that the technique is expressive, robust to noise, and capable of dealing with incomplete event logs.

Algorithm 1 outlines four major steps in the AGNEs process discovery procedure. These four steps are addressed in four subsequent sections of this chapter. AGNEs is called a **declarative process discovery** technique.

- **Declarative specification of prior knowledge.** AGNEs allows to specify prior knowledge about parallelism and locality as a logic program, and uses this prior knowledge to actively constrain its hypothesis space. The ability to specify prior knowledge as a number of clauses is very expressive. For instance, the constraint $\forall a \in A : \mathit{PriorSerial}(\text{`}\mathit{accept\ application}', a)$ indicates that the activity *accept application* cannot occur in parallel with any other activity.

- **Declarative control over the inductive bias.** In general, learning from positive data only requires an inductive bias of additional assumptions about the learning problem. The *inductive bias* of a learner is the additional set of assumptions that need to be added such that the predictions of the learner would follow deductively instead of inductively (Mitchell, 1997). Many process discovery algorithms implicitly have a completeness assumption as inductive bias. AGNEs makes this completeness assumption explicit by generating artificial negative events. Because the technique allows the end-user to configure the negative event generation procedure taking into account window size, and parallel variants, it provides the user with declarative control over the inductive bias. By manipulating the inductive bias, the end-user can set the balance between generality and specificity.

- **Declarative control over the language bias.** The *language bias* of the learner is the set of language constructs that a learner can use to produce a hypothesis that fits the learning problem. The language bias of a learner determines the hypothesis space, the set of all allowable hypotheses. Because

AGNEs makes use of an Inductive Logic Programming (ILP) classification learner, it allows the user to have a declarative control over the language constructs that can be used. The AGNEs technique aims at producing a workflow net representation of the learned hypothesis space. Therefore, the language bias is tailored toward a specific event operator. However, other language constructs, involving case data conditions or aggregation functions (such has the maximum occurrence of an event), can also be included and are visualized as guard conditions on Petri net transitions.

Algorithm 1 Process discovery by AGNEs: overview

```
1: step 1: derive parallelism and locality from frequent temporal constraints
2: step 2: generate artificial negative events
3: step 3: learn the preconditions of each activity type
4: step 4: transform the preconditions into a Petri net
```

The remainder of this chapter is structured as follows. A related work section first provides an overview of the work in the area of process discovery and indicates the contributions made to the state of the art. Section 5.3 introduces some preliminaries and notations about Petri nets, event logs and inductive logic programming. Section 5.4 provides a detailed description of the first step in the AGNEs process discovery technique: the derivation of parallelism and locality information from frequent temporal constraints. Section 5.5 explains the necessity for generating artificial negative events and provides a detailed description of the used algorithms. Section 5.6 discusses the language bias that is to be supplied to a multi-relational classification learner in order to induce activity preconditions that can be transformed into a Petri net. Section 5.7 provides a detailed description of how a set of induced preconditions can be transformed into a Petri net.

5.2 Related Work

Although grammar learning generally only considers sequential data and not concurrent behavior, process discovery can be seen as an application of the machine learning of grammars from positive data, of which Angluin and Smith (1983) provide an overview. Gold (1967) has shown that important classes of recursively enumerable languages cannot be *identified in the limit* from a finite number of positive examples only. Instead, both positive and negative examples are required for grammar learning to distinguish the right hypothesis among an infinite number of grammars that fit the positive examples. Whereas Gold's negative learnability result applies to the learning of grammars with perfect accuracy, process discovery is more concerned with the ability to discover process models that have *only* a good recall and specificity. Learning grammars from only positive examples requires a tradeoff between overly general and overly specific hypotheses. Muggleton (1996) shows that in a Bayesian framework, logic programs are learnable with arbitrarily low error from positive examples only. Bayes' theorem allows to formulate this tradeoff as a tradeoff between size and generality of the hypotheses and learning can be considered to maximization the posterior probability over all hypotheses. In this chapter, a new approach for making the tradeoff between generality and specificity is proposed, by inducing artificial negative events using a (highly configurable) assumption about the completeness of the behavior displayed by the positive examples in the event log.

The term process discovery was coined by Cook and Wolf (1998), who apply it in the field of software engineering. Their Markov algorithm can only discover sequential patterns as Markov chains cannot elegantly represent concurrent behavior. The idea of applying process discovery in the context of workflow management systems stems from Agrawal et al. (1998) and Lyytinen et al. (1998). The value of process discovery for the general purpose of process mining (van der Aalst et al., 2007a) is well illustrated by the plugins within the ProM framework. In analogy with the WEKA toolset for data mining (Witten and Frank, 2000), the ProM Framework consists of a large number of plugins for the analysis of event logs (Process Mining Group, TU/Eindhoven, 2008). The *Conformance Checker* plugin (Rozinat and van der Aalst, 2008), for instance, allows identifying the discrepancies between an idealized process model and an event log. Moreover, with a model that accurately describes the event log, it becomes possible to use the time-information in an event log for the purpose of performance analysis, using, for instance, the *Performance Analysis with Petri nets* plugin.

Table 5.1 provides a chronological overview of process discovery algorithms that have been applied to the context of workflow management systems. The α algorithm can be considered to be a theoretical learner for which van der Aalst et al. (2004) prove that it can learn an important class of workflow nets, called structured workflow nets, from complete event logs. The α algorithm assumes event logs to be complete with respect to all allowable binary sequences and assumes that the event log does not contain any noise. Therefore, the α algorithm is sensitive to noise and incompleteness of event logs. Moreover, the original α algorithm was incapable of discovering short loops or non-local, non-free choice constructs. Alves de Medeiros et al. (2004) have extended the α algorithm to mine short loops and called it α^+. Wen et al. (2007) made an extension for detecting implicit dependencies, for detecting non-local, non-free choice constructs. None of the algorithms can detect duplicate activities. The main reason why the α algorithms are sensitive to noise, is that they does not take into account the frequency of binary sequences that occur in event logs. Weijters and van der Aalst (2003) and Weijters et al. (2006) have developed a robust, heuristic-based method for process discovery, called heuristics miner, that is known to be noise resilient. Heuristics miner can discover short loops, but it is not capable of detecting non-local, non-free choice as it does not consider non-local dependencies within an event log. Moreover, heuristics miner cannot detect duplicate activities.

van Dongen and van der Aalst (2005b) present a multi-phase approach to process mining that starts from the individual process sequences, constructs so-called instance graphs for each sequence that account for parallelism, and then aggre-

Table 5.1: Chronological overview of process discovery algorithms

Algorithm (Ref.)	Summary
global partial orders (Mannila and Meek, 2000)	► Learns a two-component mixture model of a dominant series-parallel partial order and a trivial partial order by searching for the dominant partial order that yields the highest probability for generating the observed sequence database.
$\boldsymbol{\alpha}, \boldsymbol{\alpha}^+$ (van der Aalst et al., 2004) (Alves de Medeiros et al., 2004)	► Derives a Petri net from local, binary ordering relations detected within an event log.
little thumb, **heuristics miner** (Weijters and van der Aalst, 2003) (Weijters et al., 2006)	► Extends the α algorithm by taking into account the frequency of the follows relationship, to calculate dependency/frequency tables from the event log and uses heuristics to convert this information into a heuristics net.
splitpar – InWoLvE (Herbst and Karagiannis, 2004)	► Derives a so-called stochastic activity graph and converts it into a structured process model in the Adonis Modeling language.
multi-phase miner (van Dongen and van der Aalst, 2005b)	► Constructs a process model for every sequence in the log and aggregates the model into an event-driven process chain.
$\boldsymbol{\alpha}^{++}$ (Wen et al., 2007)	► Extends the α algorithm to discover non-local, non-free choice constructs.
– (Silva et al., 2005)	► A probabilistic approach to process discovery.
frecpo (Pei et al., 2006)	► A scalable technique for discovering the complete set of frequent, closed partial orders from sequential data.
FSM/Petrify miner (van der Aalst et al., 2006)	► Derives a highly configurable finite state machine from the event log and folds the finite state machine into regions using the theory of regions.
– (Ferreira and Ferreira, 2006)	► Learns the case data preconditions and effects of activities with ILP classification techniques and user-supplied negative events.
genetic miner (Alves de Medeiros et al., 2007)	► A genetic algorithm that selects the more complete and precise heuristics nets over generations of nets.
DecMiner (Lamma et al., 2007)	► A classification technique that learns the preconditions of activities with the ICL ILP learner from event logs with user-supplied negative sequences.
fuzzy miner (Günther and van der Aalst, 2007)	► An adaptive simplification and visualization technique based on significance and correlation measures to visualize unstructured processes.

gates these instance graphs according to previously detected binary relationships between activity types. Interestingly, the aggregation ensures that every discovered process model has a perfect recall, but generally scores less on specificity. Herbst and Karagiannis (2004) describe the working of the splitpar algorithm that is part of the InWoLvE framework for process analysis. This algorithm derives a so-called stochastic activity graph and converts it into a structured process model. The splitpar algorithm is capable of detecting duplicate activities, but it is not able to discover non-local dependencies.

Alves de Medeiros et al. (2007) describe a genetic algorithm for process discovery. The fitness function of this genetic algorithm incorporates both a recall and a precision measure that drives the genetic algorithm towards suitable models. The genetic miner is capable of detecting non-local patterns in the event log and is described to be fairly robust to noise. In her PhD, Alves de Medeiros (2006) describes an extension to this algorithm for the discovery of duplicate tasks.

van der Aalst et al. (2006) present a two-phase approach to process discovery that allows to configure when states or state transitions are considered to be similar. The ability to manipulate similarity is a good approach to deal with incomplete event logs. In particular, several criteria can be considered for defining similarity of behavior and states: the inclusion of future or past events, the maximum horizon, the activities that determine state, whether ordering matters, the activities that visibly can bring about state changes, etcetera. Using these criteria, a configurable finite state machine can be constructed from the event log. In a second phase, the finite state machine is folded into regions using the existing theory of regions (Cortadella et al., 1998). For the moment, the second phase of the algorithm still poses difficulties with respect to constructing suitable process models. The approach presented in this chapter considers window size (maximum horizon) and parallel variants as similarity criteria when generating artificial negative events.

Günther and van der Aalst (2007) present an adaptive simplification and visualization technique based on significance and correlation measures to visualize the behavior in event logs at various levels of abstraction. The contribution of this approach is that it can also be applied to less structured, or unstructured processes of which the event logs cannot easily be summarized in concise, structured process models.

Several authors have used classification techniques for the purpose of process discovery. Maruster et al. (2006), for instance, were among the first to investigate the use of rule-induction to predict dependency relationships between activities from a corpus of reference logs that portray various levels of noise and imbalance. To this end, the authors use a propositional rule induction technique, the uni-relational classification learner RIPPER (Cohen, 1995), on a table of direct metrics for each process task in relation to each other process task, which is generated in a pre-processing step.

Ferreira and Ferreira (2006) apply a combination of ILP learning and partial-order planning techniques to process mining. Rather than generating artificial negative events, negative examples are collected from the users who indicate whether a proposed execution plan is feasible or not. By iteratively combining planning and learning, a process model is discovered that is represented in terms of the case data preconditions and effects of its activities. In addition to this new process mining technique, the contribution of this work is in the truly integrated BPM life cycle of process generation, execution, re-planning and learning. Lamma et al. (2007) also describe the use of ILP to process mining. The authors assume the presence of negative sequences to guide the search algorithm. Unlike the approach of Ferreira and Ferreira, who use partial-order planning to present the user with an execution plan to accept or reject (a negative example), this approach does not provide an immediate answer to the origin of the negative events. Contrary to our approach, the latter two approaches are not concerned with the construction of a graphical, control-flow based process model and do not consider the generation of artificial negative events.

In the literature, there are many formalisms to represent and learn the probability distributions of stochastic, *generative* grammars over sequences of observed characters and unobserved state variables. Historically, techniques like Markov models, hidden Markov models, factorial hidden Markov models, and dynamic Bayesian networks have been first applied to speech recognition, and bio-informatics (Durbin et al., 1998). Each representation has its own particular modeling features that makes it more or less suited for representing the human-centric behavior of business processes. Factorial hidden Markov models, for instance, have a distributed state representation that allows for the modeling of concurrent behavior (Ghahramani and Jordan, 1997). Other authors describe learning mixture models to identify meaningful clusters of (hidden) Markov models (Cadez et al., 2003; Smyth, 1997). Whereas stochastic models provide useful information, their probabilistic nature tends to compromise the comprehensibility of discovered process models. Business processes have well-defined start, end, split, and synchronization nodes. The network structure of stochastic models does visualize not this. For instance, although hidden states could be useful in representing duplicate activities – the same activity label is logged in different contexts – a hidden Markov model is unlikely to be capable of comprehensively representing its different usage contexts.

In contrast, Mannila and Meek (2000) describe a technique to learn two-component mixture models of global partial orders that provide an understandable, global view of the sequential data. The authors assume the presence of one dominant, global partial order and consider a generic partial order with random behavior to deal with low-frequent variations (noise) from the former model. Silva et al. (2005) describe a probabilistic model and algorithm for process discovery that discovers so-called and/or graphs in polynomial time. These and/or graphs are comprehensible, directed acyclic graphs that have the advantage over global partial order representations that they can differentiate between parallel and serial split and join points. Pei et al. (2006) describe a scalable technique for discovering the complete set of frequent, closed partial orders from sequential data. The three aforementioned techniques assume each item to occur only once within a sequence, and do not consider recurrent behavior (cycles), nor duplicate activities.

5.3 Preliminaries

This section introduces the most important concepts and notations that are used in the remainder of this chapter.

5.3.1 Inductive Logic Programming

Inductive Logic Programming (ILP) (Džeroski, 2003; Džeroski and Lavrač, 1994, 2001; Muggleton, 1990) is a research domain in machine learning that involves learners that use logic programming to represent data, background knowledge,

ID	Gender	Age	Income	Important Costumer
C1	M	30	215000	Yes
C2	F	19	140000	Yes
C3	M	56	50000	No
C4	F	48	28000	No

(a) customer

Partner 1	Partner 2
C1	C2
C3	C4

(b) partner

Figure 5.2: Multi-relational datasets

the hypothesis space, and the learned hypothesis.

Multi-relational Representation

ILP learners are also called multi-relational data mining (MRDM) learners. Multi-relational data mining extends classical, uni-relational data mining in the sense it can not only learn patterns that occur within single tuples (within rows), but can also find patterns that may range over different tuples of different relations (between multiple rows of a single or multiple tables). For the purpose of discovering non-local, non-free choice constructs, this multi-relational property is much desired, as it allows discovering patterns in the event log that relate the occurrence of an event to the occurrence of any other event in the event log. This aspect will be addressed in Section 5.6.

To understand the idea, consider the following example. The example database of Figure 5.2 consists of two tables, whereby the second table indicates which persons from the first table are married with each other (Džeroski, 2003). From this database, one wants to establish a decision tree so that the important customers can be identified swiftly. Propositional learners create classification rules of the following form: IF (income $>$ 100000) THEN important customer $=$ YES. Observe that only the information from the first table was used for the creation of this rule. Relational algorithms on the other hand are able to use the relationships that exist among multiple tuples. An example of such a rule is: IF (x is married with a person with income $>$ 100000) THEN important customer (x) $=$ YES. To allow propositional learners to exploit this multi-relational information the multi-relational data mining problem has to be converted a uni-relational problem, as shown in Figure 5.3. We can see that we need three extra columns to describe the properties of the partner. This technique has been applied to multi-relational data mining (Lavrac et al., 1991) and allows to keep on using propositional learners. Nonetheless, it is less elegant, as it transfers the problem of non-local search to the input space. More importantly, it also exponentially increases the dimensions of the input space. High dimensional input spaces are typically hard to handle by classical data mining techniques, a problem known as 'curse of dimensionality' (Tan et al., 2005).

ID	Gender	Age	Income	Gender Partner	Age Partner	Income Partner	Important Costumer
C1	M	30	215000	F	19	140000	Yes
C2	F	19	140000	M	30	215000	Yes
C3	M	56	50000	F	48	28000	No
C4	F	48	28000	M	56	50000	No

Figure 5.3: Transformation to uni-relational problem

Logic Programming

In this section, we briefly introduce the basic concepts of logic programming that are useful to understanding the finer details of ILP classification learning with TILDE. The overview is based on the comprehensive text books of Brachman and Levesque (2004); Džeroski and Lavrač (2001); Hogger (1990).

The syntax of first-order logic or predicate logic consists of terms, atomic formulae, and well-formed formulae. Variables, functions, and constants are *terms.* A variable such as x is a term. A function symbol followed by a bracketed n-tuple of terms is a term. For example, $F(A, G(B), x)$ is a term when F, A, and G are function symbols and x is a variable - strings starting with upper case characters denote predicate and function symbols, whereas strings starting with lower case characters denote variables. A constant is a function of arity zero. A predicate symbol that is immediately followed by a bracketed n-tuple of terms is called an *atomic formula.* A *well-formed formula* is either an atomic formula or a composite of atomic formulae of the following forms: F, $\neg F$ (negation), $F \vee G$ (logical disjunction), $F \wedge G$ (logical conjunction), $F \Leftarrow G$ (logical implication), $F \Leftrightarrow G$ (logical equivalence), $\forall x : F$ (universal quantification), and $\exists x : F$ (existential quantification), where F and G are well-formed formula and x is a variable.

A theory in first-order logic is normally expressed in clausal form. A *clause* consists of a logical disjunction of literals in which each variable is universally quantified. A *positive literal* is an atom, a *negative literal* is a negated atom. In other words, a clause is of the form $\forall x_1, x_2, ...x_s(A_1 \vee A_2 \vee ... \vee A_h \vee \neg B_1 \vee \neg B_2 ... \vee \neg B_b)$, where each A_i is a positive literal and each $\neg B_i$ is a negative literal and each X_j are all the variables occurring in $(A_1 \vee A_2 \vee ... \vee A_h \vee \neg B_1 \vee \neg B_2 ... \vee \neg B_b)$. A clause can also be represented using logical implication, giving clauses an intuitive *if-then*-rule reading. The previous clause is written as $A_1, A_2, ..., A_h \Leftarrow B_1, B_2 ..., B_b$. The positive literals $A_1, A_2, ..., A_h$ appear in the *head* of the clause. The negative literals $B_1, B_2 ..., B_b$ appear in the *body* of the clause. Commas in the head denote logical disjunction, while commas in the body of the clause denote logical conjunction.

The logic programs that are used by ILP learners to represent examples $e \in E$, background knowledge B, and hypotheses $H \in \mathcal{S}$ are most often expressed in Prolog. Prolog is a logic programming language that consists of definite logic programs extended with some procedural reasoning features. Horn logic consists of a conjunction of Horn clauses. A *Horn clause* is a clause that consists of at most one positive literal, and thus at most has one literal in the rule head; it is also a *definite clause* if it contains exactly one positive literal. A set of definite clauses is called a *definite logic program.*

A *Herbrand interpretation* I of a first-order clausal theory is a set of atomic logic formulae – this is also called the model – to which the theory assigns the value true. More formally, a Herbrand interpretation I is a model for a clause c if and only if for all substitutions θ such that $c\theta$ is ground (i.e. $c\theta$ no longer contains non-substituted variables), $body(c)\theta \subset I$ implies $head(c)\theta \cap I \neq \emptyset$. A Herbrand interpretation I is a model for a clausal theory T, if and only if it is a model for all clauses in T.

Each definite logic program (Prolog program) has a single, unique Herbrand interpretation, which is called the unique, *least Herbrand model* which contains all non-negated atomic logical formula (ground facts) that are true in the logic program. Prolog programs are fit with a so-called closed-world assumption (CWA). Under this assumption, atomic logical formula (ground facts) that cannot be derived from the program are assumed to be false. This assumption is valid when the logic program has a complete knowledge about the "world" it models. Because of this closed-world assumption, Prolog allows for a special kind of negation of atoms in the body of the clause. This negation is called *negation-as-failure* and is written as $\sim B$, where B is an atom.

Prolog programs do not contain functions of non-zero arity. This means that only variables and constants appear as terms in atomic formulae.

Learning from Interpretations

Whereas logic programming (LP) is concerned with deductive reasoning with logic programs so as to derive new ground facts from an initial set of axioms and inference rules, inductive logic programming (ILP) is concerned with inductive reasoning. In particular, ILP focusses on making generalizations from observed instances in the presence of background knowledge, finding patterns that also apply to unobserved instances.

Within ILP, *concept learning* is an important learning task. An ILP classification learner will search for a hypothesis H in a hypothesis space $\mathcal{S}$ that best discriminates between the positive P and negative examples N ($E = P \cup N$) in combination with some given background knowledge B. A particularly salient feature of such learners is that they have a highly configurable language bias. The language bias $\mathcal{L}$ specifies the hypothesis space $\mathcal{S}$ of logic programs H that can be considered. In addition, users of ILP learners can specify background knowledge B as a logic program. Such background knowledge is a more parsimonious encoding of knowledge that is true about every example, than is the case for the attribute-value encoding of propositional learners (Blockeel, 1998).

In his PhD Blockeel (1998) discusses the advantages of different ILP learning settings. Most ILP learners consider a setting that is called *concept learning by entailment.* The learning task is formalized as follows (Blockeel, 1998):
given:

- a set of positive examples P and negative examples N
- a logic program B that represents the background knowledge
- a language bias $\mathcal{L}$ that specifies a hypothesis space $\mathcal{S}$ of logic programs.

find: a hypothesis $H \in \mathcal{S}$ (a logic program) such that,

- $\forall e \in P : H \wedge B \models e$
- $\forall e \in N : H \wedge B \not\models e$.

In this setting, it is possible to learn a full logic program that contains recursive clauses, relating the classification of one example to the classification of other examples. The latter requires each example to be identifiable with a specific example identifier. Whereas few concept learning problems require hypotheses that contain recursive clauses, this broad application potential unfortunately results in an increased computational complexity of many of these ILP learning algorithms (Blockeel, 1998).

In the problem setting of *concept learning from interpretations*, the classification of each example e is considered to be independent from the classification of each other example. Therefore, an example $e \in E$ constitutes an independent logic program. The intent is to learn a hypothesis H, a set of clauses, such that each positive example makes each clause in H true, whereas none of the negative examples do so. More formally, the goal is to find a hypothesis $H \in \mathcal{S}$ such that (Blockeel, 1998):

- $\forall e \in P : H$ is true in $M(B \wedge e)$
- $\forall e \in N : H$ is false in $M(B \wedge e)$,

where $M(B \wedge e)$ is least Herbrand interpretation of $B \wedge e$. Because each example is considered to be independent, it is no longer possible to learn recursive relationships. Nonetheless, the concept learning from interpretations setting maintains the capability of finding patterns that apply among multiple relations within a logic program.

In addition to their multi-relational capabilities, the power of ILP concept learners lies with the configurability of their language bias $\mathcal{L}$ and background knowledge B. The effectiveness by which an ILP learner can be applied to a learning task depends on the choices that are made in representing the examples E, the background knowledge B and the language bias $\mathcal{L}$.

TILDE

In this text, we make use of TILDE (Blockeel and De Raedt, 1998), a first-order decision tree learner available in the ACE-ilProlog data mining system (Blockeel et al., 2002). TILDE is positioned within the above-introduced learning from

interpretations setting. The learning task is classification learning rather than concept learning. Concept learning can be considered to be a special case of classification learning, in which the target classes are the binary classes true and false. Blockeel (1998) formalizes the learning task of TILDE as follows:
given:

- a set of classes C
- a set of classified examples E, each example $(e, c) \in E$ is an independent logic program for which the predicate $\mathit{Class}(c)$ denotes that e is classified into class c.
- a logic program B that represents the background knowledge
- a language bias $\mathcal{L}$ that specifies a hypothesis space $\mathcal{S}$ of logic programs.

find: a hypothesis $H \in \mathcal{S}$ (a logic program) such that for all labeled examples $(e, c) \in E$,

- $\forall e \in E : H \wedge e \wedge B \models \mathit{Class}(c)$
- $\forall e \in E, \forall c' \in C \backslash \{c\} : H \wedge e \wedge B \not\models \mathit{Class}(c')$.

TILDE is a first-order generalization of the well-known C4.5 algorithm for decision tree induction (Quinlan, 1993). Like C4.5, TILDE (Blockeel and De Raedt, 1998; Blockeel et al., 2002) obtains classification rules by recursively partitioning the dataset according to logical conditions that can be represented as nodes in a tree. This top-down induction of logical decision trees (TILDE) is driven by refining the node criteria according to the provided language bias $\mathcal{L}$. Unlike C4.5, TILDE is capable of inducing first-order logical decision trees (FOLDT). A FOLDT is a tree that holds logical formula containing variables instead of propositions. Blockeel and De Raedt (1998) show how each FOLDT can be converted into a logic program.

5.3.2 Event logs

An event log consists of events that pertain to process instances. A process instance is defined as a logical grouping of activities whose state changes are recorded as events. This logical grouping of activities is in accordance with a particular case identifier, that identifies process instances. In this chapter, we will continue to use the EM-BrA^{2}CE process modeling vocabulary introduced in Chapter 3. However, we will also use the following, abbreviated notation.

Let X be a set of event identifiers, P a set of case identifiers, the alphabet A a set of activity types, and E a set of event types. An event predicate is a quintuple $\mathit{Event}(x, p, a, e, t)$, where $x \in X$ is the event identifier, $p \in P$ is the case identifier, $a \in A$ the activity type, $e \in E$ the event type, and $t \in \mathbb{N}$ the time of occurrence of the event. The function $\mathit{Case} \in X \cup L \rightarrow P$ denotes the case identifier of an event or a sequence. The function $AT \in X \rightarrow A$ denotes the activity type of an event. The function $ET \in X \rightarrow E$ denotes the event type of an event. The function $\mathit{Time} \in X \rightarrow \mathbb{N}$ denotes the time of occurrence of an event. In this chapter, we are primarily interested in reconstructing a control flow-based process model from an event log. We will therefore assume that E is equal to the set $\{\mathit{completed}, \mathit{completeRejected}\}$, event types that respectively indicate the completion of a particular activity or the negation of this. The set X of identifiers has a complete ordering, such that $\forall x, y \in X : x < y \vee y < x$ and $\forall x, y \in L : x < y \Rightarrow \mathit{Time}(x) \leq \mathit{Time}(y)$.

Let an event log L be a set of sequences. Let $\sigma \in L$ be an event sequence, an ordered set of event identifiers of events pertaining to the same process instance as denoted by the case id; $\sigma = \{x \mid x \in X \wedge \mathit{Case}(x) = \mathit{Case}(\sigma)\}$. The function $\mathit{Position} \in X \times L \rightarrow \mathbb{N}_0$ denotes the position of an event with identifier $x \in X$ in the sequence $\sigma \in L$. Two subsequent event identifiers within a sequence σ can be represented as a sequence $x.y \subseteq \sigma$. We define the .-predicate as follows

$$x.y \Leftrightarrow \exists x, y \in \sigma : x < y \wedge \nexists z \in \sigma : x < z < y.$$

In the text, this predicate is used within the context of a single sequence σ which is therefore left implicit. Given that $AT(x) = a, AT(y) = b$ the information in the sequence can be further abbreviated as ab, because the order of the activity types in a sequence is the most important information for the purpose of process discovery. This notation is used to represent the event log in Figure 5.1. Each row σ_i in the event log represents a different execution sequence that corresponds to a particular process instance.

5.3.3 Petri nets

Figure 5.1(c) is a Petri net representation of a simplified driver's license application process. Petri nets represent a graphical language with a formal semantics that has been used to represent, analyze, verify, and simulate dynamic behavior (Murata, 1989). Petri nets consist of places, tokens, and arcs. *Places* (drawn as circles) can contain *tokens* and are an indication of state. Each different distribution of tokens over the places of a Petri net indicate a different state. Such a state is called a *marking*. *Transitions* (drawn as rectangles) can consume and produce tokens and represent a state change. *Arcs* (drawn as arrows) connect places and transitions and represent a flow relationship. More formally, a marked Petri net is a pair $((P, T, F), s)$ where,

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \emptyset$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of direct arcs, and
- $s \in P \rightarrow \mathbb{N}$ is a bag over P denoting the marking of the net (van der Aalst, 1997, 1998).

Petri nets are bipartite directed graphs, that means that each arc must connect a transition to a place or a place to a transition. The transitions in a Petri net can be labeled or not. Transitions that are not labeled are called *silent transitions*. Different transitions that bear the same label are called *duplicate transitions*.

The behavior of a Petri net is determined by the *firing rule*. A transition t is *enabled* iff each input place p of t contains at least one token. When a transition is enabled it can *fire*. When a transition fires, it brings about a state change in the Petri net. In essence, it consumes one token from each input place p and produces one token in each output place p of t.

A particular class of Petri nets that has been used to model processes are *workflow nets* (van der Aalst and van Hee, 2002). Workflow nets model individual process instances in isolation. A Petri net (P, T, F) is a *workflow net* iff

- P contains a *source place* i that has no incoming arcs,
- P contains a *sink place* o that has no outgoing arcs,
- and the short-circuited net $(P, T \cup \{t\}, F \cup \{(o,t),(t,i)\})$ – where t is an additional transition $t \notin T \cup P$ that connects the sink place to the source place – has a flow relationship that transitively connects every place or transition to every other place or transition (strong connectedness).

There exist efficient algorithms and tools to verify the correctness of a workflow net (van der Aalst, 1997, 1998).

5.4 Step 1: Derive Parallelism and Locality from Frequent Temporal Constraints

The starting point is the analysis of frequent constraints that hold in the event log. These constraints are used to gain insight in local dependency, non-local dependency, and parallelism relationships that exist between pairs of activities. This information is used in the language bias of the classification learner, to constrain the hypothesis space to locally discriminating preconditions when required. The aim is to facilitate the construction of a graphical model, a Petri net, from the learned preconditions.

5.4.1 Frequent Temporal Constraints

Frequent temporal constraints are temporal constraints that hold in a sufficient number of sequences σ within an event log L. In particular, a temporal constraint is considered frequent when its frequency of occurrence is above a particular threshold. In this section, we first define a number of temporal constraints that apply to a single sequence σ and then define a number of frequent temporal constraints that apply to an event log L. Let $a, b, c \in A$. The following predicates express temporal constraints that either hold or not for a particular sequence $\sigma \in L$:

$$\begin{aligned}
\mathit{Existence}(1, a, \sigma) &\Leftrightarrow \exists x \in \sigma : AT(x) = a.\\
\mathit{Absence}(2, a, \sigma) &\Leftrightarrow \nexists x, y \in \sigma : AT(x) = a \wedge AT(y) = a \wedge x \neq y.\\
\mathit{Ordering}(a, b, \sigma) &\Leftrightarrow \exists x, y \in \sigma : AT(x) = a \wedge AT(y) = b \wedge x < y.\\
\mathit{Precedence}(a, b, \sigma) &\Leftrightarrow \forall y \in \sigma : AT(y) = b, \exists x \in \sigma : AT(x) = a \wedge x < y.\\
\mathit{Response}(a, b, \sigma) &\Leftrightarrow \forall x \in \sigma : AT(x) = a, \exists y \in \sigma : AT(y) = b \wedge x < y.\\
\mathit{ChainPrec}(a, b, \sigma) &\Leftrightarrow \forall y \in \sigma : AT(y) = b, \exists x \in \sigma : AT(x) = a \wedge x.y.\\
\mathit{ChainResp}(a, b, \sigma) &\Leftrightarrow \forall x \in \sigma : AT(x) = a, \exists y \in \sigma : AT(y) = b \wedge x.y.\\
\mathit{ChainSeq}(a, b, c, \sigma) &\Leftrightarrow \exists x, y, z \in \sigma : x.y.z\\
&\quad \wedge AT(x) = a \wedge AT(y) = b \wedge AT(z) = c.
\end{aligned}$$

For an event log L, a temporal constraint C is considered frequent if its support is greater than or equal to a predefined threshold. Let C, D be temporal constraints. The support for a temporal constraint can be defined as

$$\mathit{Supp}_{\sigma \in L}(C, L) = \frac{|S|}{|T|}$$

for which $S = \{\sigma \mid \sigma \in L \wedge \mathit{Succeeds}(C, \sigma)\}$ and $T = \{\sigma \mid \sigma \in L\}$. Temporal constraints can be combined to form temporal association rules of the form $C \rightarrow D$. The support of an association rule is defined as

$$\mathit{Supp}_{\sigma \in L}(C \rightarrow D, L) = \mathit{Supp}_{\sigma \in L}(C, L).$$

The confidence of a temporal association rule is defined as

$$\mathit{Conf}_{\sigma \in L}(C \rightarrow D, L) = \frac{\mathit{Supp}_{\sigma \in L}(C \wedge D, L)}{\mathit{Supp}_{\sigma \in L}(C, L)}.$$

Temporal association rules can be considered frequent if their support and confidence are greater than or equal to a predefined threshold. It can be assumed that each activity type that occurs in the event log is relevant and should be included in the process model to be discovered. However, in event logs some activities can occur more frequently than others. The detection of frequent patterns must not be sensitive to the frequency of occurrence of a particular activity type in an event log. For instance, when an activity type a occurs infrequently, it can be the case that the temporal constraint $\mathit{ChainResp}(a, b, \sigma)$ has a large support, whereas in reality the sequence ab never occurs in L. To detect frequent temporal patterns in an event log, it is therefore more important to look at the confidence of an association rule $\mathit{Existence}(1, a, \sigma) \rightarrow \mathit{Response}(a, b, \sigma)$ than its support. Within an event log L the following frequent temporal association rules can be derived for an event log (the argument L is left implicit):

$$\begin{aligned}
Absence(2,a) &\Leftarrow \forall a \in A : \\
&\qquad Supp_{\sigma\in L}(Absence(2,a,\sigma),L) \geq t_{absence}. \\
Ordering(a,b) &\Leftarrow \forall a,b \in A : \\
&\qquad Supp_{\sigma\in L}(Ordering(a,b,\sigma),L) \geq t_{ordering}. \\
Precedence(a,b) &\Leftarrow \forall a,b \in A : \\
&\qquad Conf_{\sigma\in L}(Existence(1,b,\sigma) \rightarrow Precedence(a,b,\sigma),L) \geq t_{succ}. \\
Response(a,b) &\Leftarrow \forall a,b \in A : \\
&\qquad Conf_{\sigma\in L}(Existence(1,a,\sigma) \rightarrow Response(a,b,\sigma),L) \geq t_{succ}. \\
ChainPrec(a,b) &\Leftarrow \forall a,b \in A : \\
&\qquad Conf_{\sigma\in L}(Existence(1,b,\sigma) \rightarrow ChainPrec(a,b,\sigma),L) \geq t_{chain}.
\end{aligned}$$

$$\begin{aligned}
ChainResp(a,b) &\Leftarrow \forall a,b \in A : \\
&\qquad Conf_{\sigma\in L}(Existence(1,a,\sigma) \rightarrow ChainResp(a,b,\sigma),L) \geq t_{chain}. \\
ChainSeq(a,b,c) &\Leftarrow \forall a,b,c \in A : \\
&\qquad Supp_{\sigma\in L}(ChainSeq(a,b,c,\sigma),L) \geq t_{triple}.
\end{aligned}$$

In the above definitions, the predicates $Ordering(a,b)$ and $ChainSeq(a,b,c)$ do not account for the mutual distribution of the activity types a, b, and c. They are therefore sensitive to their frequency of occurrence. A change of these definitions in the implementation of AGNEs is deferred to future work.

It is not necessary to enumerate every temporal constraint for every $a, b, c \in A$ exhaustively. For example, the induction of frequent constraints can make use of an a-priori trick, deriving the (in)frequency of one constraint from the (in)frequency of another (Agrawal et al., 1996). In addition, it is not required to calculate every triple chain sequence, but only these frequent constraints that are required to identify short loops. The induction of frequent constraints from event logs is not the main contribution of this chapter, consequently the details of the induction of temporal association rules have been omitted from this text. Mannila et al. (1997) describe the more general problem of inducing frequent episodes from event logs.

5.4.2 Deriving Parallelism and Locality

From the induced frequent constraints, information can be derived about the parallelism and locality of pairs of activities. As a rule of thumb, parallelism between two activity types $a, b \in A$ can be assumed when it is the case in the event log that a frequently follows b and b frequently follows a. However, in the case of duplicate activities, a triple chain sequence aba or bab could just as well occur in serial. Therefore we require these triple chain sequence not to occur, unless it can be detected that a and b in fact are part of two separate parallel length-one loops. This is the case when sequences like aab, baa, abb, or bba are also frequent. Consequently, $\mathit{Parallel}(a,b)$ can be defined as follows.

$$\begin{aligned}
\forall a,b \in A : (\mathit{ChainPrec}(a,b) \vee \mathit{ChainResp}(a,b)) \wedge &\\
(\mathit{ChainPrec}(b,a) \vee \mathit{ChainResp}(b,a)) \wedge &\\
(\sim \mathit{ChainSeq}(a,b,a) \wedge \sim \mathit{ChainSeq}(b,a,b) \vee \mathit{ParallelLoop}(a,b)) &\quad\Rightarrow\quad \mathit{Parallel}(a,b).\\
\forall a,b \in A : \mathit{Parallel}(a,b) &\quad\Rightarrow\quad \mathit{Parallel}(b,a).\\
\forall a,b \in A : \mathit{ChainSeq}(a,a,b) \vee \mathit{ChainSeq}(b,a,a) \vee &\\
\mathit{ChainSeq}(a,b,b) \vee \mathit{ChainSeq}(b,b,a) &\quad\Rightarrow\quad \mathit{ParallelLoop}(a,b).
\end{aligned}$$

In the above derivation, we use the negation-as-failure $\sim$ rather than the classical negation $\neg$ to indicate that we derive the frequent temporal constraint from the absence of sequences in the log that portray this behavior. Although parallelism is difficult to detect in a one-event type setting, concurrency information can be often obtained from domain experts with great certainty. Therefore, AGNEs allows to define the strict relationships $\mathit{PriorParallel}(a,b)$ or $\mathit{PriorSerial}(a,b)$ that defeat any conclusions about the parallelism of a,b made by the above inference rule. The ability to include prior knowledge is an advantage of our approach with respect to other approaches.

Locality of a pair of activities is another type of information that can be derived from an event log. Intuitively, a is local to b if a is frequently followed by b or if b is frequently preceded by a.

$$\begin{aligned}
\forall a,b \in A : \mathit{ChainPrec}(a,b) \wedge \sim \mathit{Parallel}(a,b) &\quad\Rightarrow\quad \mathit{Local}(a,b).\\
\forall a,b \in A : \mathit{ChainResp}(a,b) \wedge \sim \mathit{Parallel}(a,b) &\quad\Rightarrow\quad \mathit{Local}(a,b).
\end{aligned}$$

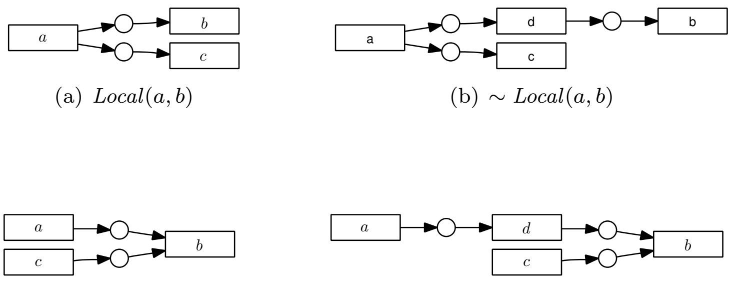
Unlike the parallelism, locality is not symmetric: $\mathit{Local}(a,b)$ does not imply $\mathit{Local}(b,a)$. Locality can also be specified as prior knowledge. Therefore, our technique allows defining the strict specifications $\mathit{PriorLocal}(a,b)$ or $\mathit{PriorNonLocal}(a,b)$ that defeat any other conclusions about locality made from the induced frequent temporal constraints.

In the presence of parallelism, it is possible, even highly likely, that not all local relationships can be detected. Consider the situation in Figure 5.4(a). Although a is local to b it is possible that the sequence a,b occurs infrequently in the event log. Because b and c are parallel activities, it is possible that the sequence a,c,b occurs far more frequently that the sequence a,b,c. In such cases, $\mathit{Local}(a,b)$ can be derived from the parallelism of b and c. However, Figure 5.4(b) depicts a situation in which such an inference cannot be made. These concerns can translated into the following derivation rule.

$$\begin{aligned}
\forall a,b \in A : \exists c \in A : \mathit{Parallel}(b,c) \wedge \sim \mathit{Parallel}(a,c) \wedge \sim \mathit{Parallel}(a,b) \wedge &\\
(\mathit{ChainPrec}(a,c) \vee \mathit{ChainResp}(a,c)) \wedge &\\
\nexists d \in A : (\mathit{ChainResp}(d,b) \vee \mathit{ChainPrec}(d,b)) \wedge \sim \mathit{Parallel}(d,b) \Rightarrow \mathit{Local}(a,b).&
\end{aligned}$$

The same reasoning for an AND-join can be applied to an AND-split, as displayed in Figure 5.4(c) and 5.4(d). The following derivation rule applies.

$$\begin{aligned}\forall a, b \in A : \exists c \in A : &\mathit{Parallel}(a,c) \wedge \sim \mathit{Parallel}(b,c) \wedge \sim \mathit{Parallel}(a,b) \wedge \\ &(\mathit{ChainPrec}(c,b) \vee \mathit{ChainResp}(c,b)) \wedge \\ \nexists d \in A : (\mathit{ChainResp}(a,d) \vee \mathit{ChainPrec}(a,d)) \wedge &\sim \mathit{Parallel}(a,d) \Rightarrow \mathit{Local}(a,b).\end{aligned}$$



(a) $\mathit{Local}(a,b)$ (b) $\sim \mathit{Local}(a,b)$

(c) $\mathit{Local}(a,b)$ (d) $\sim \mathit{Local}(a,b)$

Figure 5.4: Deriving locality from parallelism

5.5 Step 2: Generate Artificial Negative Events

A second step in the process discovery technique is the induction of artificial negative events. As mentioned in the introduction, event logs rarely contain information about transitions that are not allowed to take place. This makes process discovery an inherently unsupervised learning problem. To make a tradeoff between overly general or overly specific process models, learners make additional assumptions about the given event sequences. Such assumptions are part of the inductive bias of a learner. Process discovery algorithms generally include the assumption that event logs portray the complete behavior of the underlying process and implicitly use this completeness assumption to make a tradeoff between overly general and overly specific process models. Our technique makes this completeness assumption configurable by explicitly inducing artificial negative information from the event log.

5.5.1 A Configurable Completeness Assumption

For processes that contain a lot of recurrent and concurrent behavior, the completeness assumption can become problematic. For example, a process containing five parallel activities (ten parallel pairs) that are placed in a loop, has $\sum_{i=1}^{n}(5!)^i$ different possible execution sequences (n being the maximum number of allowed loops).

The problem of recurrent behavior is addressed by restricting the window size (parameter: *windowSize*). Window size is the number of events in the subsequence one hopes to detect at least once in the sequence database. The larger the window size, the less probable that a similar subsequence is contained by the other sequences in the event log. A limited window size can be advantageous in the presence of loops (recurrent behavior) in the underlying process. Limiting the window size to a smaller sub-sequence of the event log, makes the completeness assumption less strict. An unlimited window size ($windowSize = -1$) results in the most strict completeness assumption.

The problem of concurrent behavior is addressed by exploiting some available parallelism information, discovered by induction or provided as prior knowledge by a domain expert. Given a subsequence and parallelism information, all parallel variants of the subsequence can be calculated. Taking into account the parallel variants of a subsequence makes the completeness assumption less strict. The function $ParallelVariant(\tau)$ returns – through backtracking – all parallel variants of a sequence by permuting the activities in each sub-sequence of τ while preserving potential dependency relationships among non-parallel activities. The function $AllParallelVariants(\tau)$ returns the set of all parallel variants that can thus be obtained.

The calculation of parallel variants is illustrated in Figure 5.5. Given the event sequence τ, and information about parallelism, five parallel variants can be calculated. At position two, *bcde* is the shortest sub-sequence that contains all activities parallel to b. Because b and c are not parallel, it can be the case that c is dependent on b. Likewise, because d and e are parallel, it can be the case that c is dependent on b. Therefore, $\tau^{\|}$ consists of all permutations of the sub-sequence in which c occurs after b and e occurs after d. At position three, *cde* is the shortest sub-sequence with activities all parallel to c.

given:

τ	a	b	c	d	e	f

$Parallel(b, d)$.
$Parallel(b, e)$.
$Parallel(c, d)$.
$Parallel(c, e)$.

parallel variants:

$\tau^{\|}$	a	d	b	e	c	f
$\tau^{\|}$	a	d	e	b	c	f
$\tau^{\|}$	a	b	d	c	e	f
$\tau^{\|}$	a	b	d	e	c	f
$\tau^{\|}$	a	b	c	d	e	f

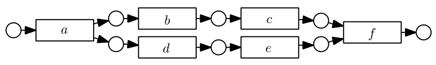


Figure 5.5: parallel variant calculation

5.5.2 Generating Negative Events

Negative events record that at a particular position in an event sequence, a particular event cannot occur. At each position k in each event sequence τ_i, it is examined which negative events can be recorded for this position. Algorithm 2 gives an overview of the negative event induction and is discussed in the next paragraphs. In a first step, the event log is made more compact, by grouping process instances $\sigma \in L$ that have identical sequences into grouped process instances $\tau \in M$ (line 1). By grouping similar process instances, searching for similar behavior in the event log can be performed more efficiently.

In the next step, all negative events are induced for each grouped process instance (lines 2–12). Making a completeness assumption about an event log boils down to assuming that behavior that does not occur in the event log, should not occur in the process model to be learned. Negative examples can be introduced in grouped process instances τ_i by checking at any given positive event $x_k \in \tau_i$ whether another event of interest z_k of activity type $b \in A \backslash \{AT(x_k)\}$ also could occur. For each event $x_k \in \tau_i$, it is tested whether there exists a similar sequence $\tau_j^{\|} \in \mathit{AllParallelVariants}(\tau_j) : \tau_j \neq \tau_i$ in the event log in which at that point a state transition y_k has taken place that is similar to z_k (line 6). If such a state transition does not occur in any other sequence, such behavior is not present in the event log L. This means under the completeness assumption that the state transition cannot occur. Consequently, z_k can be added as a negative event at this point k in the event sequence τ_i (lines 7–8). On the other hand, if a similar sequence is found with this behavior, no negative event is generated.

Finally, the negative events in the grouped process instances are used to induce negative events into the similar non-grouped sequences. If a grouped sequence τ contains negative events at position k, then the ungrouped sequence σ contains corresponding negative events at position k. At each position, a large number of negative events can generally be generated. To avoid an imbalance in the proportion of negative versus positive events the addition of negative events can be manipulated with a negative event injection probability π (line 13). π is a parameter that influences the probability that a corresponding negative event is recorded in an ungrouped trace σ. The smaller π, the less negative events are generated at each position in the ungrouped event sequences. A value of $\pi = 1.0$ means that every induced negative event for a grouped sequence is included in every similar, corresponding, non-grouped sequence. A value of $\pi = 0$ will result in no negative events being induced for any of the corresponding, non-grouped sequences.

Figure 5.6 illustrates how in an event log of two (grouped) sequences τ_1 and τ_2 artificial negative events can be generated. The event sequences originate from a simplified driver's license process, depicted at the bottom of the figure. Given the parallelism information, $\mathit{Parallel}(e, f)$, the event sequences each have two parallel variants. When generating negative events into event sequence τ_1, it is examined whether instead of the first event b the events c, d, e, f, g, h, or i could also have occurred at the first position. Because there is no similar sequence $\tau_j^{\|}$ in which c, d, e, f, g, h, or i occur at this position, it can be concluded that they are negative events. Consequently $c_n, d_n, e_n, f_n, g_n, h_n$, and i_n are added as negative events at this position. Other artificial negative events are generated in a similar fashion. Notice that history-dependent processes generally will require a larger window

Algorithm 2 Generating artificial negative events

```
1: Group similar sequences $\sigma \in L$ into $\tau \in M$
2: for all $\tau_i \in M$ do
3:   for all $x_k \in \tau_i$ do
4:     $k = Position(x_k, \tau_i)$
5:     for all $b \in A \backslash \{AT(x_k)\}$ do
6:       if $\nexists \tau_j^{||} \in AllParallelVariants(\tau_j) : \forall \tau_j \in M \wedge \tau_j \neq \tau_i \wedge$
         $\forall y_l \in \tau_j^{||}, Position(y_l, \tau_j^{||}) = l = Position(x_l, \tau_j), k - windowSize < l < k : AT(y_l) = AT(x_l) \wedge$
         $y_k \in \tau_j^{||}, Position(y_k, \tau_j^{||}) = k, AT(y_k) = b$ then
7:         $z_k =$ event with $AT(z_k) = b, ET(z_k) = completeRejected$
8:         recordNegativeEvent($z_k, k, \tau_i$)
9:       end if
10:    end for
11:  end for
12: end for
13: Induce negative events in the non-grouped sequences $\sigma \in L$ according to an
    injection frequency $\pi$
```

size to correctly detect all non-local dependencies. In the example of Figure 5.6, an unlimited window size is used. Should the window size be limited to 1, for instance, then it would no longer be possible to take into account the non-local dependency between the activity pairs *c–g* and *d–h*. In the experiments at the end of this paper, an unlimited window size has been used (parameter value -1).

5.6 Step 3: Learn the Preconditions of each Activity Type

Given an event log supplemented with artificially generated negative events, it is possible to represent process discovery as a multiple classification learning problem that learns the conditions that discriminate between the occurrence of either a positive or a negative event for each activity type. Our process discovery technique makes use of TILDE, an existing multi-relational classification learner, to perform the actual classification learning.

5.6.1 Process Discovery as Classification

The motivation for representing process discovery as a classification problem is that it has the potential to deal with so-called **duplicate tasks**. Duplicate tasks refer to the reoccurrence of identically labeled transitions, homonyms, in different contexts within the event logs. Classification learning can detect the different execution contexts for these transitions and derive different preconditions for them.

given:

τ_1	b_p	c_p	e_p	f_p	g_p	i_p
τ_2	b_p	d_p	f_p	e_p	h_p	i_p

Parallel(e, f).

parallel variants:

$\tau_1^{\parallel}$	b_p	c_p	f_p	e_p	g_p	i_p
$\tau_1^{\parallel}$	b_p	c_p	e_p	f_p	g_p	i_p
$\tau_2^{\parallel}$	b_p	d_p	f_p	e_p	h_p	i_p
$\tau_2^{\parallel}$	b_p	d_p	e_p	f_p	h_p	i_p

negative events:

τ_1	b_p	c_p	e_p	f_p	g_p	i_p
	c_n	b_n	b_n	b_n	b_n	b_n
	d_n	e_n	c_n	c_n	c_n	c_n
	e_n	f_n	d_n	d_n	d_n	d_n
	f_n	g_n	g_n	g_n	e_n	e_n
	g_n	h_n	h_n	h_n	f_n	f_n
	h_n	i_n	i_n	i_n	h_n	g_n
	i_n				i_n	h_n
τ_2	b_p	d_p	e_p	f_p	h_p	i_p
	c_n	b_n	b_n	b_n	b_n	b_n
	d_n	e_n	c_n	c_n	c_n	c_n
	e_n	f_n	d_n	d_n	d_n	d_n
	f_n	g_n	g_n	g_n	e_n	e_n
	g_n	h_n	h_n	h_n	f_n	f_n
	h_n	i_n	i_n	i_n	g_n	g_n
	i_n				i_n	h_n

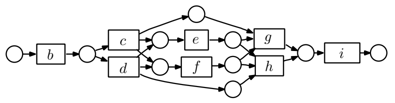


Figure 5.6: DriversLicenseI – Generating artificial negative events for an event log with two sequences

Transforming these preconditions into graphs will eventually result in duplicate, homonymic activities in the graph model that correspond to the different usage contexts. Techniques for process discovery, such as heuristics miner and genetic miner, which have causal matrices as internal representation (Alves de Medeiros et al., 2007; Weijters et al., 2006), are unable to discover duplicate activities. Alves de Medeiros (2006) presents a non-trivial extension of the genetic miner that includes duplicate tasks.

As mentioned in Section 5.3.1, multi-relational data mining allows searching for patterns among different rows within a data set. For the purpose of discovering history-dependent patterns, this multi-relational property is much desired, as it allows learning patterns that occur in the *entire history* of the event. Alternatively, the history of each event could in part be represented as extra propositions, for instance by including all preceding positive events as extra columns in the event log. This propositional representation would have many difficulties.

Being able to detect such non-local, historic patterns in an event log can also work counter-intuitive. A non-local relationship might have more discriminating power than a local one and can therefore be preferred by a learner. Unfortunately, an excess of non-local patterns makes it more difficult to generate a graphical model, a workflow net, containing local connections. Because TILDE allows specifying a language bias with dynamic refinement (Blockeel and De Raedt, 1998), it becomes possible to constrain the hypothesis space to locally discriminating patterns whenever necessary. One technique is the dynamic generation of constants, that can be used to constrain the combinations of activity type constants that are to be considered for a particular language bias construct. Additionally, it is also possible to constrain the occurrence of particular literals in a hypothesis $\mathcal{H}$, given the presence or absence of other literals that are already part of the hypothesis.

5.6.2 The Language Bias of AGNEs

The classification task of TILDE is to predict whether for a given activity type $a \in A$, at a given time point $t \in \mathbb{N}$ in a given sequence $\sigma \in L$, a positive or a negative event takes place. In the case of a positive event, the target predicate evaluates to $\mathit{Class}(a, \sigma, t, \mathit{completed})$. In the case of a negative event, the target predicate evaluates to $\mathit{Class}(a, \sigma, t, \mathit{completeRejected})$. In the language bias, the target activity, indicated by a, will be used for dynamically constraining the combinations of activity type constants generated.

The primary objective of AGNEs being the construction of a graphical model from an event log, the language bias consists of a logical predicate that can represent the conditions under which a Petri net place contains a token. This predicate is called the "no-sequel" predicate, $\mathit{NS}(a_1, a, \sigma, t)$. Let $a_1, a \in A$ be activity types, $\sigma \in L$ the sequence of a process instance, and t the time of observation. The NS predicate is defined as follows:

$$\begin{aligned}\forall a_1, a \in A, t \in \mathbb{N} : \exists x \in \sigma : AT(x) = a_1 \ \wedge \ \mathit{Time}(x) < t \\ \wedge \nexists y \in \sigma : AT(y) = a \ \wedge \ \mathit{Time}(x) < \mathit{Time}(y) < t \quad \Rightarrow \mathit{NS}(a_1, a, \sigma, t).\end{aligned}$$

In the remainder of this text, the arguments σ and t will be implicitly assumed and therefore left out. The predicate $\mathit{NS}(a_1, a)$ evaluates to true when at the time of observation, an activity a_1 has completed, but has not (yet) been followed by an activity a. In combination with conjunction ($\wedge$), disjunction ($\vee$) and negation-as-failure ($\sim$), the $\mathit{NS}(a_1, a)$ predicate makes it possible to learn fragments of Petri nets using a multi-relational classification learner. The fragments included in the language bias are illustrated in Figure 5.7.

A local sequence of two transitions labeled $a_1, a \in A$ in a Petri net can be represented by $\mathit{NS}(a_1, a)$. Figure 5.7(a) shows a graphical representation of this predicate. The following constraints apply:

sequence:	$\mathit{NS}(a_1, a)$	
constraints:	$\mathit{Local}(a_1, a)$.	(1)
	$\nexists l \in \mathcal{H} : l = [\mathit{NS}(\_, a_1)]$.	(2)

The conversion from NS-based preconditions into Petri nets requires the conditions to refer to local, immediately preceding events as much as possible. Therefore, constraint (1) requires to restrict the generation of constants a_1 to constants that are local to a. In constraint (1) we do not require a_1 and a to be different activity types. This is sufficient for the discovery of length-one loops. Graphically, a conjunction of NS constructs can be considered to be the logical counterpart of a single Petri net place. Besides the case of length-one loops – that can be expressed with a single NS construct – there is no reason for a Petri net place to contain both an input and an output arc that is connected to the same transition.

Constraint (2) has been put in place to keep a multi-relational learner from constructing such useless hypotheses. The constraint stipulates that the construct $NS(a_1, a)$ can only be added to the current hypothesis $\mathcal{H}$, if $\mathcal{H}$ does not already contain a logical condition that boils down to an output arc towards a_1.

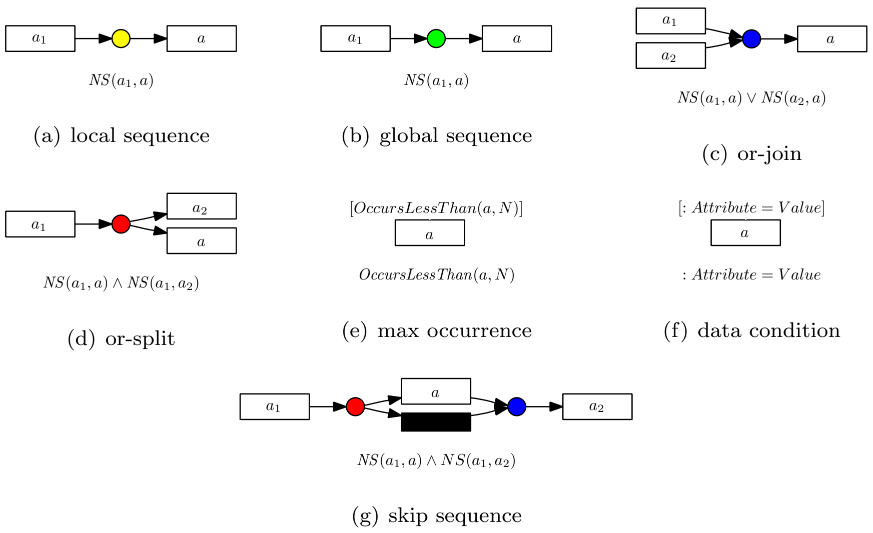


Figure 5.7: Petri net patterns in the language bias

The same $NS(a_1, a)$ construct, with different constraints, can be used to keep track of a global sequence (see Figure 5.7(b)). Global sequences are used to represent non-local, non-free choice constructs. We require TILDE only to consider global sequence between activity types for which both a precedence and a response relationship has been detected from the event log, hence constraint (3).

global sequence:	$NS(a_1, a)$	
constraints:	$Precedence(a_1, a) \wedge Response(a_1, a).$	(3)
	$\exists l \in \mathcal{H}.$	(4)

Because we assume that any transition must be locally connected, that is connected to other transitions to which it is local in the event log, we require as an additional constraint that the global sequence construct must not be added first to any hypothesis. In other words, the addition of a global sequence literal to the hypothesis, requires that the hypothesis $\mathcal{H}$ currently under consideration contains at least one literal.

An or-join can be represented as a disjunction of different NS constructs. This is represented in Figure 5.7(c). The input place of a has two incoming arcs from transitions a_1 and a_2. In an or-join transitions a_1 and a_2 must not be parallel and must both be local to a (5).

or-join: $NS(a_1, a) \vee NS(a_2, a)$

constraints:

$$a \neq a_1 \neq a_2 \wedge Local(a_1, a) \wedge Local(a_2, a) \wedge \sim Parallel(a_1, a_2). \quad (5)$$

$$Absence(2, a) \Rightarrow \forall b \in A \backslash \{a_1, a_2\} : Ordering(a, b) \Rightarrow Ordering(a_1, b) \wedge Ordering(a_2, b). \quad (6)$$

$$Absence(2, a_1) \wedge Absence(2, a_2) \Rightarrow (\forall b \in A : Ordering(a_1, b) \Rightarrow Ordering(a, b)) \wedge (\forall c \in A : Ordering(a_2, c) \Rightarrow Ordering(a, c)). \quad (7)$$

Constraints (6) and (7) require that a_1 and a_2 are only eligible as input transitions for an or-join, when they have a similar post-ordering as the a transition. In the case of duplicate activities, these constraints cannot be meaningfully enforced.

An or-split can be represented as a conjunction of different *NS* constructs. This is represented in Figure 5.7(d). In an or-split, the outgoing transitions are not parallel, moreover, both outgoing transitions are local to the transition from which the or-split originates (8). Constraints (9) and (10) impose that for a and a_2 to be part of an or-split, they must have similar pre-orderings as a_1. Of course, the constraints must be relaxed when the activities can occur more than once in a process instance.

or-split: $NS(a_1, a) \wedge NS(a_1, a_2)$

constraints:

$$a \neq a_1 \neq a_2 \wedge Local(a_1, a) \wedge Local(a_1, a_2) \wedge \sim Parallel(a, a_2) \wedge \sim Local(a, a_2). \quad (8)$$

$$Absence(2, a_1) \Rightarrow \forall b \in A : Ordering(b, a_1) \Rightarrow Ordering(b, a) \wedge Ordering(b, a_2). \quad (9)$$

$$Absence(2, a) \wedge Absence(2, a_2) \Rightarrow (\forall b \in A \backslash \{a_1\} : Ordering(b, a) \Rightarrow Ordering(b, a_1)) \wedge (\forall c \in A \backslash \{a_1\} : Ordering(c, a_2) \Rightarrow Ordering(c, a_1)). \quad (10)$$

$$\nexists l \in \mathcal{H} : l = [NS(\_, a_1)]. \quad (11)$$

An or-split is a logical conjunction of *NS* constructs, that represents a single Petri net place. As it was the case for the local sequence construct, there is no reason for a Petri net place to contain both an input and an output arc that is connected to the same transition. Similarly to the constraint (2) attached to a local sequence, constraint (11) has been put in place to keep a multi-relational learner from adding $NS(a_1, a) \wedge NS(a_1, a_2)$ to the current hypothesis $\mathcal{H}$, if it already contains a logical condition that boils down to an output arc towards a_1.

The language bias of TILDE is limited to conjunctions and disjunctions of *NS* constructs of length two and three. Or-splits and or-joins that involve more activity types are obtained by grouping conjunctions and disjunctions of *NS* constructs into larger conjunctions and disjunctions in step 4. However, this limitation in length sometimes leads to TILDE make inadequate refinements. Solving this language bias issue, requires constructing a proprietary ILP classification algorithm that during each refinement step allows considering conjunctions of *NS* constructs of variable lengths. We leave this improvements to future work.

A skip sequence, depicted in Figure 5.7(g), has the same logical structure as an or-split, but has different constraints imposed on it. Whereas the outgoing activities of an or-split cannot be local (8), it is a requirement that the skip-activity a is local to a_2 (12).

skip sequence: $NS(a_1, a) \wedge NS(a_1, a_2)$

constraints: $a \neq a_1 \neq a_2 \wedge Local(a_1, a) \wedge Local(a_1, a_2) \wedge \sim Parallel(a, a_2) \wedge Local(a, a_2).$ (12)

$Absence(2, a_1) \Rightarrow \forall b \in A : Ordering(b, a_1) \Rightarrow Ordering(b, a) \wedge Ordering(b, a_2).$ (13)

$Absence(2, a) \wedge Absence(2, a_2) \Rightarrow (\forall b \in A \backslash \{a_1\} : Ordering(b, a) \Rightarrow Ordering(b, a_1)) \wedge (\forall c \in A \backslash \{a_1\} : Ordering(c, a_2) \Rightarrow Ordering(c, a_1)).$ (14)

$\nexists l \in \mathcal{H} : l = [NS(\_, a_1)].$ (15)

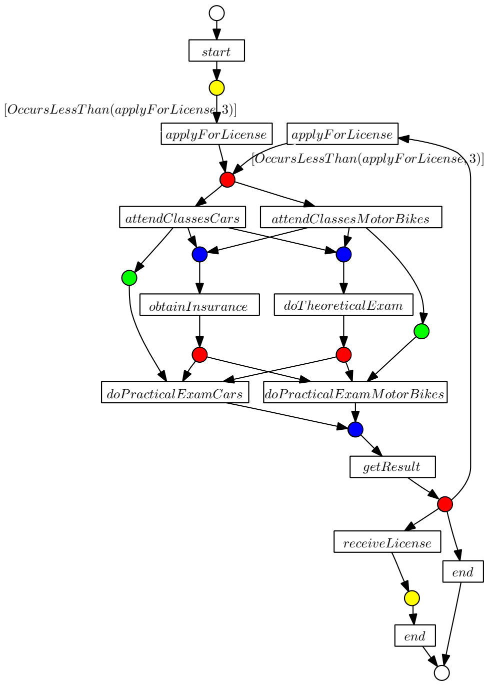
As a skip sequence is also a logical conjunction of *NS* constructs, it addition to $\mathcal{H}$ is constrained by $\mathcal{H}$ already containing a_1 on the outflow (15).

The language bias is extended with two non-Petri net constructs. The first construct, *OccursLessThan*, is defined as follows. Let $a_1, a \in A$ be activity types, σ a sequence of events pertaining to one process instance and t be the time of observation:

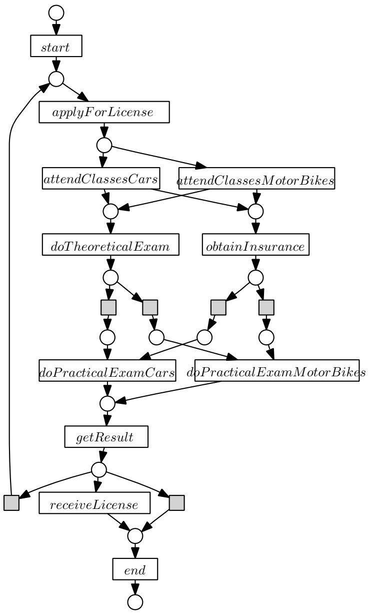
$$OccursLessThan(a, n, \sigma, t) \Leftarrow X = \{x \mid x \in \sigma \wedge AT(x) = a \wedge Time(x) < t\} \wedge |X| < n.$$

Graphically, the *OccursLessThan* precondition can be represented as a guard condition on a transition. This is depicted in Figure 5.7(e). The second non-Petri net construct deals with case data conditions and can be given the semantics of a guard condition in a colored Petri net, as depicted in Figure 5.7(f).

Figure 5.8 displays the mining results after applying heuristics miner and AGNEs on the DriversLicensel event log. Figure 5.8(c) shows how the preconditions in the Petri net can be represented as conjunctions and disjunctions of *NS* atoms. AGNEs is capable of detecting the different contexts in which the activities *applyForLicense* and *end* can take place and derives different preconditions for these activities. In addition, our technique has detected the non-local, non-free choice construct between the activities *attendClassesCars–doPracticalExamCars* and *attendClassesMotorBikes–doPracticalExamMotorBikes* and the maximum recurrence of the activity *applyForLicense*. However, having learned the transitions rules that determine whether a positive or a negative event occurs, for each activity type in the event log, is not sufficient for the construction of a graphical model. Roughly speaking, we now have individual Petri net fragments that are to be puzzled together into a sound workflow net. This problem is addressed in the next section.



(a) result of AGNEs



(b) result of heuristics miner

	activity	precondition
a	start	*true*
b	applyForLicense	$NS(a,b)$
b	applyForLicense	$(NS(i,b) \wedge NS(i,j) \wedge NS(i,k)) \wedge OccursLessThan(b,3)$
c	attendClassesCars	$NS(b,c) \wedge NS(b,d)$
d	attendClassesMotorBikes	$NS(b,c) \wedge NS(b,d)$
e	obtainInsurance	$NS(c,e) \vee NS(d,e)$
f	doTheoreticalExam	$NS(c,f) \vee NS(d,f)$
g	doPracticalExamCars	$(NS(f,g) \wedge NS(f,h)) \wedge (NS(e,g) \wedge NS(e,h)) \wedge NS(c,g)$
h	doPracticalExamMotorBikes	$(NS(f,g) \wedge NS(f,h)) \wedge (NS(e,g) \wedge NS(e,h)) \wedge NS(d,h)$
i	getResult	$NS(g,i) \vee NS(h,i)$
j	receiveLicense	$NS(i,b) \wedge NS(i,j) \wedge NS(i,k)$
k	end	$NS(j,k)$
k	end	$NS(i,b) \wedge NS(i,j) \wedge NS(i,k)$

(c) A representation with activity preconditions

Figure 5.8: DriversLicenseI – the result AGNEs and heuristics miner on a complete, zero noise event log

5.7 Step 4: Transform the Preconditions into a Petri Net

After discovering frequent temporal patterns, and supplementing the event log with artificial negative events, AGNEs runs TILDE (Blockeel and De Raedt, 1998) that is supplied with the language bias discussed in the previous section. As discussed in Section 5.3.1, TILDE constructs for each activity type a logical decision tree (LDT) that best partitions the negative and positive events. Blockeel and De Raedt show how a LDT can be transformed into an equivalent logic program. In this section, we show how the logic programs produced by TILDE can be transformed into a Petri net.

5.7.1 Rule-level Pruning

In the previous step, TILDE has learned preconditions for each activity type independently. In a Petri net, the preconditions for each activity type are nonetheless interrelated. Therefore, the logic programs of activity preconditions are submitted to several rule-level pruning steps, to make sure that they do not produce redundant duplicate places in the Petri net to be constructed. These pruning steps occur among the conditions within a single precondition, within a set of preconditions to the same activity type and among preconditions of activity types that pertain to the same or-split. Algorithm 3 provides an overview of these procedures. In the remainder of this section, these different pruning steps will be discussed.

The logic programs constructed by TILDE contain rules that classify the occurrence of either a positive or a negative event. By construction, the language bias of AGNEs is such that TILDE will never consider a negation of an *NS* construct to be explanatory for the occurrence of a positive event. Therefore, TILDE will never construct a tree in which a right leaf predicts a positive event. The latter entails that, in equivalent the logic program, the rules with a $class(a, \sigma, t, comleted)$ rule head can be taken from the logic programs without loss of information (line 1).

In a second step, a number of intra-rule pruning operations take place (lines 2–6). The top-down refinement of hypotheses, can result in the derivation of logically redundant conditions. These logical redundancies are removed for each rule (line 3). Each rule in the logic program consists of conjunctions of groups of *NS* constructs. A conjunction of a pair of or-split constructs that originate from the same activity a_1 can be combined into a larger or-split (line 4). Likewise, a conjunction of a pair of or-joins can be combined into a larger or-join (line 5).

In a third step, a number of inter-rule pruning operations are performed for the preconditions that pertain to each activity type. In particular, a precondition that subsumes another precondition for the same activity type, is redundant and thus removed from consideration (lines 10–12). Furthermore, it is examined whether a more specific or-join can be constructed out of the groups of *NS* constructs within the different preconditions of the same rule (lines 13–15). Finally, all rules are examined to find the most specific or-split condition from the preconditions extracted by TILDE (lines 18–26).

Algorithm 3 Rule-level pruning

```
 1: $R^+ = \{r \in R \mid r$ has rule head $class(a, \sigma, t, comleted)$ $\}$.
    // intra-rule pruning:
 2: for all $r \in R^+$ do
 3:   reduce $r$ according to $(NS_1 \vee NS_2) \wedge NS_1 \equiv NS_1$.
 4:   group or-splits: $NS(a_1, a) \wedge NS(a_1, a_2) \in r$ and $NS(a_1, a) \wedge NS(a_1, a_3) \in r$
      into $NS(a_1, a) \wedge NS(a_1, a_2) \wedge NS(a_1, a_3)$.
 5:   group or-joins: $NS(a_1, a) \vee NS(a_2, a) \in r$ and $NS(a_3, a) \vee NS(a_4, a) \in r$
      into $NS(a_1, a) \vee NS(a_2, a) \vee NS(a_3, a) \vee NS(a_4, a)$.
 6: end for
    //inter-rule pruning:
 7: for all $a \in A$ do
 8:   $R^+_a = \{r \in R^+ \mid r$ is a precondition of $a\}$
 9:   for all $r \in R^+_a$ do
10:     if $\exists s \in R^+_a$ : $s$ is more specific than $r$ then
11:       remove $r$.
12:     end if
13:     if $\exists s \in R^+_a$ : $s$ combined with $r$ lead to a more specific or-join than $r$
        then
14:       replace the or-join in $r$ with the more specific or-join.
15:     end if
16:   end for
17: end for
    //keep the most specific or-split:
18: for all $a \in A$ do
19:   $R^{split}_a = \{r \in R^+ \mid r$ contains an or-split going out $a$ $\}$.
20:   $s$ = the most specific or-split by combining the or-splits going out $a$ in
      $R^{split}_a$.
21:   for all $r \in R^{split}_a$ do
22:     if $s$ is more specific than $r$ then
23:       replace the or-split in $r$ with the or-split in $s$.
24:     end if
25:   end for
26: end for
```

5.7.2 Petri Net Construction

Given a pruned rule set of preconditions for each activity type, the construction of a Petri net is fairly straightforward. Algorithm 4 describes the procedure in detail. Each induced precondition corresponds to a different transition in a Petri net to be constructed (lines 2–6). Because AGNEs may produce several preconditions for an activity type, the constructed Petri net may contain duplicate transitions. These duplicate transitions correspond to the different contexts in which a particular activity type can be performed.

Each group ν_i of NS constructs within the precondition of a transition corresponds to a particular input place of this transition (lines 9–13). Such a group can correspond to an a sequence, a global sequence, an or-split, an or-join, or a skip sequence. Transitions can share input places. If a particular input place has not yet been created for another transition, it is created and labeled with its precondition. The output arcs of the newly constructed place connect the place to all transitions that have the precondition of the place as a conjunctive part in their precondition (lines 14–16). The input arcs of the newly constructed place originate from transitions $u \in T$ that bear labels that correspond to the incoming activities $NS(c, \_)$ of ν_i (lines 17–23). However, in the case of duplicate transitions, not every transition u necessarily is to be connected to the newly constructed place. As a heuristic, the proportion of frequent triple chains $ChainSeq(c, b, a)$ compared to the total of triple chains that would be make possible by connecting a transition u to the new place, must be higher than a threshold $t_{connect}$ (lines 19–20). In other words $t_{connect}$ is the proportion of triples triples that can occur by connecting a transition u to the new place, that are frequent.

Having constructed a Petri net graph, the graph construction algorithm applies two final pruning steps. In a first step, the algorithm removes redundant places, that correspond to redundant global sequences in the event log (line 27). Transitions that produce tokens on multiple output places can give rise to unwanted concurrent behavior. Therefore, in a second pruning step, the algorithm merges pairs of output nodes that are input nodes to transitions with activity types that are all pairwise serial (line 28).

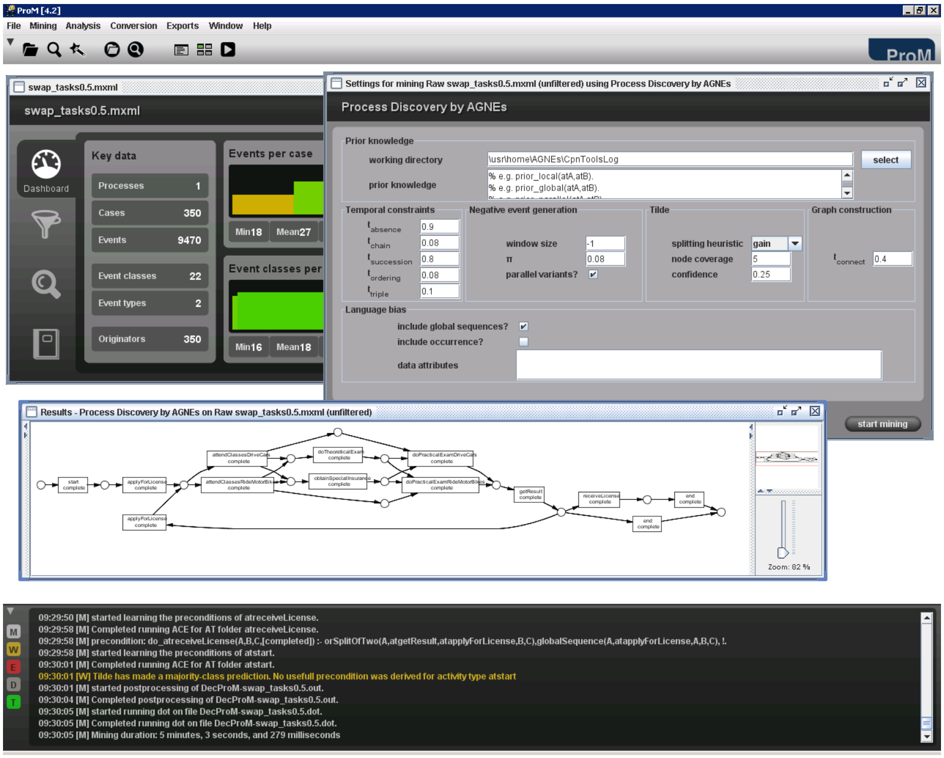
5.8 Implementation

AGNEs has been implemented in SWI-Prolog (Wielemaker, 2003). In particular, the frequent temporal constraint induction, the artificial negative event generation, the language bias constraints, and the pruning and graph construction algorithms all have been written in Prolog. As mentioned before, AGNEs makes use of TILDE, an existing multi-relational classifier (Blockeel and De Raedt, 1998), available in the ACE-ilProlog data mining system (Blockeel et al., 2002). To be able to benefit from the facilities of the ProM framework, a plugin was written that makes AGNEs accessible in ProM [1]. Figure 5.9 depicts a screen shot of AGNEs in ProM.

[1]The AGNEs plugin is available from `http://www.processintelligence.be`.

Algorithm 4 Petri net construction

```
1: $R =$ the set of pruned rules
   // construct all transitions t:
2: for all $r \in R$ do
3:    $a = target(r)$
4:    $\nu = body(r)$
5:    draw a transition $t \in T$, such that $label(t) = a$ and $precondition(t) = \nu$
6: end for
7: draw a source place $p_{source}$ and a sink place $p_{sink}$, draw the start transition
   $t_{start}$ and the stop transitions $t_{end}$ and connect them properly
8: for all $t \in T$ do
9:    $\nu = precondition(t)$
10:   $a = label(t)$
      // construct the input places of t:
11:   for all $\nu_i \in \nu : \nu_i$ is a group of $NS$-constructs do
12:      if $\nexists p \in P : condition(p_i) = \nu_i$ then
13:         draw a place $p_i$ with $condition(p_i) = \nu_i$
            // construct the output arcs of $p_i$:
14:         for all $u \in T : \mu = precondition(u) \wedge \nu_i \in \mu$ do
15:            draw an arc from $p_i$ to $u$
16:         end for
            // construct the input arcs of $p_i$:
17:         for all $b \in A : NS(b, \_) \in \nu_i$ do
18:            for all $u \in T : label(u) = b$ do
19:               if $\forall c \in A : NS(c, \_) \in \mu \wedge \mu = precondition(u) \Rightarrow$
                  $ChainSeq(c, b, a)$ has a confidence $\geq t_{connect}$
20:                then draw an arc from $u$ to $p_i$
21:               end if
22:            end for
23:         end for
24:      end if
25:   end for
26: end for
27: remove redundant places that correspond to global sequences.
28: merge pairwise serial output places.
```


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
swap_tasks0.5.mxml
Dashboard
Events per case
Event classes per
Min18 Mean27
Min16 Mean18
Results - Process Discovery by AGNEs on Raw swap_tasks0.5.mxml (unfiltered)
Zoom: 82 %
start mining
select

Key data

Processes	1
Cases	350
Events	9470
Event classes	22
Event types	2
Originators	350

Settings for mining Raw swap_tasks0.5.mxml (unfiltered) using Process Discovery by AGNEs

Process Discovery by AGNEs

Prior knowledge
working directory: \usr\home\AGNEs\CpnToolsLog
prior knowledge:
% e.g. prior_local(atA,atB).
% e.g. prior_global(atA,atB).

Temporal constraints
$t_{absence}$ 0.9
t_{chain} 0.08
$t_{succession}$ 0.8
$t_{ordering}$ 0.08
t_{triple} 0.1

Negative event generation
window size -1
π 0.08
parallel variants? ✓

Tilde
splitting heuristic gain
node coverage 5
confidence 0.25

Graph construction
$t_{connect}$ 0.4

Language bias
include global sequences? ✓
include occurrence? ☐
data attributes

```
09:29:50 [M] started learning the preconditions of atreceiveLicense.
09:29:58 [M] Completed running ACE for AT folder atreceiveLicense.
09:29:58 [M] precondition: do_atreceiveLicense(A,B,C,[completed]) :- orSplitOfTwo(A,atgetResult,atapplyForLicense,B,C),globalSequence(A,atapplyForLicense,A,B,C), !.
09:29:58 [M] started learning the preconditions of atstart.
09:30:01 [M] Completed running ACE for AT folder atstart.
09:30:01 [W] Tilde has made a majority-class prediction. No usefull precondition was derived for activity type atstart
09:30:01 [M] started postprocessing of DecProM-swap_tasks0.5.out.
09:30:04 [M] Completed postprocessing of DecProM-swap_tasks0.5.out.
09:30:05 [M] started running dot on file DecProM-swap_tasks0.5.dot.
09:30:05 [M] Completed running dot on file DecProM-swap_tasks0.5.dot.
09:30:05 [M] Mining duration: 5 minutes, 3 seconds, and 279 milliseconds
```

Figure 5.9: AGNEs in ProM 4.2

5.9 Towards Mining Access Rules

Up to this point in this chapter, we have focussed on the use of generating artificial negative events for the purpose of process discovery, i.e. related to the *start* and *complete* transition types. However, the principles outlined in this chapter, in theory also can be applied to other process mining tasks related to other transition types in the EM-BrA^{2}CE Framework, as outlined in Section 4.4.1. In this section, we briefly present how the outlined approach can also be used to discover useful access control policies from event logs. The primary intent of this section is to further illustrate the usefulness of declarative process mining techniques with respect to configuring the prior knowledge, language bias, and inductive bias. We again use TILDE as classification learner.

Consider the fictitious credit application process, of which we have generated an artificial event log from simulation as explained in Section 4.3. From this event log we want to learn for each activity type the conditions that discriminate between whether an *assigned* or an *assignRejected* event occurs at a given time *Time*, for a particular agent *Agent*, for an activity that has the business identified

BId. For instance, for an activity 'review credit', we want to learn the logic of the following transition constraint:

$$authorizationReviewCredit(BId, Agent, Time).$$

To learn to conditions that discriminate between *assigned* or *assignRejected* events, TILDE must be provided with a **language bias** that specifies the literals that may be used to express these conditions. Because access control is granted on the basis of the properties of the agent who is to be assigned to the activity, properties of the activity, and historic events, one can think of many language constructs. Considering properties of agents, we have included the seniority and department of an agent. These properties of agents are of course time-varying. In the simulation, as in reality, it is possible for agents to obtain a higher seniority or to switch departments as time goes by. The predicates *seniority* and *fromDepartment* in the language bias therefore should account for the department and seniority of an agent at the moment of assigning an agent to a task. It is possible to declaratively configure the language bias of TILDE such that that this timing property can be included in the language bias. To this end, we have included a language bias that includes the *seniority* and *fromDepartment* predicates as Event Calculus fluents (Kowalski and Sergot, 1986).With this background knowledge, we can express the meaning of the *factAdded*, *factRemoved* and *factUpdated* event types. As a consequence we can include time-dependent properties into the language bias of TILDE learners. Rather than for example including predicates like

$$\textit{fromDepartment}(\textit{Agent}, \textit{Department}), \textit{seniority}(\textit{Agent}, \textit{Seniority})$$

we can now express these properties by including a time point at which they hold:

$$\textit{fromDepartment}(\textit{Agent}, \textit{Department}, \textit{Time})s, \textit{seniority}(\textit{Agent}, \textit{Seniority}, \textit{Time})$$

effectively transforming the event log into a temporal database. Properties of the activity to which an agent is to be assigned also affect the authorization policy. The predicate *applicant*(*BId*, *Agent*, *Time*) was included in the language bias, specifying that at time *Time*, the agent *Agent*, was the applicant of the credit application *BId*. Notice that a credit application concept is the business identifier of the activity 'review credit'. Considering historic events that might be used to discriminate between positive or negative events, the predicate *historicevent*(*AT*, *BId*, *ET*, *Agent*, *Time*) was included in the language bias, indicating that prior to time point *Time*, an event of type *ET* for an activity of activity type *AT* with business identifier *BId* was provoked by an agent *Agent*.

Given that the provided language bias specifies a hypothesis space that contains the simulated authorization rules, one can only expect that TILDE is capable of discovering the correct classification rules. A small experiment reveals this is the case. In this experiment we have included 200 process instances, with the time-varying behavior that employees can randomly switch between the sales and the risk department. Figure 5.10 shows the result of the induced authorization rule as a first-order logical decision tree (FOLT). A comparison to the simulated authorization rule, depicted in Figure 5.10 reveals that nearly the same authorization logic has been detected. This demonstrates the representational power of the proposed language bias and background knowledge.

```
authorizationReviewCredit(BId,Agent,Time)
historicevent(CheckDebt,BId,assigned,Agent,Time) ?
+--yes: [assignRejected]
+--no:  fromDepartment(Agent,risk_control,Time) ?
        +--yes: applicant(BId,Agent,Time) ?
        |       +--yes: [assignRejected]
        |       +--no:  [assigned]
        +--no:  seniority(Agent,S,Time),S>=6 ?
                +--yes: [assigned]
                +--no:  [assignRejected]
```

Figure 5.10: Credit application – induced authorization rule

TILDE also allows users to provide prior knowledge that can further constrain the hypothesis space. Consider for instance, the following prior knowledge constraint: “All things considered equal, an agent with less seniority can never be assigned to an activity, whereas an agent with more seniority cannot.” If such a constraint were provided to TILDE the learner would refrain from considering hypotheses that violate this prior knowledge.

Inductive bias. One of the problems with access control policies is that they suffer from *access right accumulation.* Access control comprises a constant tension between revoking access rights to avoid misuses and granting access rights to allow workers to get their job done. However, the revocation of access rights generally causes more problems than the granting of access rights. Therefore, many access control policies grant more access rights to their users than they actually need to perform their jobs. Because of that, a user is not so much interested in the actual access control policy – this policy might already have been formally specified and automated – but rather in a more restrictive policy that reveals the access rights that are actually needed. By manipulating the **inductive bias** of a learner, this effect could in theory be obtained. In particular, it could be possible to add additional *artificial* negative events to the event log that stipulated that for a given activity, and time, a particular agent officially had the access right to perform the activity, but in reality has never (or only rarely) requested such access in similar situations. In this way, a learner could distinguish access rules that are *actually* needed, rather than the access rules that have been assigned by a modeler in the first place. We leave such a technique for manipulating the inductive bias for the purpose of access control specifications to future work.

5.10 Conclusion

Process discovery aims at accurately summarizing an event log in a structured process model. So far, the discovery of structured processes by supplementing event logs with *artificial* negative events has not been considered in the literature. The advantage is that it allows representing process discovery as a multi-relational classification problem to which existing classification learners can be applied. In this chapter, we have defined the AGNEs process discovery technique. This technique is capable of having prior knowledge constrain the hypothesis space during process discovery. In addition, our technique has a new, declarative way of dealing with incomplete event logs that diminishes the effects of concurrent and recurrent behavior on the generation of artificial negative events. Finally, it has a configurable language bias that is suitable for the discovery of Petri net patterns within event logs. The technique shows how these discovered Petri net patterns can be combined into a single Petri net graph. These declarative aspects – the inclusion of prior knowledge, the configurability of the negative event generation procedure and the language bias – are potentially useful in practical applications. In the next chapter, an evaluation of the performance of the AGNEs algorithm is provided in terms of recall and specificity.

Although the proposed process discovery technique shows a more than acceptable performance, as revealed in the next chapter, each step of the technique can still be improved in several ways. In step 1, the definitions of the $Ordering(a, b)$ and $ChainSeq(a, b, c)$ frequent temporal constraints could be altered to take into account the mutual distribution of a, b, and c activity types. Although our way of detecting parallel pairs of activity types proved to be capable of detecting parallelism in the presence of noise, the detection of parallelism by taking into account dependency to a common predecessor (Weijters et al., 2006) could also easily be incorporated. The generation of artificial negative events, which takes place in step 2, can also be made more configurable. Currently, the user can have the algorithm take into account window size and parallel variants, when looking for non-existent behavior. However, the algorithm can be made even more configurable, for instance, by specifying whether ordering matters, or by imposing on the algorithm to filter out a particular activity type. Another issue is to improve the performance of step 3, the learning of activity preconditions. A substantial performance gain can be expected, if the *NS* operator were defined on grouped rather than regular event logs. Implementing this is however non-trivial, as it would require to make substantial changes to the TILDE classification algorithm. The language bias of TILDE is limited to conjunctions and disjunctions of *NS* constructs of length two and three. Or-splits and or-joins that involve more activity types are obtained by grouping conjunctions and disjunctions of *NS* constructs into larger conjunctions and disjunctions. However, this limitation in length sometimes leads to TILDE make inadequate refinements. Solving this language bias issue, requires constructing a proprietary ILP classification algorithm that during each refinement step allows considering conjunctions of *NS* constructs of variable lengths. We leave these improvements to future work.

CHAPTER 6

An Evaluation of Declarative Process Discovery

In this chapter, we provide both an experimental and an empirical evaluation of the AGNEs process discovery technique that is introduced in the previous chapter. Section 6.1 introduces new metrics for quantifying the recall and specificity of a discovered process model vis-à-vis an event log. These metrics are used for evaluating AGNEs in the remaining sections. Section 6.2 provides an extensive experimental evaluation of the AGNEs process discovery technique. It consists of a benchmark experiment with 34 artificial event logs and comparison to four state-of-the-art process discovery algorithms. Finally, section 6.3 gives an idea of the scalability of AGNEs towards real-life event logs and the usefulness of its declarative properties.

6.1 New Recall and Precision Metrics

Discovered process models preferably allow the behavior in the event log (recall) but no other, unobserved, random behavior (specificity). Having formulated process discovery as a binary classification problem on event logs supplemented with artificial negative events, it becomes possible to use the true positive and true negative rate from classification learning theory to quantify the recall and specificity of a process model:

- **true positive rate** TP_{rate} or **recall**: the percentage of correctly classified positive events in the event log. This probability can be estimated as follows: $TP_{rate} = \frac{TP}{TP+FN}$, where TP is the amount of correctly classified positive events and FN is the amount of incorrectly classified positive events.

- **true negative rate** TN_{rate} or **specificity**: the percentage of correctly classified negative events in the event log. This probability can be estimated as follows: $TN_{rate} = \frac{TN}{TN+FP}$, where TN is the amount of correctly classified negative events and FP is the amount of incorrectly classified negative events.

Provost et al. (1998) recommended using Receiver Operator Characteristic (ROC) curves when evaluating binary decision problems. Recall and 1-specificity are the two axes on an ROC graph.

Accuracy is the sum of the true positive and true negative rate, weighted by the respective class distributions. The fact that accuracy is relative to the underlying class distributions can lead to unintuive interpretations. Moreover, in process discovery, these class distributions can be quite different and have no particular meaning. In order to make abstraction of the proportion of negative and positive events, we propose, from a practical viewpoint to attach equal importance to both recall and specificity: $acc = 0.5 recall + 0.5 specificity$. According to this definition, the accuracy of a majority-class predictor is 0.5. Flower models, such as the one in Figure **??**, are an example of such a majority-class predictors. Because a flower model represents random behavior, it has a perfect recall of the all behavior in the event log but it also has much *additional* behavior compared to the event log. Because of the latter fact, the flower model has zero specificity, and an accuracy of 0.5. Any useful process model should have an accuracy higher than 0.5.

6.1.1 Existing Metrics

Weijters et al. (2006) define a metric that has a somewhat similar interpretation as TP_{rate}: the parsing measure PM. The measure is defined as follows:

- **parsing measure** PM: the number of sequences in the event log that are correctly parsed by the process model, divided by the total number of sequences in the event log. For efficiency, the similar sequences in the event log are grouped. Let k represent the number of grouped sequences, n_i the number of process instances in a grouped sequence i, c_i a variable that is equal to 1 if grouped sequence i can be parsed correctly, and 0 if grouped sequence i cannot be parsed correctly. The parsing measure can be defined as follows Weijters et al. (2006):

$$PM = \frac{\sum_{i=1}^{k} n_i c_i}{\sum_{i=1}^{k} n_i}.$$

PM is a coarse-grained metric. A single missing arc in a Petri net can result in parsing failure for all sequences. A process model with a single point of failure is generally better than a process model with more points of failure. This is not quantified by the parsing measure PM. It has been included in the experiments because it gives an idea about the soundness of the discovered process models. A disadvantage of the current ProM 4.2 implementation PM that it converts process models to heuristics nets, and performs log parsing via the heuristics net. The conversion to a heuristics net requires duplicate activities not to share any input or output activities.

Rozinat and van der Aalst (2008) define two metrics that have a somewhat similar interpretation as TP_{rate} and TN_{rate}: the fitness metric f and the advanced behavioral appropriateness metric $a_B^{'}$:

- **fitness** f: Fitness is a metric that is obtained by replaying each (grouped) sequence in the event log in a workflow net. At the start of the sequence replay, f has an initial value of one. During sequence replay, the transitions in the workflow net will produce and consume tokens to reflect the state transitions of the sequence replay. However, the proportion of tokens that must additionally be created during sequence replay, so as to *force* a transition to fire, is subtracted from this initial value. Likewise, the fitness measure f punishes for extra behavior by subtracting the proportion of remaining tokens relative to the total number of produced tokens from this initial value. Let k represent the number of grouped sequences, n_i the number of process instances, c_i the number of tokens consumed, m_i the number of missing tokens, p_i the number of produced tokens, and r_i the number of remaining tokens for each grouped sequence i $(1 \leq i \leq k)$. The fitness metric can be defined as follows (Rozinat and van der Aalst, 2008):

$$f = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i m_i}{\sum_{i=1}^{k} n_i c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i r_i}{\sum_{i=1}^{k} n_i p_i}\right).$$

- **behavioral appropriateness** $a_B^{'}$: Behavioral appropriateness is a metric that is obtained by an exploration of the state space of a workflow net and by comparing the different types of *follows* and *precedes* relationships that are allowed in the workflow net with the different types of *follows* and *precedes* relationships that occur in the event log. The metric is defined as the proportion of number of *follows* and *precedes* relationships that the workflow net has in common with the event log vis-à-vis the number of relationships allowed by the workflow net. Let S_F^m be the S_F relation and S_P^m be the S_P relation for the process model, and S_F^l the S_F relation and S_P^l the S_P relation for the event log. The advanced behavioral appropriateness metric $a_B^{'}$ is defined as follows (Rozinat and van der Aalst, 2008):

$$a_B^{'} = \left(\frac{|S_F^l \cap S_F^m|}{2.|S_F^m|} + \frac{|S_P^l \cap S_P^m|}{2.|S_P^m|}\right).$$

Rozinat and van der Aalst (2008) also report a solution to two non-trivial problems that are encountered when replaying Petri nets with silent steps and duplicate activities. In the presence of silent steps (or invisible tasks) it is non-trivial to determine wether there exist a suitable firing sequence of invisible tasks such that the right activities become enabled for the workflow net to optimally replay a given sequence of events. Likewise, in the presence of multiple enabled duplicate activities, it is a non-trivial problem of determining the optimal firing, as the firing of one duplicate activity affects the ability of the workflow net to replay the remaining events in a given sequence of events. Rozinat and van der Aalst (2008) present two local approaches that are based on heuristics involving the next activity event in the given sequence of events.

Fitness f and behavioral appropriateness $a_B^{'}$ are particularly useful measures to evaluate the performance of a discovered process model. Moreover, these metrics have been implemented in the ProM framework. However, the interpretation of the fitness measure requires some attention: although it accounts for recall as it punishes for the number of missing tokens that had to be created, it also punishes for the number of tokens that remain in a workflow net after log replay. The latter can be considered *extra* behavior. Therefore, the fitness metric f also has a specificity semantics attached to it. Furthermore, it is to be noted that the behavioral appropriateness $a_B^{'}$ metric is not guaranteed to account for all non-local behavior in the event log (for instance, a non-local non-free choice construct that is part of a loop will not be detected by the measure). In addition, the $a_B^{'}$ metric requires an analysis of the state space of the process model or a more or less exhaustive simulation of the behavior in the model.

6.1.2 New Recall and Precision Metrics

The availability of an event log supplemented with artificial negative events, allows for the definition of a new specificity metric that does not require a state space analysis. Instead, specificity can be calculated by replaying the (grouped) sequences, supplemented with negative events. We therefore define:

- **behavioral recall** r_B^p: The behavioral recall r_B^p metric is obtained by sequence replay. The values for TP and FN are initially zero. Starting from the initial marking, each sequence is being replayed. Whenever an enabled transitions fires during sequence replay, the value for TP is increased by one. Whenever a transition is not enabled, but must be forced to fire the value for FN is increased. As an optimization, identical sequences only are to be replayed once. Let k represent the number of grouped sequences, n_i the number of process instances, TP_i number of events that are correctly parsed, and FN_i the number events for which a transition was forced to fire for each grouped sequence i $(1 \leq i \leq k)$. At the end of the sequence replay, r_B^p is obtained as follows:

$$r_B^p = \left(\frac{\sum_{i=1}^{k} n_i TP_i}{\sum_{i=1}^{k} n_i TP_i + \sum_{i=1}^{k} n_i FN_i} \right).$$

  In the case of multiple enabled duplicate transitions, sequence replay fires the transition of which the succeeding transition is the next positive event (or makes a random choice). In the case of multiple enabled silent transitions, log replay fires the transition of which the succeeding transition is the next positive event (or makes a random choice). Unlike the fitness metric f, r_B^p does not punish for remaining tokens. Whenever after replaying a sequence tokens remaining tokens cause *additional behavior* by enabling particular transitions, this is punished by our behavioral specificity metric s_B^n.

- **behavioral specificity** s_B^n: The behavioral specificity s_B^n metric can be obtained during the same replay as r_B^p. The values for TN and TP are initially zero. Whenever during replay, a negative event is encountered for which no transitions are enabled, the value for TN is increased by one. In contrast, whenever a negative event is encountered during sequence replay for which there is a corresponding transition enabled in the workflow net, the value for FP is increased by one. As an optimization, identical sequences only are to be replayed once. Let k represent the number of grouped sequences, n_i the number of process instances, TN_i number of negative events for which no transition was enabled, and FP_i the number negative events for which a transition was enabled during the replay of each grouped sequence i $(1 \leq i \leq k)$. At the end of the sequence replay, s_B^n is obtained as follows:

$$s_B^n = \left(\frac{\sum_{i=1}^{k} n_i TN_i}{\sum_{i=1}^{k} n_i TN_i + \sum_{i=1}^{k} n_i FP_i} \right).$$

The metrics make use of heuristics to calculate the appropriate duplicate or silent transition to fire within a Petri net, rather than performing a partial state space analysis or a backtracking procedure to determine the exact transition to fire. The motivation for this choice is that process discovery is to construct a graphical model that must be comprehensible to end users. When an end user replays a process instance in a Petri net, he or she is unlikely to determine the right routing choice by backtracking trough a labyrinth of silent and duplicate transitions.

Because the behavioral specificity metric s_B^n checks whether the workflow net recalls negative event, it is inherently dependent on the way in which these negative events are generated. For the moment, the negative event generation procedure is configurable by the negative event injection probability π, and whether or not it must account for the parallel variants of the given sequences of positive events. For the purpose of uniformity, negative events are generated in the test sets with π equal to 1.0 and account for parallel variants = true. Admittedly, evaluating AGNEs on a metric that uses the same principle as its learning technique introduces a bias in comparing the performance of AGNEs with respect to other learning algorithms. For this reason, we also report the PM, f, and $a_B^{'}$ metrics in our evaluation of AGNEs. A comparable bias, of course, also applies to the genetic miner, which has a metric similar to f in its population fitness function.

The metrics that have been introduced in this section will be used to evaluate the performance of AGNEs in the following sections. They have been defined and implemented for workflow nets. However, they can also be applied to other formal process modeling languages.

6.2 Experimental Evaluation

In this section the results of an experimental evaluation of AGNEs are presented. First we will discuss the properties of the event logs and the parameter settings that have been used. Then, in Section 6.2.1 we describe the results of a number of zero-noise experiments that allow to benchmark the expressiveness of the AGNEs language bias with respect to other learners. In Section 6.2.2, we analyze the results of a number of noise experiments with different types and levels of noise that have been carried out to test how the learning algorithm behaves in the presence of low-frequent additional behavior. Finally in Section 6.2.3, we compare the ability of AGNEs to generalize from incomplete event logs.

In order to evaluate and compare the performance of AGNEs, a benchmark experiment with 34 event logs has been set up. These event logs have previously been used by Alves de Medeiros (2006) and Alves de Medeiros et al. (2007) to evaluate the genetic miner algorithm. Table 6.1 describes the properties of the underlying artificial process models of the event logs. The number of different process instance sequences (column "$\neq$ process inst.") gives an indication of the amount of different behavior that is present in the event log. This number is to be compared with the total number of process instances in the event logs. In general, the presence of loops and parallelism exponentially increases the amount of different behavior that can be produced by a process. Therefore, the number of activity types that are pairwise parallel and the number and type of loops have been reported in Table 6.1. In correspondence with the naming conventions used by Alves de Medeiros, nfc stands for non-free-choice, l1l and l2l stands for the presence of a length-one and length-two loop respectively, and st and unst stands for structured and unstructured loops. Furthermore, the presences of special structures such as skip activities and (parallel or serial) duplicate activities have been indicated. For most of the event logs in the experiment, a reference model was available that can be assumed to represent the behavior in the event log. Columns "r_B^p reference model" and "s_B^n reference model" indicate the behavioral recall and specificity of the reference models with respect to the original event logs. Under the assumption that the reference models are entirely correct and that the event logs completely represent all possible behavior, the r_B^p should be equal to one for every dataset. As can be observed from Table 6.1, this is not the case because not all event logs completely represent all possible behavior in the reference models. Nonetheless, the number of incorrectly derived negative events is almost always less than one percent of the total amount of negative events. Negative events have been generated with an unlimited window size (-1).

In the experiments, the performance of AGNEs is compared to the performance of four state-of-the-art process discovery algorithms: α^+ (Alves de Medeiros et al., 2004; van der Aalst et al., 2004), α^{++} (Wen et al., 2007), genetic miner (Alves de Medeiros et al., 2007) and heuristics miner (Weijters et al., 2006). Admittedly, choosing to include these four algorithms seems arbitrary, and one could wonder why algorithms such as the multi-phase miner (van Dongen and van der Aalst, 2005b), FSM/Petrify (van der Aalst et al., 2006), Fuzzy miner (Günther

Table 6.1: Event log properties

	activity types	**$\neq$ process inst.**	**process inst.**	**r_B^p reference model**	**s_B^n reference model**	**$\|\|$ activity types**	**loops**	**skip**	**nfc**	**duplicate**
a10skip	12	6	300	1.000	1.000	1		1		
a12	14	5	300	1.000	1.000	2				
a5	7	13	300	1.000	1.000	1	1 l1l			
a6nfc	8	3	300	1.000	1.000	1			1	
a7	9	14	300	1.000	1.000	4				
a8	10	4	300	1.000	1.000	1				
al1	9	98	300	1.000	0.996	n.a.	1 unst			
al2	13	92	300	1.000	0.992	n.a.	2 unst			
betaSimplified	13	4	300	1.000	1.000	0		1	1	2
bn1	42	4	300	1.000	1.000	0				
bn2	42	25	300	1.000	1.000	0	1 st			
bn3	42	150	300	1.000	0.999	0	2 st			
choice	12	16	300	1.000	1.000	0				
DriversLicense	9	2	300	1.000	1.000	0				
DriversLincensel	11	87	350	1.000	0.986	1	1 st	1	1	1
herbstFig3p4	12	32	300	1.000	0.999	3	1 st			
herbstFig5p19	8	6	300	1.000	1.000	1				1
herbstFig6p18	7	153	300	1.000	0.977	0	1 l1l, 1 l2l			
herbstFig6p19	5	136	300	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
herbstFig6p31	9	4	300	1.000	1.000	0				1
herbstFig6p33	10	4	300	1.000	1.000	0				1
herbstFig6p36	12	2	300	1.000	1.000	0			1	
herbstFig6p37	16	135	300	1.000	0.996	36				
herbstFig6p38	7	5	300	1.000	1.000	3				1 par.
herbstFig6p39	7	12	300	1.000	1.000	1				
herbstFig6p41	16	12	300	1.000	1.000	4				
herbstFig6p45	8	12	300	1.000	1.000	5				
l1l	6	69	300	1.000	0.988	1	2 l1l			
l1lSkip	6	269	300	1.000	0.732	0	2 l1l			
l2l	6	10	300	1.000	1.000	0	1 l2l			
l2lOptional	6	9	300	1.000	1.000	0	1 l2l			
l2lSkip	6	8	300	1.000	0.999	0	1 l2l			
parallel5	10	109	300	1.000	1.000	10				
repair2	8	48	1000	0.998	0.995	2	1 unst			

and van der Aalst, 2007), or DecMiner (Lamma et al., 2007) have not been considered. A first reason is to keep the size of the benchmarking experiment under control. In its current setup, the experiment requires 496 ($= 34 + 8 \times 34 + 190$) independent runs of each algorithm included in the test. Being the first large-scale, comparative benchmark study in the literature of process discovery, we have chose to include algorithms that already have appeared as journal publications (α^+, α^{++}, and genetic miner) or that are much referenced in the literature (heuristics miner).

During all experiments, the algorithms were run with the same, standard parameter settings of their ProM 4.2 implementation, as reported in Table 6.2. These parameter settings coincide with the ones used to run similar experiments by the authors of the algorithms. The parameter settings of AGNEs have been empirically chosen. Ideally, a procedure should be developed for determining proper parameter values for AGNEs prior to running an experiment. Arguably, choosing a parameter setting for AGNEs that yields good results, and choosing a standard parameter setting for the other learning algorithms induces a bias in the benchmark experiment. However, this effect in minimized by the size of the experiment, including event logs of 34 different process instances. The same parameter settings are used for all datasets. Furthermore, the parameter settings of the genetic miner are also used in experiments that involve a subset of the event logs used in our benchmark study.

To enable a comparison on the same terms, AGNEs was not provided with prior knowledge regarding parallelism or locality of activity types. In particular, the thresholds used to induce frequent temporal patterns have been given the following values ($t_{absence} = 0.9$, $t_{chain} = 0.08$, $t_{succ} = 0.8$, $t_{ordering} = 0.08$, $t_{triple} = 0.10$). In practice, a good threshold depends on the amount of low-frequent behavior (noise) one is willing to accept within the discovered process model. The negative event injection probability π influences the proportion of artificially generated negative events in an event log. A strong imbalance of this proportion may bias a classification learner towards a majority class prediction, without deriving any useful preconditions for a particular activity type. As a rule of thumb, it is a good idea to set this parameter value as low as possible, without the learner making a majority class prediction. In the experiments π has been given a default value of 0.08. Ex-post, AGNEs warns the user when too low a value for π has led to a majority-class prediction. A set of other parameter settings allow to adjust the language bias of AGNEs. To allow for a comparison based on process discovery only, *OccursLessThan* constructs and case data conditions have been left out of the language bias. TILDE's C4.5 gain metric was used as a heuristic for selecting the best branching criterion. In addition, TILDE's C4.5 post pruning method was used with a standard confidence level of 0.25. Furthermore, TILDE is forced to stop node splitting when the number of process instances in a tree node drops below 5. The Petri-net construction algorithm, uses a connect threshold $t_{connect}$ of 0.4, indicating that at least 40% of the triples that can occur by connecting a transition to a place, must be frequent.

The AGNEs technique, has run times in between 20 seconds and 2 hours for the data sets in the experiments on a Pentium 4, 2.4 Ghz machine with 1GB internal memory. These processing times are well in excess of the processing times of α^+, α^{++} and heuristics miner. In comparison to the run times of the genetic miner algorithm, processing times are considerably shorter. Most of the time is required by TILDE to learn the preconditions for each activity type. The generation of negative events also can take up some time. As process discovery generally is not a real-time data mining application, less attention has been given to computation times.

Table 6.2: Parameter settings

Algorithm (Ref.)	**Parameter settings**	
α^+ (Alves de Medeiros et al., 2004)	derive succession from partial order information = true	
	enforce causal dependencies within events of the same activity = false	
	enforce parallelism by overlapping events = false	
α^{++} (Wen et al., 2007)	(no settings)	
heuristics miner Weijters et al. (2006)	relative to best threshold = 0.05	
	positive observations = 10	
	dependency threshold = 0.9	
	length-one-loops threshold = 0.9	
	length-two-loops threshold = 0.9	
	long-distance threshold = 0.9	
	dependency divisor = 1	
	AND threshold = 0.1	
	use all-activities-connected heuristic = true	
	use long-distance dependency heuristic = false	
genetic miner (Alves de Medeiros et al., 2007)	population size = 100	
	max number generations = 1000	
	initial population type = possible duplicates	
	power value = 1	
	elitism rate = 0.2	
	selection type = tournament 5	
	extra behavior punishment with $\kappa = 0.025$	
	enhanced crossover type with crossover probability = 0.8	
	enhanced mutation type with mutation probability = 0.2	
AGNEs	prior knowledge	none
	temporal constraints	$t_{absence} = 0.9$, $t_{chain} = 0.08$, $t_{succ} = 0.8$, $t_{ordering} = 0.08$, $t_{triple} = 0.1$
	negative event generation	injection probability $\pi = 0.08$
		calculate parallel variants = true
		include global sequences = true
	language bias:	include occurrence count = false
		data conditions = none
	TILDE	splitting heuristic: gain
		minimal cases in tree nodes = 5
		C4.5 pruning with confidence level = 0.25
	graph construction	$t_{connect} = 0.4$

6.2.1 Expressiveness

A first experiment has been set up to evaluate the expressiveness of the proposed language bias. In particular, the experiment intends to evaluate the degree to which AGNEs is capable of discovering complete and precise process models from the artificial event logs. To this end, α^+ (Alves de Medeiros et al., 2004; van der Aalst et al., 2004), α^{++} (Wen et al., 2007), genetic miner (Alves de Medeiros et al., 2007), heuristics miner (Weijters et al., 2006) and AGNEs have been run once on the 34 event logs. For the purpose of this experiment, no noise was added to the event logs.

The aggregated results, with averages over all event logs are reported in Table 6.3. The **best** average performance over the 34 event logs is underlined and denoted in bold face for each metric. We then use a paired t-test to test the significance of the performance differences. Performances that are **not significantly different at the 5% level** from the top-ranking performance with respect to a one-tailed paired t-test are tabulated in bold face. Statistically *significant underperformances at the 1% level* are emphasized in italics. Performances significantly different at the 5% level but not at the 1% level are reported in normal font. For the *PM* measure, no paired t-tests could be performed, because the metric could not be calculated on some of the process models discovered by α^+ and α^{++}. The latter is the case when the discovered process models have disconnected elements.

Table 6.3: zero-noise experiment - aggregated results

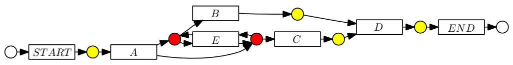
	PM	f	$a_B^{'}$	*acc*	r_B^p	s_B^n	acc_B^{pn}
α^+	0.72	*0.96*	$\underline{\textbf{0.92}}$	$\underline{\textbf{0.94}}$	*0.96*	*0.85*	*0.91*
α^{++}	0.82	**0.98**	**0.87**	**0.92**	**0.98**	**0.93**	**0.96**
AGNEs	0.90	**0.99**	**0.87**	**0.93**	$\underline{\textbf{0.99}}$	$\underline{\textbf{0.96}}$	$\underline{\textbf{0.98}}$
genetic	0.91	$\underline{\textbf{1.00}}$	*0.83*	**0.91**	**0.99**	**0.95**	**0.97**
heuristics	0.88	**0.99**	**0.86**	**0.92**	**0.99**	**0.95**	**0.97**
flower	0.00	1.00	0.23	0.61	1.00	0.00	0.50

To calibrate the metrics, we also report their evaluation of the so-called flower model. Because the flower model parses every possible sequence, it has a perfect recall but zero specificity. These properties are to some extent reflected in the metrics in Table 6.3. The fitness measure f and the behavioral recall measure r_B^p are both 1.0, whereas the behavioral specificity metric s_B^n amounts to 0. The behavioral appropriateness measure $a_B^{'}$ does not really seem to quantify the lack of specificity of the flower model.

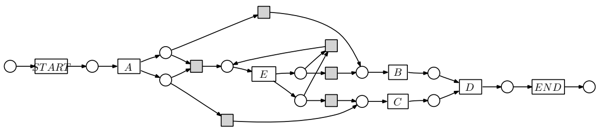
Furthermore, the results show genetic miner to score the highest on fitness f and α^+ to score the highest on behavioral appropriateness $a_B^{'}$. Remarkably, the metrics introduced in this chapter show the inverse picture: AGNEs, genetic miner, and heuristics miner score the highest on the behavioral recall of the positive events r_B^p and the behavioral specificity with regard to negative events s_B^n. Theses small differences between both pairs of metrics can be attributed to a difference in semantics of the metrics. Whereas f punishes for remaining tokens in the discovered Petri net, r_B^p does not. Whereas $a_B^{'}$ compares the succession and precedence relationships between activities in the Petri net with these observed in the log, s_B^n compares the extent to which the Petri net parses negative events.

The metrics r_B^p and s_B^n do not indicate any significant difference for the performance of α^{++}, AGNEs, genetic miner, and heuristics miner. Only by looking at the individual process models, the expressiveness of AGNEs with respect to the detection of non-local, non-free choice constructs or the discovery of duplicate tasks becomes apparent. For example, Figure 5.8, already discussed in Section 5.6, shows how AGNEs is capable of detecting non-local, non-free choice constructs, even within the loop of the DriversLicensel reference problem. Length-one loops can become very complicated, for instance when they occur prior to a parallel split. This is correctly detected for the a5 reference problem, as depicted in Figure 6.1. AGNEs is also particularly suited for the detection of duplicate activities. In the herbstFig6p33 event log, the activity A occurs in three different contexts and AGNEs draws three different, identically labeled transitions correspondingly. Figure 6.2 compares the results of heuristics miner and AGNEs on this event log. Likewise AGNEs was capable of detecting the duplicate activities TravelTrain and TravelCar in the betaSimplified event log, as displayed in Figure 6.4. Figure 6.3 shows that AGNEs is capable of detecting the complex AND/OR split–join and the skip sequence that are characteristic for the a10skip event log.

Not all reference problems, however, were discovered by AGNEs with perfect recall and specificity. A particularly difficult problem is the occurrence of parallel, duplicate tasks. In a one-event setting, AGNEs is incapable of distinguishing a pair of parallel duplicate tasks from a length-one loop. Figure 6.5 is an example of a process model that contains such a difficulty. None of the process discovery algorithms in the experiment were capable of producing a model that summarizes the event log with perfect recall and specificity. Likewise, AGNEs is incapable of discovering a perfect model for the a6nfc reference problem, whereas α^{++} does. This is displayed in Figure 6.6.



(a) AGNEs result



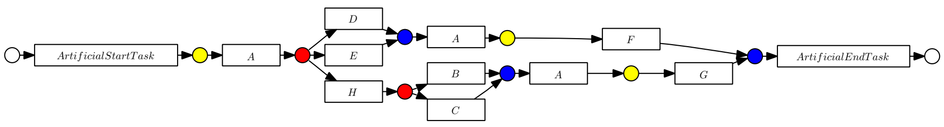
(b) heuristics miner result

Figure 6.1: a5 – AGNEs detects the complex length-one loop E.

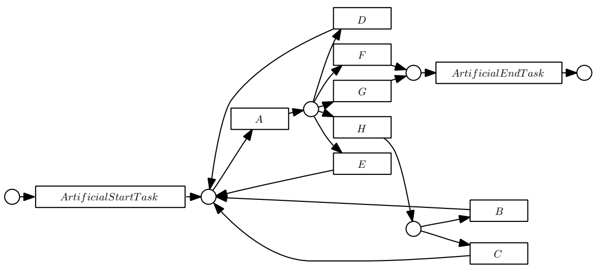
The outcomes of genetic miner also require some special attention. Genetic miner is a non-deterministic mining algorithm. From the literature, it is known that genetic miner can discover process models with perfect specificity and recall for almost each of the event logs included in our experiment. However, due to the longer run times of the algorithm only a single run (with random seed 1) was performed in our experiments. This is justifiable, as we only report average performances for genetic miner over the 34 datasets.

6.2.2 Robustness to Noise

In a second experiment, we have stirred up the 34 event logs with artificial noise. In the literature, six artificial noise types have been described (Alves de Medeiros et al., 2007; Maruster, 2003): (1) *missing head*: the removal of the head of a sequence, (2) *missing body*: the removal of the mid section of a sequence, (3) *missing tail*: the removal of the end section of a sequence, (4) *swap tasks*: the

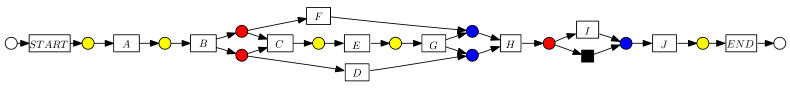


(a) AGNEs result

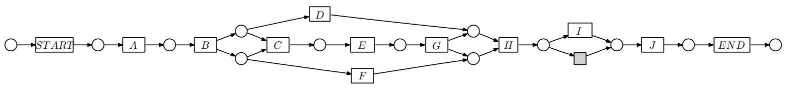


(b) heuristics miner result

Figure 6.2: herbstFig6p33 – AGNEs detects the duplicate activity A.

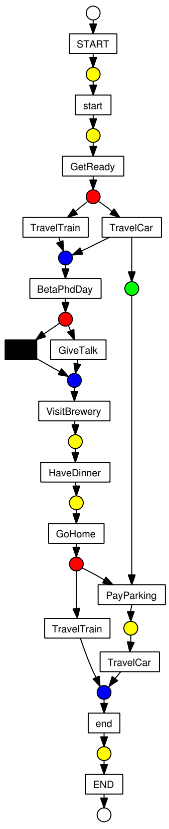


(a) AGNEs result

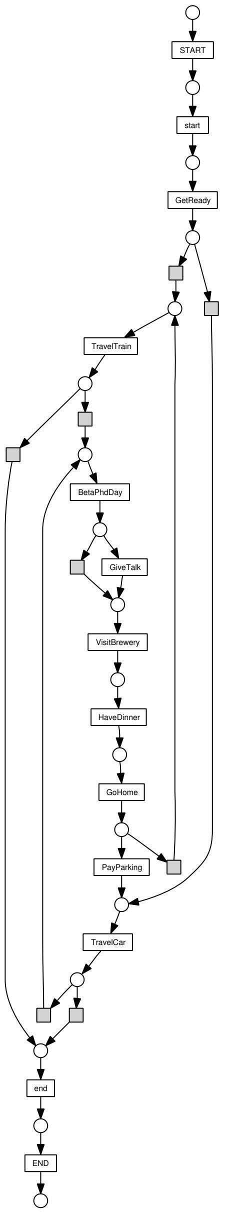


(b) heuristics miner result

Figure 6.3: a10skip – AGNEs detects the complex or-splits.

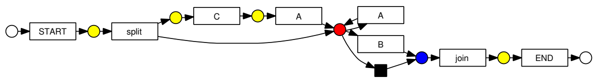


(a) AGNEs result

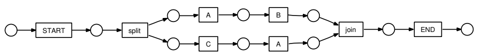


(b) heuristics miner result

Figure 6.4: betaSimplified – AGNEs detects the duplicate activities TravelTrain and TravelCar.

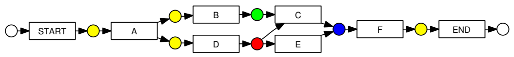


(a) AGNEs result

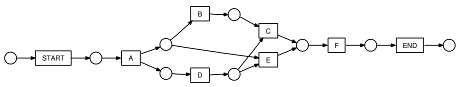


(b) the underlying process

Figure 6.5: herbstFig6p38 – AGNEs does not correctly detect the parallel, duplicate activity A.



(a) AGNEs result



(b) the α^{++} mining result

Figure 6.6: a6nfc – Unlike α^{++}, AGNEs does not correctly detect the complex non-local, non-free choice construct.

interchange of two random events in a sequence, (5) *remove task*: the removal of a random event in a sequence, and (6) *mix all*: a combination of all of the above. The noise has been added with the AddNoiseLogFilter event log filter available in the ProM framework. This filter has been applied after ungrouping the 34 event logs. To keep the size of the experiment under control, we have limited the noise types used in our experiments to *mix all* and *swap tasks*. For both noise types, the used noise levels of 5%, 10%, 20% and 50% are applied.

Table 6.4 reports the average results of the discovered process models over all 34 zero noise event logs. As is known from the literature, heuristics miner, and genetic miner are resilient to noise, whereas the formal approaches of α^{+} and α^{++} are known to overfit the noise in event logs. On 11 event logs from the bn1, bn2, and bn3 processes, the α^{++} implementation was incapable of producing an outcome. These missing values resulted in a score of 0 for each measure. Furthermore, the state space analysis required to calculate the behavioral appropriateness measure $a_B^{'}$ produces invalid outcomes that occur 27 times for the results of the genetic miner algorithm out of a total of 272 (34 x 8) experiments, the reported results for the genetic miner are less suitable for comparison. For this reason, we only indicate the significance of the differences between the r_B^p and s_B^n measures in the previous section. For the PM, f, and $a_B^{'}$ metrics, the algorithm that has obtained the <u>best</u> average score, is underlined. For every noise level, AGNEs obtains accuracy results that are robust and not significantly different from the results obtained by heuristics miner. This is a remarkable result, as AGNEs is a more expressive algorithm than heuristics miner, also capable of detecting more complex structures such as non-local dependencies and duplicate activities.

Table 6.4: noise experiments - aggregated results

		PM	f	$a_B^{'}$	acc	r_B^p	s_B^n	acc_B^{pn}
mix_all0.05	α^+	0.11	0.83	0.85	0.84	*0.87*	*0.62*	*0.74*
	α^{++}	0.00	0.79	0.65	0.72	*0.75*	*0.63*	*0.69*
	AGNEs	0.89	0.99	0.87	0.93	**0.99**	**0.95**	**0.97**
	genetic	0.74	0.99	0.63	0.81	**0.98**	**0.91**	**0.95**
	heuristics	0.88	0.98	0.87	0.93	**0.98**	**0.94**	**0.96**
mix_all0.1	α^+	0.08	0.80	0.84	0.82	*0.84*	*0.59*	*0.72*
	α^{++}	0.00	0.73	0.80	0.76	*0.64*	*0.64*	*0.64*
	AGNEs	0.83	0.99	0.89	0.94	**0.99**	**0.96**	**0.97**
	genetic	0.51	0.97	0.59	0.78	*0.94*	*0.78*	*0.86*
	heuristics	0.88	0.99	0.86	0.92	**0.99**	**0.95**	**0.97**
mix_all0.2	α^+	0.00	0.77	0.91	0.84	*0.82*	*0.51*	*0.67*
	α^{++}	0.00	0.65	0.65	0.65	*0.49*	*0.63*	*0.55*
	AGNEs	0.79	0.97	0.87	0.92	**0.97**	**0.94**	**0.96**
	genetic	0.47	0.96	0.53	0.74	*0.93*	*0.73*	*0.83*
	heuristics	0.86	0.98	0.85	0.92	**0.98**	**0.94**	**0.96**
mix_all0.5	α^+	0.00	0.63	0.75	0.69	*0.67*	*0.46*	*0.56*
	α^{++}	0.00	0.51	0.61	0.58	*0.26*	*0.70*	*0.48*
	AGNEs	0.54	0.96	0.77	0.87	**0.97**	**0.90**	**0.93**
	genetic	0.20	0.95	0.43	0.69	*0.86*	*0.53*	*0.69*
	heuristics	0.66	0.97	0.74	0.85	**0.96**	**0.88**	**0.92**
swap_tasks0.05	α^+	0.00	0.65	0.85	0.75	*0.76*	*0.45*	*0.60*
	α^{++}	0.00	0.59	0.67	0.63	*0.52*	*0.61*	*0.56*
	AGNEs	0.90	0.99	0.87	0.93	**0.99**	**0.96**	**0.97**
	genetic	0.44	0.95	0.61	0.78	*0.90*	*0.74*	*0.82*
	heuristics	0.88	0.99	0.85	0.92	**0.99**	**0.95**	**0.97**
swap_tasks0.1	α^+	0.00	0.58	0.86	0.72	*0.69*	*0.48*	*0.58*
	α^{++}	0.00	0.53	0.66	0.59	*0.38*	*0.61*	*0.49*
	AGNEs	0.78	0.98	0.87	0.93	**0.98**	**0.94**	**0.96**
	genetic	0.38	0.94	0.53	0.74	*0.89*	*0.65*	*0.77*
	heuristics	0.80	0.97	0.86	0.92	**0.98**	**0.94**	**0.96**
swap_tasks0.2	α^+	0.00	0.54	0.77	0.66	*0.59*	*0.52*	*0.55*
	α^{++}	0.00	0.45	0.65	0.55	*0.27*	*0.67*	*0.47*
	AGNEs	0.73	0.97	0.86	0.92	**0.98**	**0.93**	**0.95**
	genetic	0.19	0.93	0.62	0.77	*0.84*	*0.52*	*0.68*
	heuristics	0.69	0.96	0.87	0.92	**0.96**	**0.88**	**0.92**
swap_tasks0.5	α^+	0.00	0.41	0.61	0.51	*0.40*	*0.63*	*0.51*
	α^{++}	0.00	0.36	0.61	0.48	*0.16*	**0.77**	*0.46*
	AGNEs	0.32	0.91	0.72	0.81	**0.95**	**0.82**	**0.89**
	genetic	0.07	0.93	0.77	0.85	*0.79*	*0.40*	*0.59*
	heuristics	0.45	0.94	0.66	0.80	**0.94**	**0.83**	**0.89**

The reasons why AGNEs is robust to noise can be put down to the following. First of all, the generation of negative events is not invalidated by the presence of noise. Noise is *additional* low-frequent behavior that will result in less negative events being generated by AGNEs. However, the presence of noisy positive events does not lead to the generation of noisy negative ones. Another property that adds to robustness, is that the constraints in AGNEs ' language bias allows it to come up with so-to-say structured patterns and to some extent prevents the construction of arbitrary connections between transitions, while remaining expressiveness with regard to short loops, duplicate tasks and non-local behavior. Finally, the formulation of process discovery as a classification allows for the application of an already robust classification algorithm (TILDE). Like many classification learners, TILDE takes into account the frequency of an anomaly, when constructing the preconditions for each activity type. Moreover, TILDE applies the same tree-level pruning method as C4.5 (Quinlan, 1993).

6.2.3 Ability to Generalize

As reported by (van der Aalst et al., 2004), process discovery algorithms make a completeness assumption when learning from event logs containing positive events only. AGNEs explicitly makes this completeness assumption by artificially generating negative events from the observed behavior in the event log. In particular, it assumes that any sub-sequence of an indicated window size that is not in the event log (or that is not a parallel variant of a sub-sequence in the event log), should not be part of the process model to be learned.

Unfortunately, event logs rarely exhibit complete behavior. This is a fortiori the case when the underlying business process contains a lot of parallelism and loops. Although AGNEs takes into account the parallel variants of each observed sub-sequence, AGNEs can still generate a number of incorrect negative events from an incomplete event log. As goes for other process discovery algorithms, incomplete event logs can invalidate the ability to produce the correct underlying process model. The ability to generalize towards unseen behavior, while maintaining an acceptable level of specificity can be considered to be generalization.

To evaluate AGNEs' ability to generalize, a 10-fold cross-validation experiment has been set up. In the literature on process discovery, cross-validation has only been considered by Goedertier et al. (2007b); Rozinat et al. (2007). The reason for the absence of cross-validation experiments, is that process discovery is an inherently *descriptive* learning task rather than a *predictive* one. The primary intent of process discovery is to produce a model that accurately describes the event log at hand. Nonetheless, it is interesting to test the *predictive* ability of process discovery algorithms in an experimental setting. To apply cross-validation, a randomization routine has been written in SWI-Prolog that groups similar sequences, randomly partitions the grouped event log in $n = 10$ uniform subgroups, and produces n pairs of training and test event logs. Training event logs are used for the purpose of process discovery. Test event logs are used for evaluation, this is for calculating the specificity and recall metrics.

Precision metrics must be calculated based on the combination of training and test event logs, the entire event log. Although this might seem unintuitive, precision and specificity metrics make a completeness assumption as well, as they account for the amount of *extra* behavior in a process model vis-à-vis the event log

(Rozinat and van der Aalst, 2008). To correctly evaluate the proposed learning technique, it is important that the negative events in the test set accurately indicate the state transitions that are not present in the event log. For this reason, the negative events in the test log are created with information from the entire event log. Should the negative event generation be based on training set instances only, it is possible that additional, erroneous negative events are injected because it is possible that some behavior is not present in the test set. In short, the experiment applies the above-described partitioning, after having generated negative events for each grouped process instance. Intended to be used by the evaluation metric, the negative events have been generated with an injection probability π equal to 1, an infinite window size, and by considering parallel variants. Evidently, the thus generated negative events were not retained in the training set. For training purposes, negative events have been calculated based on the information in the training set only. For the same reasons, the behavioral appropriateness metric $a_B^{'}$ has also been calculated based on the whole of training and test set data.

In the experiment, only 19 out of the 34 event logs were retained, as the other event logs have less than 10 different sequences. Table 6.5 shows the aggregated, average results of the 10-fold cross validation experiment over 190 event logs. As explained in Section 6.2.1 the font of the metrics indicates the statistical significance of their difference with the top-ranking performance. From the results for the parsing measure *PM*, the fitness measure f, and behavioral recall measures r_B^p, it can be concluded that genetic miner scores slightly better on the recall requirement. Moreover, the behavioral specificity metric s_B^n shows genetic miner and heuristics miner to produce slightly more specific models.

Table 6.5: 10-fold cross validation experiment - aggregated results

		PM	f	$a_B^{'}$	*acc*	r_B^p	s_B^n	acc_B^{pn}
zero_noise	α^+	0.72	*0.96*	$\underline{\mathbf{0.96}}$	$\underline{\mathbf{0.96}}$	*0.97*	*0.83*	*0.90*
	α^{++}	0.77	*0.97*	*0.81*	*0.88*	0.97	*0.90*	0.93
	AGNEs	0.80	0.98	*0.81*	*0.89*	**0.98**	*0.91*	0.94
	genetic	0.83	$\underline{\mathbf{0.99}}$	*0.84*	*0.91*	$\underline{\mathbf{0.98}}$	**0.93**	$\underline{\mathbf{0.95}}$
	heuristics	0.79	*0.97*	*0.85*	*0.91*	0.97	$\underline{\mathbf{0.93}}$	**0.95**

From the cross-validation experiment, we conclude that AGNEs portrays similar generalization behavior to other process discovery algorithms. The reason that it is not sensitive to incomplete event logs can be attributed to the following. Given an incomplete event log, AGNEs is likely to generate a proportion of incorrect negative events. However, this proportion of negative events is relatively small, as the negative event injection parameter π is not required to be excessively large. More importantly, the coarse-grained language bias that combines *NS* constructs into larger structures, prevents TILDE from overfitting the incomplete event log and allows it to generalize, to some extent, beyond the observed behavior. The additional incorporation of process knowledge expressed by a domain expert would only add to this benefit. Finally, the negative event injection procedure takes into account parallelism and window size. Concurrent and recurrent behavior are the root causes of incomplete event logs. The ability to include information about parallel variants and window size, gives our learning technique a configurable inductive bias, with different strategies to accounts for incompleteness.

6.3 A Case Study

This section shows the result of the AGNEs process discovery algorithm applied to an event log of customer-initiated processes, recorded by a European telecom provider [1]. The goal of the case study was to investigate whether process discovery can be usefully applied to map the routing choices that are made between queues of a workflow management system (WfMS). With 18721 process instances and 127 queues, the obtained log file has a size of over 130 megabytes in the form of a comma-separated text file. The case study gives an idea of the scalability of the algorithm towards large event logs and the usefulness of the AGNEs process discovery algorithm on realistic, real-life processes.

6.3.1 The Obtained Event Log

The event log consists of events about customer-initiated processes that are handled at three different locations by the employees of the telecom provider. The handling of cases is organized in a first line and a second line. First-line operators are junior operators that deal with frequent customer requests for which standardized procedures have been put in place. When a first-line operator cannot process a case, it is routed to a queue of the second line. Second-line case handling is operated by senior experts who have the authority to make decisions to solve the more involved cases. The second-line processes are coordinated and supported by means of a workflow management system (WfMS). The obtained event log consists of these second-line case handling events. The second-line WfMS is organized as a system of 127 logical queues. Each queue corresponds to a number of similar types of activity that are to be carried out. At any given moment each active case resides in exactly one queue. Employees can process a case by taking it out of the queue into their personal work bin. Every evolution of a case is documented by adding notes. Moreover, employees can classify the nature of the case according to a number of data fields. In addition, a worker or dispatcher has the ability to reroute cases to different queues whenever this is necessary. The system imposes no restrictions with regard to the routing of cases. Queues represent a work distribution system and are akin to roles in WfMS. For the purpose of this analysis, queues are considered to be activity types. To handle cases that have a common cause, a parent-child structure exists among cases. Occasionally,

[1]Due to a non-disclosure agreement, the identity of the telecom provider and the content of the event log cannot be made public. Consequently, all reported queue names have been made anonymous, and all reported queuing time metrics have been randomized.

rules can perform updates on cases, such as automatically closing a case, and automatically rerouting a case.

Table 6.6 summarizes the properties of the original event log and of the event log that was obtained after preprocessing this event log. Because the goal of the case study is to map the routing choices, only *dispatch* event types were retained from the event log. In this context, queues can be seen as activity types. Canfora et al. (2005) and Mendling et al. (2007) assert that a high number of activity types and arcs quickly render a process model incomprehensible. Therefore, the 40 most frequently occurring queues were retained for further analysis. Nine process instances that did not involve at least one of these 40 queues were retained from the event log.

Table 6.6: telecom – the properties of the original and preprocessed event log

Property	Definition
real-life process	Second-line customer-initiated processes regarding the resolution of internet, television, and telephony service problems.
event log	The event log recorded by a WfMS, containing all *dispatch* events of 17812 cases that were closed between January 6, 2008 and February 6, 2008.
activity types	127 different queues of which processes involving the 40 most frequent queues were retained. In addition, the *create* and *close case* event types are considered to be activity types.
event types	From the original four event types (*create*, *notes*, *dispatch*, and *case close*) only the *dispatch* event type was retained.
process inst.	The *case id* field allowed to identify 17821 process instances of which 17812 remained after filtering out processes that did not involve the retained activity types.
case data	the following case data fields were given a value for more than 20% of the cases: *problem code*, *problem cause*, and *service type*.
$\neq$ process inst.	After grouping according to the same follows relationship 1375 different process instances were retained.

First, an exploratory study was made using descriptive statistics. A number of plain vanilla SQL queries on the event log were used to calculate the average and standard deviation of its queue times, the total number of cases that were routed to this queue, and the inflow and outflow to other queues. This data is visualized by means of graphs such as the one in Figure 6.7. Each rectangle represents a queue. Each arc represents cases being routed from one queue to another. For each queue, the average cycle time, standard deviation of the cycle time, and the total number of cases is indicated. On each arc, the outflow of cases from one queue to another is indicated in absolute quantities. This visualization makes it immediately apparent that cases in a given queue can be routed to a great number of queues. To reduce the information overload on the graph, arcs with less than 30 cases were left out. Disconnected queues are left out as well. A lean sigma black belt expert, who is currently making improvements to the second-line case handling, confirmed that the graph was a useful visualization of the queue routing choices that were currently made. Moreover, he asserted that the cross-table representation of the same data, allowing the expert to drill down to individual cases whenever required, was a useful tool in understanding the complexity of the underlying processes.

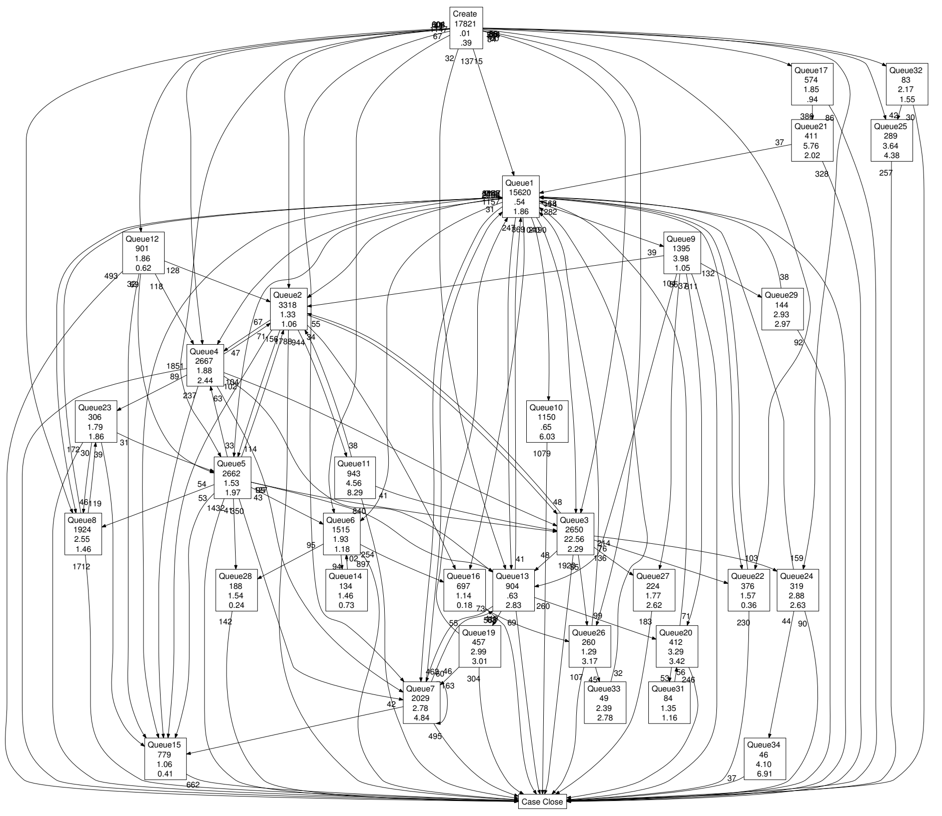


Figure 6.7: telecom – a visualization of queue routing using descriptive statistics only

6.3.2 Validity of Assumptions

Although many process discovery algorithms have been developed, only a few authors report on the practical application of process discovery algorithms on real-life event logs (Alves de Medeiros et al., 2007; Ferreira et al., 2007; Günther and van der Aalst, 2007; van der Aalst et al., 2007a). Practical applications are important, because they give an indication of the validity of the assumptions that are made by process discovery algorithms. In particular, authors like Günther and van der Aalst (2007) and Ferreira et al. (2007) have identified that process discovery algorithms make a number of assumptions that are not always guaranteed for realistic event logs. In the next paragraphs, the validity of these assumptions with respect to the obtained event log is evaluated.

assumption 1. "There is a one-to-one mapping between a system event and a business event." This assumption underestimates the importance of preprocessing an event log. Günther and van der Aalst (2007) assert that system logs may contain events of different levels of abstraction. Moreover, the authors point out that in many processes it is not meaningful to separate the control flow, data flow, and resource perspectives. System logs can indeed have an ontology that is different from the process modeling ontology that is used in this text. Moreover, the interpretation of a system event can differ in function of the intended analysis. Another issue is that the most useful information to classify a system event may reside in unstructured data, such as free text fields or e-mails. The labeling of unstructured data for the purpose of activity recognition can be believed to be very difficult. For these reasons, it is often not possible to automatically convert a system event into a relevant business event. In the obtained event log, the resource and control flow perspective are intertwined: the queue to which a particular case is being dispatched (resource perspective) is considered to be the activity type (control flow perspective). Undoubtedly, the textual annotations that employees add to each case conceal a lot of information about the nature of the underlying activity, but we did not attempt to use this text field to discover or recognize more fine-grained activity types.

assumption 2. "It is possible to identify meaningful process instances in an event log." This assumption entails that each system event can be related to a meaningful process instance by means of a suitable identifier, such as a customer id, an employee number, or a case id. Ferreira et al. (2007) argue that for many system logs the existence of such an identifier is not guaranteed. Consequently, a lot of preprocessing can be required to identify clusters of events that represent process instances. In the obtained event log, process instances are identified using the *case id* identifier of the WfMS.

assumption 3. "The events in the log are generated by exactly one underlying process." Even when meaningful process instances can be identified from an event log, it is possible that the event log represents the history of multiple underlying, real-life processes. Ferreira et al. (2007) demonstrate that the use of sequence clustering algorithms can be useful to detect clusters of frequently occurring sequences. As the obtained event log contains cases that deal with a variety of customer problems – there are over one hundred different *problem codes* – it can be assumed that the event log is actually generated by many underlying processes. Unfortunately, due to poor data quality of case data fields, we did not find a useful filtering criterion. Sequence clustering can be a useful technique to identify clusters of underlying processes (Ferreira et al., 2007). However, due to the high number of underlying processes, the application of expectation maximization algorithm for mixtures of first-order Markov models (Cadez et al., 2003) did not lead to the identification of meaningful clusters of process instances.

assumption 4. "The processes take place in a structured fashion." Structured processes take into account concerns such as prerequisites, synchronization, parallelism, and exclusiveness. Consequently, it is assumed that it is possible to summarize the underlying processes in structured process modeling languages such as workflow nets. Günther and van der Aalst (2007) report on a number of real-life event logs, for which these assumptions were not valid. The authors promote a series of multi-perspective metrics and adaptive simplification and visualization techniques to bring structure to these seemingly unstructured event logs. From Figure 6.7 it can be observed that there are few dominant queue routing choices. The underlying processes apparently have little structure. The latter can be attributed to the fact that the high number of unidentifiable, underlying processes each observe a different routing and by themselves are not guaranteed to occur in a structured fashion. Another reason why the routing occurs in an unstructured fashion is that the WfMS does not impose any restriction on the routing of cases.

The obtained event log to some extent differs from ideal event logs. It does not really record which activities take place, but rather records the different queues each case is routed to. Therefore, some of the assumptions that process discovery algorithms make are not valid for the event log. In particular, assumptions 3 and 4 are clearly problematic for the obtained event log. Another difference is that the underlying processes contain no parallel behavior. Detecting concurrency is one of the more difficult aspects of process discovery. It is interesting to see how deterministic process discovery algorithms cope with these differences.

6.3.3 Results

In this section, we compare the mining results of AGNEs, genetic miner, and heuristics miner. Table 6.7 indicates the parameter settings. Parameter settings that are different from the experimental evaluation in the previous section are indicated in bold face. In particular, AGNEs was provided with the prior knowledge that no activity can occur concurrently: $\forall a, b \in A : \mathit{PriorSerial}(a, b)$. This prior knowledge is justifiable, as no case can be routed to or reside in several queues at the same time. Moreover, the $t_{connect}$ parameter was lowered to a value of 0 (the $\mathit{ChainSeq}(a, b, c)$ predicate being sensitive to the frequency of occurrence of its composing activity types). Genetic miner has been running for 5000 generations, with a population size of 10. These parameter settings correspond to the parameter settings in the case study described by Alves de Medeiros et al. (2007). To account for the prior knowledge that no concurrent behavior is contained in the event log, heuristics miner needs to have a sufficiently large AND threshold, here set to a value of of 500,000.0.

Table 6.7: telecom – parameter settings

Algorithm (Ref.)	**Parameter settings**	
heuristics miner Weijters et al. (2006)	relative to best threshold = 0.05	
	positive observations = 10	
	dependency threshold = 0.9	
	length-one-loops threshold = 0.9	
	length-two-loops threshold = 0.9	
	long-distance threshold = 0.9	
	dependency divisor = 1	
	AND threshold = **500,000.0**	
	use all-activities-connected heuristic = true	
	use long-distance dependency heuristic = false	
genetic miner (Alves de Medeiros et al., 2007)	population size = **10**	
	max number generations = **5000**	
	initial population type = possible duplicates	
	power value = 1	
	elitism rate = 0.2	
	selection type = tournament 5	
	extra behavior punishment with $\kappa = 0.025$	
	enhanced crossover type with crossover probability = 0.8	
	enhanced mutation type with mutation probability = 0.2	
AGNEs	prior knowledge	$\forall a, b \in A : \mathit{PriorSerial}(a, b)$
	temporal constraints	$t_{absence} = 0.9$, $t_{chain} = 0.08$, $t_{succ} = 0.8$, $t_{ordering} = 0.08$, $t_{triple} = 0.1$
	negative event generation	injection probability $\pi = \mathbf{0.04}$
		calculate parallel variants = true
	language bias:	include skip sequences = **false**
		include global sequences = **false**
		include occurrence count = false
		data conditions = none
	TILDE	splitting heuristic: gain
		minimal cases in tree nodes = 5
		C4.5 pruning with confidence level = 0.25
	graph construction	$t_{connect} = \mathbf{0.0}$

The results of applying these process discovery algorithms on the filtered event log are displayed in Table 6.8. To calibrate the metrics, we also report their evaluation of the so-called flower model. Because the flower model represents random behavior, it has a perfect recall of the all behavior in the event log but it also has much *additional* behavior compared to the event log. Because of the latter fact, the flower model has zero specificity. These properties are to some extent reflected in the metrics in Table 6.8. The fitness measure f and the behavioral recall measure r_B^p are both 1.0, whereas the behavioral specificity metric s_B^n amounts to 0. The parsing measure PM does not reflect the recall of the flower model, most likely because the implementation requires a conversion into a heuristics net, which is very complex for a flower model. The behavioral appropriateness measure $a_B^{'}$ does not really seem to quantify the lack of specificity of the flower model.

Table 6.8 also shows the results of the visualization of queue routing of Figure 6.7 that was obtained from descriptive statistics. To calculate these results, the visualization has been converted into a Petri net in the following way. Every

Table 6.8: telecom – mining results

	PM	f	$a_B^{'}$	acc	r_B^p	s_B^n	acc_B^{pn}	#nodes	#arcs	run time
AGNEs	0.06	0.94	0.67	0.80	0.93	0.67	0.80	129	153	0d:10h:06m:34s
descriptive statistics	0.73	0.86	x.xx	x.xx	0.94	0.80	0.87	42	139	<1m
flower model	0.00	1.00	0.76	0.88	1.00	0.00	0.50	47	88	00s:032ms
genetic	0.03	x.xx	x.xx	x.xx	0.83	0.62	0.73	42	14392	7d:00h:08m:26s
heuristics	<u>0.80</u>	<u>0.97</u>	<u>0.72</u>	<u>0.85</u>	<u>0.96</u>	<u>0.88</u>	<u>0.92</u>	42	132	01s:578ms

rectangle (queue) is mapped to a labeled transition with exactly one input and output place. Each arc (queue routing) is mapped into an silent transition that connects the output place of the from-queue to the input place of the to-queue. This mapping is justifiable, because the event log contains no concurrent behavior. If the underlying processes do portray concurrent behavior, there would be no straightforward mapping of descriptive statistics into Petri nets.

The mining results of AGNEs, genetic miner, and heuristics miner are depicted in Figures 6.8, 6.9 and 6.10 respectively. In the next paragraphs, we address the question whether the discovered models produce useful information for an expert to get a good idea of the queue routing choices that are made within the second-line handling of customer problems. In particular, it is determined whether, the discovered models provide additional information compared to the graph with descriptive statistics displayed in Figure 6.7. In Section 4.4.2, three general requirements have been introduced that make a process mining model acceptable: accuracy, comprehensibility, and justifiability. In the next paragraphs, these requirements are applied to the discovered process models.

Accuracy refers to the extent to which the induced model fits the behavior in the event log and can be generalized towards unseen behavior. Accuracy refers both to recall (allowing all observed behavior in the event log) and specificity (disallowing all unobserved or disallowed behavior). Compared to the flower model, AGNEs has a fairly good recall with a fitness f of 0.94 and a behavioral recall r_B^p of 0.93. In addition, the AGNEs result represents a specificity improvement: the behavioral specificity metric s_B^n amounts to 0.67, indicating that 67% of all disallowed behavior is also disallowed by the model. For the genetic miner mining result, the ProM 4.2 implementations of f, and $a_B^{'}$ did not produce an outcome. To calculate these metrics, a conversion of the heuristics nets into Petri nets is required. The resulting Petri nets, which have many invisible transitions, are seemingly too complex to determine a good firing sequence or to perform a state space analysis (Rozinat and van der Aalst, 2008). Because we use simple heuristics to determine a good firing sequence of invisible and duplicate transitions, we are capable of calculating the r_B^p and s_B^n recall and specificity metrics. Notice that much of the complexity of calculating our specificity metric is transferred to generating the artificial negative events. Looking at the available r_B^p and s_B^n measures, it can be observed that the genetic miner result is less good than the AGNEs mining result, both in terms of recall and specificity. Accuracy can be seen as a weighted average of precision and recall. If we attach equal importance

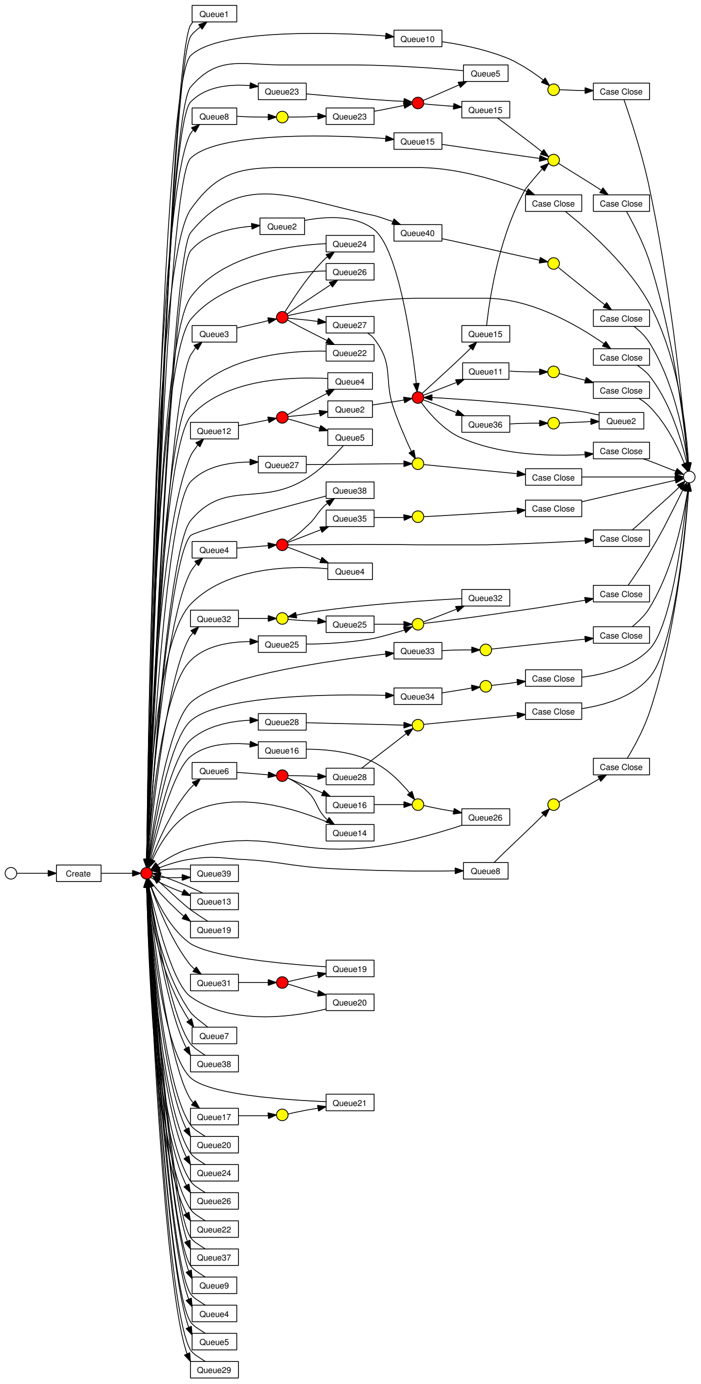


Figure 6.8: telecom – AGNEs mining results as a Petri net

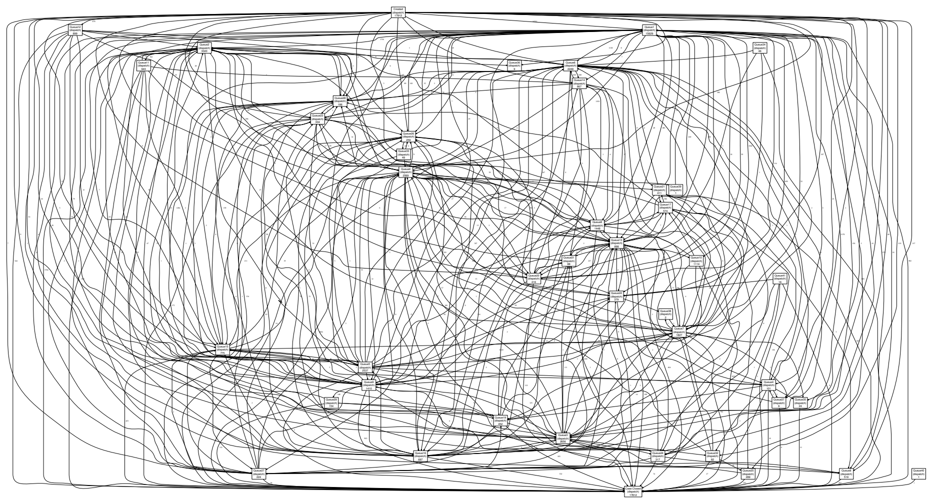


Figure 6.9: telecom – genetic miner mining results as a heuristics net

to precision and recall, the discovered process model by AGNEs has an accuracy of 80%, whereas the models discovered by genetic miner and heuristics miner have an accuracy of 73% and 92% respectively. Heuristics miner is the only algorithm that outperforms the overall accuracy of 87% of the model obtained from the descriptive statistics.

Comprehensibility refers to the extent to which an induced model is comprehensible to end-users. Whereas comprehensibility is more difficult to quantify, it is an important evaluation criterium for process models. Table 6.8 indicates the number of arcs and nodes in each of the mining models. These numbers should be treated with care, because the number of nodes is dependent on the used process modeling languages. For AGNEs the number of places, transitions, and arcs in the Petri net are reported, whereas the genetic miner and heuristics miner this is done with the number of nodes and arcs in the discovered heuristics nets. Mendling et al. (2007) have evaluated graph-based business process model understandability and quality by means of surveys and experiments. In general, the understandability of a process model deteriorates with the number of arcs and nodes in the model. Although the AGNEs mining result contains several duplicate, homonymic transitions, it has less arcs than the genetic miner result. However, in terms of comprehensibility, the simple visualizations in Figure 6.7, obtained by descriptive statistics, and Figure 6.10, obtained by heuristics miner, already give a clear idea of the queue routing frequencies, and queuing times. By allowing the user to filter out the arcs and nodes of which the frequency is below a particular threshold, the end-user obtains a comprehensible and configurable image of the routing choices. Notice that the comprehensibility of the

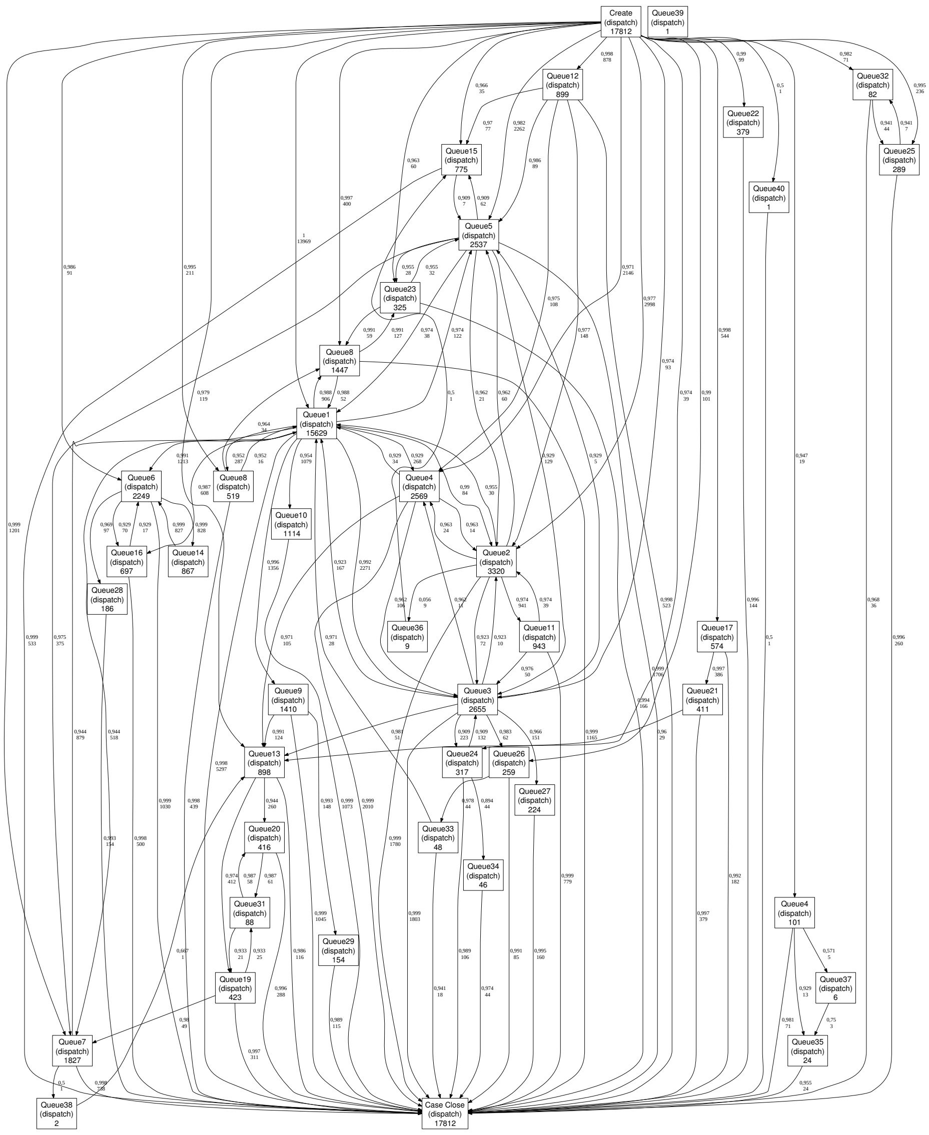


Figure 6.10: telecom – heuristics miner mining results as a heuristics net

descriptive statistics model is mainly due to the lack of concurrent behavior. The visualization does not allow to comprehensively represent concurrency.

Justifiability refers to the extent to which an induced model is aligned with the existing domain knowledge. Domain knowledge can refer to knowledge about concurrency (parallelism), impossibility (mutually-exclusive activities), etcetera. For the obtained event log, a domain expert might be surprised to see a process model in which a routing to several queues is allowed at the same time. Because AGNEs was provided with the requirement that all queue routings must occur in serial, the discovered model does not contain any parallel transitions. The same a-priori knowledge can be conveyed to heuristics miner by setting the AND threshold to a sufficiently large value. In general AGNEs allows to provide parallelism information for individual pairs of activity types. This fine-grained a-priori knowledge cannot be provided by fine-tuning the AND threshold with heuristics miner. Currently, it is not possible to constrain the search space of genetic miner with this a-priori knowledge.

Run time refers to the time that is required to run a particular process discovery algorithm. Table 6.8 indicates considerable differences in required run time on the filtered event log. Whereas a single run of heuristics miner takes less than two seconds, running AGNEs and genetic miner takes over ten hours and over seven days respectively. 85% of the run time required by AGNEs was used for running TILDE to discover activity preconditions. In general, process discovery is not a real-time data mining application. However, an end-user is likely to prefer process discovery algorithms that are less stringent on computational requirements. In practice, shorter run times will encourage the end-user to interactively explore the event log, applying various filters, and parameter settings (Hammori et al., 2006; van Dongen et al., 2005). Reporting run times is perhaps unfair. The implementations of AGNEs, heuristics miner, and genetic miner can be considered to be scientific prototypes, built as a proof-of-concept, not as a commercial process mining algorithm. A commercial version of genetic miner, for instance, contains a much faster implementation of genetic miner (Pallas Athena, 2008b). The descriptive statistics from which model 6.7 has been constructed, were implemented as a number of SQL queries. By constructing a number of indexes on the event log, run time could be reduced to less than 30 seconds.

Case studies can give an idea of the validity of the assumptions and the scalability of the algorithms in the domain of process discovery. Like other case studies (Ferreira et al., 2007; Günther and van der Aalst, 2007; van der Aalst et al., 2007a), this case study brings forward that human-centric processes can take place in an unstructured fashion. Bringing order in the chaos of unstructured processes probably requires a different search strategy and process modeling language than the ones used by existing process discovery algorithms. In spite of that, the AGNEs process discovery technique copes reasonably well with the lack of structure in the event log, but is outperformed in terms of accuracy and comprehensibility both by the heuristics miner and the descriptive statistics model. The case study also indicates that, despite the long run time, our technique is scalable towards real-life sized event logs. Another outcome of the case study, is that the proposed measures for behavioral recall r_B^p and behavioral specificity s_B^n in practice turn out to be valuable metrics for assessing the accuracy of a discovered process model.

6.4 Conclusion

Process discovery algorithms must – in addition to finding the right balance between recall and specificity – deal with challenges such as expressiveness, noise, incomplete event logs, and the inclusion of prior knowledge. Dealing with one challenge sometimes goes to the deterioration of another. The AGNEs technique simultaneously addresses many of these challenges. This can be concluded from the results of an extensive benchmark study applied to AGNEs and four state-of-the art process discovery algorithms. In this benchmark study, we make use of a new metric for quantifying specificity, based on the generation of artificial negative events. The benchmark experiments indicate that our technique can discover complex structures such as short loops, duplicate activities, and non-free choice constructs, while remaining robust to noise.

The declarative aspects of the AGNEs technique are useful in practice. This claim is supported by a real-life case study. The case study indicates that in practice the challenge of process discovery is to deal with unstructured processes, rather than to discover complex structures such as non-local, non-free choice, parallelism, and synchronization. In spite of that, our learning technique was capable of discovering a useful process model from the event log.

Another contribution is the definition of two new metrics for quantifying behavioral recall r_B^p and behavioral specificity s_B^n. These metrics are calculated by replaying the event log supplemented with artificial negative events. Because the metrics can be efficiently calculated, they produce an outcome on every discovered model in the experiments, facilitating comparison.

CHAPTER 7

Conclusion

7.1 Conclusion

In this text, we have characterized and introduced a number of declarative techniques for the modeling and mining of business processes intended to raise an organization's insight into its business processes.

Declarative process modeling has the potential of reconciling compliance and flexibility, both at the design-time and runtime. We start out from a general characterization of declarative process modeling. This characterization indicates how declarative process modeling techniques could contribute to flexibility and compliance. Moreover, the characterization contains important design principles for declarative modeling languages. These principles are to some extent present in existing works. An important contribution, is the identification of these works and a comparison based on considered state space and transition types.

Given the fact that there already exist a large number of formal declarative process modeling languages in the literature, introducing a new language, inevitably would borrow constructs from existing languages. Moreover, one language is likely only to cover a limited aspect of the many business process concerns that exist in reality. Rather than developing another formal language for declarative process modeling, we have presented a unifying framework, within which existing approaches can be positioned. In that respect, the EM-BrA^{2}CE Framework can be seen as a way to bring structure to the related work in the literature of both process modeling and service modeling. It consists of a unifying vocabulary and a unifying execution model. A remarkable property of the vocabulary is that it makes use of the SBVR ontology language. This ontology language has only recently appeared as an official specification of the Object Management Group (2008). The SBVR is perhaps a complex specification, but it has many appealing properties, such as its combination of linguistics and logics. The execution semantics of the framework is specified by means of colored Petri nets. The latter allows for the simulation of declarative process models in CPN Tools. These simulations are an important proof-of-concept of the execution semantics and furthermore provide artificial event logs useful in process mining experiments.

Albeit a unifying approach, the framework purposefully leaves many gaps to fill in. After defining the framework, we identify sixteen business rule types within the framework that allow to declaratively document several business concerns related to business processes. Documenting business concerns in declarative process models can already realize many of the design-time advantages of declarative process modeling. Realizing the run-time advantages, however, requires a specification in formal languages, that are automatically executable, and verifiable. PENELOPE is an example of such languages. It allows to express the dynamics of obligations, permissions, and due dates that come into existence during the enactment of business processes. For this language, we indicate how it can be visualized and verified – with many limitations for the time being.

A second part of the text deals with declarative techniques for process discovery within the EM-BrA^{2}CE Framework. Declarative process discovery techniques focuss on comprehensibility and justifiability of the discovered process models, in addition to their accuracy. Inductive Logic Programming (ILP) is a machine learning technique that is particularly suited for raising comprehensibility and justifiability. Ferreira and Ferreira (2006) show that process discovery can be formulated as an ILP classification learning problem on event logs with both positive and negative events. In reality, event logs often do not contain negative examples and therefore process discovery cannot be represented as a classification problem that discriminates between positive and negative events. To overcome the problem, we develop a configurable technique to artificially generate negative events (AGNEs). By generating artificial negative events, a classification learner is given a configurable completeness assumption as inductive bias. The idea of manipulating the inductive bias of learners by generating artificial negative events undoubtedly is one of the most surprising outcomes of the text. This idea was first explored for the learning of access rules from event logs, but fully implemented for the discovery of activity preconditions with the AGNEs process discovery technique. The AGNEs process discovery technique has been implemented as a mining plugin in the ProM framework. We call this learning technique declarative, because it allows the user to configure the prior knowledge, inductive bias, and language bias of the learner. In addition the generation of artificial negative events, the technique also shows how it is possible to convert a set of transition rules for each activity type, into a Petri net representation. The conversion of transition rules into a graphical model greatly facilitates comprehensibility of the discovered mining model.

The ability to artificially generate negative events, has lead to the specification of two new metrics for quantifying recall and specificity of a discovered process model. Because these metrics can be efficiently calculated, they have proven to be very useful to compare the AGNEs learning technique with existing state-of-the-art learning techniques. An important contribution, is the experimental setup of the extensive benchmark experiment. This experiment evaluates the expressiveness, robustness to noise, and ability to generalize of AGNEs in comparison to four state-of-the-art process discovery algorithms using both existing metrics and our own metrics. A comparative benchmark study of this scale is the first in the field of process discovery. The experiments show very good results for our learning technique. The artificial event logs, however, contain the events of simulated, structured processes. In reality, it might very well be that there are only few human-centric processes that actually take place in a structured manner. A real-life case study conducted for a large, European telecom provider, provided us with an event log of customer-initiated processes that clearly violates a number of assumptions that are traditionally made in the domain of process discovery. Although in comparison to other learners the AGNEs technique still is capable of discovering an acceptable process model from the event log, the case study is a further indication that research in process discovery should focuss on discovering less structured processes. In such as setting, process mining techniques that have a declarative representation with transition rules instead of Petri nets as language bias, are likely to be more useful.

7.2 Limitations and Future Work

Declarative process modeling and mining techniques still require substantial effort, in order to realize their design goals. Looking at our own techniques, we can suggest – among others – the following future work.

7.2.1 A Configurable Completeness Assumption

Although we have done a large number of experiments to evaluate the performance of AGNEs we did not manipulate the history size and parallel variants parameters. To what extent would it affect the ability of the algorithm of generalizing? Under the current settings –infinite window size and parallel variant calculation – a rather strong completeness assumption is made. By reducing the window size, a weaker completeness assumption is made, less negative events are generated, and the amount of allowed behavior increases. Under the current settings, AGNEs nonetheless has proven to be fairly robust to incomplete event logs.

Furthermore, it would be interesting to extend the configurability of the inductive bias. Currently, the user can have the algorithm take into account window size and parallel variants, when looking for non-existent behavior. However, the algorithm can be made even more configurable, for instance, by specifying whether ordering matters, or by imposing on the algorithm to filter out a particular activity type. Such configurability of the inductive bias would resemble to the approach of van der Aalst et al. (2006), which allows to configure how a finite state machine is constructed from the traces in an event log and how this state machine is folded into a Petri net.

7.2.2 Learning Other Transition Types

In this text, we have primarily focussed on the use of declarative classification techniques for the purpose of process discovery. The control flow aspects of process discovery only relate to *start* and *complete* transitions within the framework. However, the learning of the conditions that discriminate between positive and negative events for some other transition types within the framework, present relevant, and non-trivial challenges. In particular, the learning of access rules is a challenging problem that could also benefit from declarative learning features. As illustrated in section 5.9, learning access rules that do away with access right accumulation is a real challenge that requires a declarative manipulation of the inductive bias. Furthermore, the conversion of a set of access rules into a role-based access control specification (RBAC) with defeasible access constraints is a non-trivial challenge that could alleviate the role-engineering problem from a mining rather than from a modeling perspective. Future research has to reveal whether this approach could yield good results in practice.

7.2.3 Declarative Process Mining Models

The AGNEs process discovery technique has been provided with a language bias that can be converted into Petri nets. Whereas the use of Petri nets as external mining model representation language allowed for the use of existing evaluation metrics, the language bias of AGNEs by no means should be restricted to Petri net based languages only. AGNEs' technique for inducing artificial negative events is independent of the chosen language bias. Therefore, the learner could easily be provided with a language bias such as the ConDec language that allows for a declarative representation of the mining model. The contribution of AGNEs is that it enables the use of ILP classification learners even when the event log does not naturally contain negative events. The ability to choose a process representation language that is best capable of capturing the underlying behavior – potentially a declarative or a procedural language – in practice will undoubtedly be a huge advantage of the ILP learning technique. Therefore, future work should also focus on providing AGNEs with other, non-graphical, yet declarative process representation languages.

7.2.4 Learning from Unstructured Processes

Whereas the AGNEs process discovery algorithm shows good performance for event logs of structured processes, our telecom case study and other case studies in the literature reveal that the bulk of human-centric processes are essentially semi-structured or even unstructured. Most state-of-the art process discovery algorithms provide little added value on event logs of unstructured processes. Because more and more human-centric processes are computer-supported, there are also many event logs available of unstructured processes. Examples are unstructured workflow processes, online shopping, or self-scan supermarket visits. Because of its declarative nature, AGNEs is a learning technique that can be tailored towards unstructured processes. Essentially, this would require a combination of the two approaches: a more suitable language bias, and a probabilistic approach.

1. **language bias approach**: AGNEs has been designed for discovering rule-based Petri net patterns from event logs. Clearly, unstructured processes cannot easily be represented as Petri nets. However, other event operators might be more successful in capturing the hidden temporal structure of an event log. Instead of a graphical model, AGNEs is more likely to present a number of rules that best discriminate between the observed (positive events) and unobserved behavior (negative events).

2. **probabilistic approach**: Unstructured processes are essentially probabilistic: from a given state of a process, almost any transition can be observed with a non-zero probability. Probabilistic learning approaches, such as the ones intended by De Raedt and Kersting (2003), are particularly suited for the analysis of unstructured or semi-structured, human-centric processes, because their transition probabilities account for the uncertainty that originates from noise, unobserved states, and the lack of structure.

In the literature, there are already many formalisms to represent and learn the probability distributions of stochastic, *generative* grammars over sequences of observed characters and unobserved state variables. Historically, techniques like Markov models, hidden Markov models, factorial hidden Markov models, and dynamic Bayesian networks have been first applied to speech recognition and bioinformatics (Durbin et al., 1998). Each representation has its own particular modeling features that makes them more or less suited for representing human-centric behavior. Factorial hidden Markov models, for instance, have a distributed state representation, that allows for the modeling of concurrent behavior (Ghahramani and Jordan, 1997). Lafferty et al. (2001) show how conditional random fields are useful *discriminative* models that can be used for **activity segmentation,** the supervised learning of a labeling function from a finite set of training samples that gives meaning to segments of observed activities in terms of a sequence of predefined labels. Both for generative hidden Markov models and discriminative conditional random fields, a first-order, inductive logic programming (ILP) extension exists that allows for the representation of sequences of logical atoms rather than alphabets of flat characters (Kersting et al., 2006). These extensions with regard to language bias and probabilistic learning undoubtedly have much potential, but at the same time pose many challenges with regard to comprehensibility and justifiability.

LIST OF FIGURES

1.1 Modeling and mining in a multi-disciplinary BPM life cycle 2

2.1 Declarative rule enforcement: derivation 13
2.2 Declarative rule enforcement: authorization 14
2.3 Separating communication logic from process models 14

3.1 The transition types in the EM-BrA^{2}CE Framework 29
3.2 Payment-after-shipment 31
3.3 SBVR: conceptual schema and conceptual model 36
3.4 The SBVR Meaning and Representation Vocabulary 37
3.5 The SBVR Vocabulary for Describing Business Rules 38
3.6 A MOF/UML representation of the EM-BrA^{2}CE Vocabulary . . . 39
3.7 A UML 2.0 powertype representation of business concept type . . 40
3.8 State and state space in the EM-BrA^{2}CE Vocabulary 44
3.9 An illustration of the state transitions of a place order activity a1 . 45
3.10 The CP-Net places that span the EM-BrA^{2}CE state space 55
3.11 A CP-Net model of the Create transition 58
3.12 A CP-Net model of the Schedule transition 61
3.13 A CP-Net model of the Assign and Revoke transition 63
3.14 A CP-Net model of the Start transition 64
3.15 A CP-Net model of the fact manipulation transitions 66
3.16 A CP-Net model of the Complete transition 69
3.17 A CP-Net model of the Abort, Skip and Redo transition 70

4.1 Two process models visualized by PENELOPE 93
4.2 The state space of the example deontic assignments 94
4.3 A running CP-Net simulation of the Credit Application example . 100

4.4 Driver's license – simulated activity precondition 101
4.5 Credit application – simulated authorization rule 102

5.1 DriversLicensel – a tradeoff between generality and specificity . . . 110
5.2 Multi-relational datasets 118
5.3 Transformation to uni-relational problem 119
5.4 Deriving locality from parallelism 128
5.5 parallel variant calculation 129
5.6 DriversLicensel – Generating artificial negative events for an event log with two sequences 132
5.7 Petri net patterns in the language bias 134
5.8 DriversLicensel – results 137
5.9 AGNEs in ProM 4.2 142
5.10 Credit application – induced authorization rule 144

6.1 a5 – results 157
6.2 herbstFig6p33 – results 158
6.3 a10skip – results 158
6.4 betaSimplified – results 159
6.5 herbstFig6p38 – results 160
6.6 a6nfc – results 160
6.7 telecom – a visualization of queue routing 166
6.8 telecom – AGNEs mining results as a Petri net 171
6.9 telecom – genetic miner mining results as a heuristics net 172
6.10 telecom – heuristics miner mining results as a heuristics net 173

LIST OF TABLES

2.1 Concerns that affect processes 11
2.2 Procedural versus declarative process modeling 12
2.3 Chronological overview of declarative process modeling languages . 19

3.1 Traditional layering of ontologies 31

4.1 Relating transition types to business rule types 76
4.2 The event Calculus axioms 90
4.3 The PENELOPE axioms 91
4.4 Payment-after-shipment 92
4.5 Credit application – event log 102
4.6 An Overview of Process Mining Space
(van der Aalst et al., 2003; van der Aalst and Weijters, 2004) . . . 103

5.1 Chronological overview of process discovery algorithms 115

6.1 Event log properties 153
6.2 Parameter settings 155
6.3 zero-noise experiment - aggregated results 156
6.4 noise experiments - aggregated results 161
6.5 10-fold cross validation experiment - aggregated results 163
6.6 telecom – the properties of the original and preprocessed event log 165
6.7 telecom – parameter settings 169
6.8 telecom – mining results 170

BIBLIOGRAPHY

Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining process models from workflow logs. In Schek, H., Saltor, F., Ramos, I., and Alonso, G., editors, *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. (1996). *Fast discovery of association rules*, pages 307–328. American Association for Artificial Intelligence, Menlo Park, CA, USA.

Alonso, G., Dadam, P., and Rosemann, M., editors (2007). *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*. Springer.

Alves de Medeiros, A. K. (2006). *Genetic Process Mining*. PhD thesis, Technische Universiteit Eindhoven.

Alves de Medeiros, A. K. and Günther, C. W. (2005). Process mining: Using CPN Tools to create test logs for mining algorithms. In *Proceedings of the 6th Workshop on the Practical Use of Coulored Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, pages 177–190, Aarhus, Denmark.

Alves de Medeiros, A. K., van Dongen, B. F., van der Aalst, W. M. P., and Weijters, A. J. M. M. (2004). Process mining: Extending the alpha-algorithm to mine short loops. BETA working paper series 113, Eindhoven University of Technology.

Alves de Medeiros, A. K., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2007). Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304.

Angluin, D. and Smith, C. H. (1983). Inductive inference: Theory and methods. *ACM Computing Surveys*, 15(3):237–269.

Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2001). Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287.

Atkinson, C. and Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41.

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., and Vanthienen, J. (2003). Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society*, 54(6):627–635.

Bailey, J., Bry, F., and Patranjan, P.-L. (2005). Composite event queries for reactivity on the web. In Ellis, A. and Hagino, T., editors, *Proceedings of the 14th international conference on World Wide Web, Special interest tracks and posters (WWW 2005)*, pages 1082–1083. ACM.

Baisley, D. (2005). OMG and Business Rules. Presentation available at `http://www.omg.org/docs/omg/05-04-09.pdf`.

Baisley, D. E., Hall, J., and Chapin, D. (2005). Semantic Formulations in SBVR. In Hawke et al. (2005).

Bassiliades, N., Kontopoulos, E., and Antoniou, G. (2005). A visual environment for developing defeasible rule bases for the semantic web. In Adi, A., Stoutenburg, S., and Tabet, S., editors, *Proceedings of the 1st International Symposium Advances in Rule Interchange and Applications (RuleML 2005)*, volume 3791 of *Lecture Notes in Computer Science*, pages 172–186. Springer.

Bernstein, P. A. and Goodman, N. (1981). Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221.

Bhattacharya, K., Gerede, C. E., Hull, R., Liu, R., and Su, J. (2007). Towards formal analysis of artifact-centric business process models. In Alonso et al. (2007), pages 288–304.

Blockeel, H. (1998). *Top-Down Induction of First-Order Logical Decision Trees*. PhD thesis, Katholieke Universiteit Leuven.

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.

Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., and Vandecasteele, H. (2002). Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166.

Brachman, R. and Levesque, H. (2004). *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Broersen, J., Dignum, F., Dignum, V., and Meyer, J.-J. C. (2004). Designing a deontic logic of deadlines. In *Proceedings of the 7th International Workshop on Deontic Logic in Computer Science (DEON'04)*, pages 43–56.

Bussler, C. (2001). The role of B2B protocols in inter-enterprise process execution. In *TES '01: Proceedings of the 2nd International Workshop on Technologies for E-Services*, pages 16–29, London, UK. Springer-Verlag.

Cadez, I., Heckerman, D., Meek, C., Smyth, P., and White, S. (2003). Model-based clustering and visualization of navigation patterns on a web site. *Data Mining and Knowledge Discovery*, 7(4):399–424.

Canfora, G., García, F., Piattini, M., Ruiz, F., and Visaggio, C. A. (2005). A family of experiments to validate metrics for software process models. *Journal of Systems and Software*, 77(2):113–129.

Chakravarthy, S. and Mishra, D. (1994). Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26.

Chapin, D. (2005). Semantics of Business Vocabulary & Business Rules (SBVR). In Hawke et al. (2005).

Clarke, E. M., Grumberg, O., and Long, D. E. (1993). Verification tools for finite-state concurrent systems. In de Bakker et al. (1994), pages 124–175.

Cohen, W. (1995). Fast effective rule induction. In Prieditis, A. and Russell, S., editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA. Morgan Kaufmann Publishers.

Cook, J. and Wolf, A. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249.

Cortadella, J., Kishinevsky, M., Lavagno, L., and Yakovlev, A. (1998). Deriving petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. (2003). The next step in web services. *Communications of the ACM*, 46(10):29–34.

Davenport, T. H. (1993). *Process innovation: reengineering work through information technology.* Harvard Business School Press, Boston, MA, USA.

de Bakker, J. W., de Roever, W. P., and Rozenberg, G., editors (1994). *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science.* Springer.

De Raedt, L. and Kersting, K. (2003). Probabilistic logic learning. *SIGKDD Explorations*, 5(1):31–48.

Debevoise, T. (2005). *Business Process Management With a Business Rules Approach: Implementing the Service Oriented Architecture.* Business Knowledge Architects.

Digital Business Ecosystem (DBE) (2007). Sbeaver. `http://sbeaver.sourceforge.net`.

Dumas, M., van der Aalst, W. M. P., and ter Hofstede, A. H. M. (2005). *Process-aware information systems: bridging people and software through process technology.* John Wiley & Sons, Inc., New York, NY, USA.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge university press.

Dustdar, S., Fiadeiro, J., and Sheth, A., editors (2006). *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*. Springer.

Džeroski, S. (2003). Multi-relational data mining: an introduction. *SIGKDD Explorations*, 5(1):1–16.

Džeroski, S. and Lavrač, N. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, New York.

Džeroski, S. and Lavrač, N., editors (2001). *Relational Data Mining.* Springer-Verlag, Berlin.

Dybowski, R., Laskey, K. B., Myers, J. W., and Parsons, S. (2003). Introduction to the special issue on the fusion of domain knowledge with data for decision support. *Journal of Machine Learning Research*, 4:293–294.

Eshghi, K. (1988). Abductive planning with event calculus. In *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP)*, pages 562–579.

Eshuis, R. and Dehnert, J. (2003). Reactive Petri Nets for Workflow Modeling. In van der Aalst, W. M. P. and Best, E., editors, *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 296–315. Springer.

Fensel, D. and Bussler, C. (2002). The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137.

Ferraiolo, D. F., Sandhu, R. S., Gavrila, S. I., Kuhn, D. R., and Chandramouli, R. (2001). Proposed nist standard for role-based access control. *ACM Transactions on Information System Security*, 4(3):224–274.

Ferreira, D. R. and Ferreira, H. M. (2005). Learning, planning, and the life cycle of workflow management. In *Proceedings of the 9th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005)*, pages 39–46. IEEE Computer Society.

Ferreira, D. R., Zacarias, M., Malheiros, M., and Ferreira, P. (2007). Approaching process mining with sequence clustering: Experiments and findings. In Alonso et al. (2007), pages 360–374.

Ferreira, H. and Ferreira, D. (2006). An integrated life cycle for workflow management based on learning and planning. *International Journal of Cooperative Information Systems*, 15(4):485–505.

Føllesdal, D. and Hilpinen, R. (1971). Deontic logic: An introduction. In Hilpinen, R., editor, *Deontic Logic: Introductory and Systematic Readings*, pages 1–35. D. Reidel Publishing Company, Dordrecht.

Galton, A. and Augusto, J. C. (2002). Two approaches to event definition. In Hameurlain, A., Cicchetti, R., and Traunmüller, R., editors, *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA 2002)*, volume 2453 of *Lecture Notes in Computer Science*, pages 547–556. Springer.

Gatziu, S. and Dittrich, K. (1993). Events in an Active Object-Oriented Database System. In Paton, N. and Williams, H., editors, *Proceedings of the first International Workshop on Rules in Database Systems (RIDS)*, Edinburgh, UK. Springer-Verlag, Workshops in Computing.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden markov models. *Machine Learning*, 29(2-3):245–273.

Goedertier, S. (2007a). A Vocabulary and Execution Model for Declarative Service Orchestration, 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Brisbane, Australia, September 24, 2007. Presentation.

Goedertier, S. (2007b). FETEW Doctoral Seminar, Declarative BPM: on the use of Business Rules in the BPM Life Cycle, Leuven, Belgium, May 14, 2007. Presentation.

Goedertier, S. (2007c). Process Mining as First-Order Classification Learning on Logs with Negative Events, 3rd Workshop on Business Processes Intelligence (BPI'07), Brisbane, Australia, September 24, 2007. Presentation.

Goedertier, S. (2008). Analyzing Business Processes using Data Mining: Data Mining Garden, workshop on New Frontiers in Data Mining, Leuven, Belgium, January 9, 2008. Presentation.

Goedertier, S., Haesen, R., and Vanthienen, J. (2007a). EM-BrA^{2}CE v0.1: A vocabulary and execution model for declarative business process modeling. FETEW Research Report KBI_0728, K.U.Leuven.

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007b). A new approach for discovering business process models from event logs. FETEW Research Report KBI_0716, K.U.Leuven.

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007c). Process mining as first-order classification learning on logs with negative events. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 42–53. Springer.

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2008a). Process Mining as First-Order Classification Learning on Logs with Negative Events. In *Proceedings of the 3rd Workshop on Business Processes Intelligence (BPI'07)*, volume 4928 of *Lecture Notes in Computer Science*. Springer.

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008b). Benchmarking state-of-the-art algorithms for process discovery. submitted for review to the journal of *Informations Systems* on September 1, 2008.

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008c). Robust process discovery with artificial negative events. resubmitted for review to the *Journal of Machine Learning Research* on September 1, 2008.

Goedertier, S., Mues, C., and Vanthienen, J. (2007d). Specifying process-aware access control rules in SBVR. In Paschke, A. and Biletskiy, Y., editors, *Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007)*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer. (Best Paper Award).

Goedertier, S. and Vanthienen, J. (2005a). Rule-based Business Process Modeling and Execution. IEEE EDOC Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005).

Goedertier, S. and Vanthienen, J. (2005b). Rule-based business process modeling and execution. In *Proceedings of the IEEE EDOC Workshop on Vocabularies Ontologies and Rules for The Enterprise (VORTE 2005). CTIT Workshop Proceeding Series (ISSN 0929-0672)*, Enschede.

Goedertier, S. and Vanthienen, J. (2006a). Bedrijfsregels voor conforme en flexibele bedrijfsprocessen. *Business InZicht*, 21.

Goedertier, S. and Vanthienen, J. (2006b). Business rules for compliant business process models. In Abramowicz, W. and Mayr, H. C., editors, *Proceedings of the 9th International Conference on Business Information Systems (BIS 2006)*, volume 85 of *Lecture Notes in Informatics*, pages 558–572. Gesellschaft für Informatik.

Goedertier, S. and Vanthienen, J. (2006c). Compliant and flexible business processes with business rules. In Regev, G., Soffer, P., and Schmidt, R., editors, *Proceedings of the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) at CAiSE'06*, pages 94–104. Presses Universitaires de Namur.

Goedertier, S. and Vanthienen, J. (2006d). Designing compliant business processes with obligations and permissions. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer.

Goedertier, S. and Vanthienen, J. (2007a). Declarative process modeling with business vocabulary and business rules. In Meersman, R., Tari, Z., and Herrero, P., editors, *OTM Workshops (1)*, volume 4805 of *Lecture Notes in Computer Science*, pages 603–612. Springer.

Goedertier, S. and Vanthienen, J. (2007b). A vocabulary and execution model for declarative service orchestration. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Proceedings of the 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 496–501. Springer.

Goedertier, S. and Vanthienen, J. (2008). Rule-based business process modeling and enactment. *International Journal of Business Process Integration and Management.* (accepted for publication).

Goethals, F. G., Snoeck, M., Lemahieu, W., and Vandenbulcke, J. (2007). Considering (de)centralization in a web services world. In *International Conference on Internet and Web Applications and Services (ICIW 2007)*, page 22. IEEE Computer Society.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10(5):447–474.

Governatori, G. (2005). Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216.

Governatori, G. and Rotolo, A. (2002). A gentzen system for reasoning with contrary-to-duty obligations. a preliminary study. In Jones, A. J. and Horty, J., editors, *Proceedings of the 6th International Workshop on Deontic Logic in Computer Science (DEON'02)*, pages 97–116, London. Imperial College.

Grosof, B. N., Labrou, Y., and Chan, H. Y. (1999). A declarative approach to business rules in contracts: courteous logic programs in XML. In *ACM Conference on Electronic Commerce*, pages 68–77.

Guizzardi, G. and Wagner, G. (2005). *Some Applications of a Unified Foundational Ontology in Business Modeling*, chapter in: Ontologies and Business Systems Analysis, ed. M. Rosemann and P. Green, pages 345–367. IDEA Publisher.

Günther, C. W. and van der Aalst, W. M. P. (2005). Modeling the case handling principles with colored petri nets. In Jensen, K., editor, *Proceedings of the 6th Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, pages 211–230, Aarhus, Denmark.

Günther, C. W. and van der Aalst, W. M. P. (2007). Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In Alonso et al. (2007), pages 328–343.

Gupta, A., Sagiv, Y., Ullman, J. D., and Widom, J. (1994). Efficient and complete tests for database integrity constraint checking. In Borning, A., editor, *PPCP*, volume 874 of *Lecture Notes in Computer Science*, pages 173–180. Springer.

Hadeli, K., Valckenaers, P., Kollingbaum, M., and Brussel, H. (2004). Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53(1):75–96.

Haesen, R., Goedertier, S., Van de Cappelle, K., Lemahieu, W., Snoeck, M., and Poelmans, S. (2007). A phased deployment of a workflow infrastructure in the enterprise architecture. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 270–280. Springer.

Halpin, T. A. (1991). A fact-oriented approach to schema transformation. In Thalheim et al. (1991), pages 342–356.

Halpin, T. A. (2000). A fact-oriented approach to business rules. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2000)*, pages 582–583.

Halpin, T. A. (2004). Information modeling and higher-order types. In Grundspenkis, J. and Kirikova, M., editors, *CAiSE Workshops (1)*, pages 233–248. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia.

Hammer, M. and Champy, J. (1993). *Reengineering the corporation.* Harper Collins, New York, NY.

Hammori, M., Herbst, J., and Kleiner, N. (2006). Interactive workflow mining - requirements, concepts and implementation. *Data & Knowledge Engineering*, 56(1):41–63.

Hawke, S., de Sainte Marie, C., and Tabet, S., editors (2005). *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA.* W3C.

Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., and Teschke, M. (1999). A comprehensive approach to flexibility in workflow management systems. *SIGSOFT Software Engineering Notes*, 24(2):79–88.

Herbst, J. and Karagiannis, D. (2004). Workflow mining with InWoLvE. *Computers in Industry*, 53(3):245–264.

Hogger, C. J. (1990). *Essentials of logic programming.* Oxford University Press, Inc., New York, NY, USA.

InterNational Committee for Information Technology Standards (INCITS) (2004). Role-Based Access Control. `http://csrc.nist.gov/rbac`. American National Standard ANSI/INCITS 359-2004.

Jablonski, S. and Bussler, C. (1996). *Workflow Management. Modeling Concepts, Architecture and Implementation.* International Thomson Computer Press, London.

Jensen, K. (1993). An introduction to the theoretical aspects of coloured petri nets. In de Bakker et al. (1994), pages 230–272.

Jensen, K. (1996). *Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use: volume 1.* Springer-Verlag, London, UK.

Kardasis, P. and Loucopoulos, P. (2005). A roadmap for the elicitation of business rules in information systems projects. *Business Process Management Journal*, 11(4):316–348.

Kersting, K., Raedt, L. D., and Raiko, T. (2006). Logical hidden markov models. *Journal of Artificial Intelligence Research*, 25:425–456.

Klein, M., Dellarocas, C., and Bernstein, A. (2000). Introduction to the special issue on adaptive workflow systems. *Computer Supported Cooperative Work*, 9(3/4):265–267.

Knottenbelt, J. and Clark, K. (2004). An architecture for contract-based communicating agents. In *Proceedings of the 2nd European Workshop on Multi-Agent Systems.*

Kontopoulos, E., Bassiliades, N., and Antoniou, G. (2006). Visualizing defeasible logic rules for the semantic web. In Mizoguchi, R., Shi, Z., and Giunchiglia, F., editors, *ASWC*, volume 4185 of *Lecture Notes in Computer Science*, pages 278–292. Springer.

Kopecký, J., Roman, D., Moran, M., and Fensel, D. (2006). Semantic web services grounding. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, page 127. IEEE Computer Society.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4(1):67–95.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Brodley, C. E. and Danyluk, A. P., editors, *Proceedings of the 18th International Conference on Machine Learning (ICML 2001)*, pages 282–289. Morgan Kaufmann.

Lamma, E., Mello, P., Montali, M., Riguzzi, F., and Storari, S. (2007). Inducing declarative logic-based models from labeled traces. In Alonso et al. (2007), pages 344–359.

Lavrac, N., Džeroski, S., and Grobelnik, M. (1991). Learning Nonrecursive Definitions of Relations with LINUS. In Kodratoff, Y., editor, *EWSL*, volume 482 of *Lecture Notes in Computer Science*, pages 265–281. Springer.

Lemahieu, W., Snoeck, M., Michiels, C., Goethals, F. G., Dedene, G., and Vandenbulcke, J. (2003). Event based web service description and coordination. In Bussler, C., Fensel, D., Orlowska, M. E., and Yang, J., editors, *WES*, volume 3095 of *Lecture Notes in Computer Science*, pages 120–133. Springer.

Leung, C. M. R. and Nijssen, G. M. (1988). Relational database design using the NIAM conceptual schema. *Information Systems*, 13(2):219–227.

Li, N., Grosof, B. N., and Feigenbaum, J. (2003). Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information System Security*, 6(1):128–171.

Ly, L., Rinderle, S., Dadam, P., and Reichert, M. (2005). Mining staff assignment rules from event-based data. In Bussler, C. and Haller, A., editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Lyytinen, K., Mathiassen, L., Ropponen, J., and Datta, A. (1998). Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3):275–301.

M. Zaremba, C. B. (2005). Towards dynamic execution semantics in semantic web services. In *Proceedings of the Workshop on Web Service Semantics: Towards Dynamic Business Integration, International Conference on the World Wide Web (WWW2005)*, Chiba, Japan.

Maher, M. J., Rock, A., Antoniou, G., Billington, D., and Miller, T. (2001). Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(4):483–501.

Mannila, H. and Meek, C. (2000). Global partial orders from sequential data. In *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00)*, pages 161–168, New York, NY, USA. ACM.

Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289.

Mansar, S. and Reijers, H. A. (2005). Best practices in business process redesign: validation of a redesign framework. *Computers in Industry*, 56(5):457–471.

Marín, R. H. and Sartor, G. (1999). Time and norms: a formalisation in the event-calculus. In *ICAIL '99: Proceedings of the 7th international conference on Artificial intelligence and law*, pages 90–99, New York, NY, USA. ACM Press.

Martens, D. (2008). *Building acceptable classification models for financial engineering applications.* Phd thesis, Katholieke Universiteit Leuven, Faculty of Business and Economics, Leuven.

Martens, D., Baesens, B., Van Gestel, T., and Vanthienen, J. (2007a). Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research*, 183(3):1466–1476.

Martens, D., Vanthienen, J., Goedertier, S., and Baesens, B. (2007b). Placing process intelligence within the business intelligence framework. In *Proceedings of the 6th International Conference on Information and Management Sciences (IMS 2007).*

Maruster, L. (2003). *A machine learning approach to understand business processes.* PhD thesis, Eindhoven University of Technology, Eindhoven.

Maruster, L., Weijters, A. J. M. M., van der Aalst, W. M. P., and van den Bosch, A. (2006). A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Mining and Knowledge Discovery*, 13(1):67–87.

Mendling, J., Reijers, H. A., and Cardoso, J. (2007). What makes process models understandable? In Alonso et al. (2007), pages 48–63.

Milner, R., Tofte, M., and Harper, R. (1990). *The definition of Standard ML.* MIT Press, Cambridge, MA, USA.

Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

Muggleton, S. (1990). Inductive logic programming. In *Proceedings of the 1st International Conference on Algorithmic Learning Theory*, pages 42–62.

Muggleton, S. (1996). Learning from positive data. In Muggleton, S., editor, *Inductive Logic Programming Workshop*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer.

Mulyar, N. and van der Aalst, W. M. P. (2005). Patterns in Colored Petri Nets. BETA Working Paper Series WP 139, Eindhoven University of Technology, Eindhoven.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Nau, D., Ghallab, M., and Traverso, P. (2004). *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Nute, D. (1994). *Handbook of Logic in Artificial Intelligence and Logic Programming*, chapter Defeasible Logic, page 353395. Oxford University Press.

Object Management Group (2005). UML 2.0 Superstructure Specification. OMG Document – formal/05-07-04.

Object Management Group (2006a). Business Motivation Model (BMM) – adopted specification. OMG Document – dtc/2006-08-03.

Object Management Group (2006b). Business Process Modeling Notation (BPMN) – final adopted specification. OMG Document – dtc/06-02-01.

Object Management Group (2006c). UML 2.0 Infrastructure Specification. OMG Document – formal/05-07-05.

Object Management Group (2008). Semantics of Business Vocabulary and Business Rules (SBVR), v1.0. OMG Document – dtc/08-01-02.

O'Conor, M. (2005). The implications of Sarbanes-Oxley for non-US IT departments. *Network Security*, 2005(7):17–20.

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production.* Productivity Press.

O'Sullivan, J., Edmond, D., and ter Hofstede, A. H. M. (2002). What's in a service? towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, 12(2/3):117–133.

Pallas Athena (2008a). BPM|FLOWer – case handling & workflow implementation. `http:\www.pallas-athena.com`.

Pallas Athena (2008b). BPM|Protos – process modeling and mining tool. `http:\www.pallas-athena.com`.

Paschke, A. and Bichler, M. (2005). SLA Representation, Management and Enforcement. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 158–163, Washington, DC, USA. IEEE Computer Society.

Pazzani, M., Mani, S., and Shankle, W. (2001). Acceptance by medical experts of rules generated by machine learning. *Methods of Information in Medicine*, 40(5):380–385.

Pei, J., Wang, H., Liu, J., Wang, K., Wang, J., and Yu, P. S. (2006). Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1467–1481.

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, 28(10):46–52.

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007a). Declare: Full support for loosely-structured processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–300. IEEE Computer Society.

Pesic, M., Schonenberg, M. H., Sidorova, N., and van der Aalst, W. M. P. (2007b). Constraint-based workflow models: Change made easy. In Meersman, R. and Tari, Z., editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer.

Pesic, M. and van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer.

Porter, M. (1985). *Competitive Advantage.* Free Press, New York.

PricewaterhouseCoopers (2008). Société Générale – synthèse du diagnostic de PwC et analyse du plan d'action. `http://www.socgen.com`.

Process Mining Group, TU/Eindhoven (2008). Process mining web page: research, tools and application. `http://processmining.org`.

Provost, F. J., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the 15th International Conference on Machine Learning (ICML '98)*, pages 445–453, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Quinlan, J. (1993). *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Reichert, M. and Dadam, P. (1998). ADEPT$_{\text{flex}}$-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129.

Reijers, H. A. and Limam, S. (2005). Best practices in business process redesign: An overview and qualitative evaluation of successful redesign heuristics. *OMEGA - The International Journal of Management Science*, 33(4):283–306.

Roberts, J. and et al. (2001). Tentative hold protocol part 2: Technical specification. W3C note, World Wide Web Consortium. `http://www.w3.org/TR/tenthold-2/`.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology. *Applied Ontology*, 1(1):77–106.

Ross, R. G. (2003). *Principles of the Business Rule Approach.* Addison-Wesley Professional.

Rozinat, A., Alves De Medeiros, A. K., Günther, C. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2007). Towards an evaluation framework for process mining algorithms. BETA working paper series 224, Eindhoven University of Technology.

Rozinat, A. and van der Aalst, W. M. P. (2006). Decision mining in ProM. In Dustdar et al. (2006), pages 420–425.

Rozinat, A. and van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95.

Russell, N., van der Aalst, W. M. P., and ter Hofstede, A. H. M. (2006). Workflow exception patterns. In Dubois, E. and Pohl, K., editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 288–302. Springer.

Sadiq, S., Governatori, G., and Namiri, K. (2007). Modeling control objectives for business process compliance. In Alonso et al. (2007), pages 149–164.

Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, 29(2):38–47.

Schmidt, K. (1999). Of maps and scripts: The status of formal constructs in cooperative work. *Information & Software Technology*, 41(6):319–329.

Schmidt, K. and Simone, C. (1996). Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work*, 5(2/3):155–200.

Segerberg, K. (1982). A deontic logic of action. *Studia Logica*, 10:269–282.

Shanahan, M. (1997). *Solving the frame problem: a mathematical investigation of the common sense law of inertia.* MIT Press, Cambridge, MA, USA.

Silva, R., Zhang, J., and Shanahan, J. (2005). Probabilistic workflow mining. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 275–284, New York, NY, USA. ACM.

Smyth, P. (1997). Clustering sequences with hidden markov models. In *Advances in Neural Information Processing Systems*, volume 9, pages 648–654. MIT Press.

Snoeck, M., Lemahieu, W., Goethals, F., Dedene, G., and Vandenbulcke, J. (2004). Events as atomic contracts for component integration. *Data & Knowledge Engineering*, 51(1):81–107.

Strembeck, M. and Neumann, G. (2004). An integrated approach to engineer and enforce context constraints in RBAC environments. *ACM Transactions on Information System Security*, 7(3):392–427.

Suchman, L. A. (1987). *Plans and situated actions: the problem of human-machine communication.* Cambridge University Press, New York, NY, USA.

Suchman, L. A. (1995). Making work visible. *Communications of the ACM*, 38(9):56–64.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining.* Addison Wesley.

Thalheim, B., Demetrovics, J., and Gerhardt, H.-D., editors (1991). *MFDBS 91, 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Bases Systems, Rostock, Germany, May 6-9, 1991, Proceedings*, volume 495 of *Lecture Notes in Computer Science*. Springer.

The OWL Services Coalition (2004). OWL-S 1.1. Available from `http://www.daml.org/services/owl-s/1.1/`.

The OWL Services Coalition (2006). OWL-S 1.2 Pre-Release. Available from `http://www.ai.sri.com/daml/services/owl-s/1.2/`.

Unisys (2005). Unisys rules modeler. `www.unisys.com/eprise/main/admin/corporate/doc/Unisys_Rules_Modeler_Insert.pdf` [10-11-2005].

van der Aalst, W. M. P. (1997). Verification of workflow nets. In Azéma, P. and Balbo, G., editors, *Proceedings of the 18th International Conference on the Application and Theory of Petri Nets 1997 (ICATPN '97)*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer.

van der Aalst, W. M. P. (1998). The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

van der Aalst, W. M. P. (2007). Exploring the CSCW spectrum using process mining. *Advanced Engineering Informatics*, 21(2):191–199.

van der Aalst, W. M. P. and Jablonski, S. (2000). Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276.

van der Aalst, W. M. P., Reijers, H. A., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593.

van der Aalst, W. M. P., Reijers, H. A., Weijters, A. J. M. M., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., and Verbeek, H. M. W. (2007a). Business process mining: An industrial application. *Information Systems*, 32(5):713–732.

van der Aalst, W. M. P., Rosemann, M., and Dumas, M. (2007b). Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511.

van der Aalst, W. M. P., Rubin, V., van Dongen, B. F., Kindler, E., , and Günther, C. W. (2006). Process mining: A two-step approach using transition systems and regions. BPM-06-30, BPM Center Report.

van der Aalst, W. M. P. and Song, M. (2004). Mining social networks: Uncovering interaction patterns in business processes. In *Proceedings of the International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer.

van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). YAWL: yet another workflow language. *Information Systems*, 30(4):245–275.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267.

van der Aalst, W. M. P. and van Hee, K. M. (2002). *Workflow Management: Models, Methods, and Systems.* MIT Press, Cambridge, MA.

van der Aalst, W. M. P., van Hee, K. M., and Houben, G. J. (1994). Modelling and analysing workflow using a Petri-net based approach. In De Michelis, G., Ellis, C., and Memmi, G., editors, *Proceedings of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50.

van der Aalst, W. M. P. and Weijters, A. J. M. M. (2004). Process mining: a research agenda. *Computers in Industry*, 53(3):231–244.

van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

van der Aalst, W. M. P., Weske, M., and Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162.

van Dongen, B. F., Alves de Medeiros, A. K., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The ProM Framework: A New Era in Process Mining Tool Support. In Ciardo, G. and Darondeau, P., editors, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer.

van Dongen, B. F. and van der Aalst, W. M. P. (2005a). A meta model for process mining data. In Missikoff, M. and De Nicola, A., editors, *EMOI-INTEROP*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org.

van Dongen, B. F. and van der Aalst, W. M. P. (2005b). Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. In *Proceedings of the 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB).*

van Hee, K. M., Oanea, O., Serebrenik, A., Sidorova, N., and Voorhoeve, M. (2006). History-based joins: Semantics, soundness and implementation. In Dustdar et al. (2006), pages 225–240.

Van Nuffelen, B. and Kakas, A. C. (2001). A-system: Declarative programming with abduction. In Eiter, T., Faber, W., and Truszczynski, M., editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *Lecture Notes in Computer Science*, pages 393–396. Springer.

W3C (2004a). Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C Recommendation 10 February 2004.

W3C (2004b). Web Services Architecture. Technical report, W3C Working Group Note 11 February 2004.

Wagner, G. (1991). A database needs two kinds of negation. In Thalheim et al. (1991), pages 357–371.

Wagner, G. (2003). The agent-object-relationship metamodel: towards a unified view of state and behavior. *Information Systems*, 28(5):475–504.

Wainer, J., Kumar, A., and Barthelmess, P. (2007). DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems*, 32(3):365–384.

Weijters, A. J. M. M. and van der Aalst, W. M. P. (2003). Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162.

Weijters, A. J. M. M., van der Aalst, W. M. P., and Alves de Medeiros, A. K. (2006). Process mining with the heuristics miner-algorithm. BETA working paper series 166, Eindhoven University of Technology.

Weiss, G., editor (1999). *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT Press, Cambridge, MA, USA.

Wen, L., van der Aalst, W. M. P., Wang, J., and Sun, J. (2007). Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180.

Wielemaker, J. (2003). An overview of the SWI-Prolog programming environment. In Mesnard, F. and Serebenik, A., editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium. Katholieke Universiteit Leuven. CW 371.

Witten, I. H. and Frank, E. (2000). *Data mining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Woolridge, M. and Wooldridge, M. J. (2001). *Introduction to Multiagent Systems.* John Wiley & Sons, Inc., New York, NY, USA.

Workflow Management Coalition (WfMC) (1995). The Workflow Reference Model. Document Number TC00-1003, issue 1.1.

Workflow Management Coalition (WfMC) (1999). Terminology & Glossary. Document Number WFMC-TC-1011, issue 3.

Yolum, P. and Singh, M. P. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253.

Zhang, L., Ahn, G.-J., and tseng Chu, B. (2003). A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information System Security*, 6(3):404–441.

zur Muehlen, M. (2004). *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems.*, volume 6 of *Advances in Information Systems and Management Science.* Logos, Berlin.

Stijn Goedertier

⋆	January 16, 1981
✉	Willem Coosemansstraat 152
	3010 Kessel-lo
	Belgium
e-✉	stijngoedertier@yahoo.com
📱	+32 496 105 390

education

2004-2008	PhD in Applied Economic Sciences
	Katholieke Universiteit Leuven
	Faculty of Business and Economics
	PhD thesis: “Declarative Techniques for Modeling and Mining Business Processes”
2002-2003	ERASMUS Exchange: HEC Lausanne, Switzerland
1999-2004	Master in Business Engineering, Management Informatics (handelsingenieur in de beleidsinformatica)
	Katholieke Universiteit Leuven
	Faculty of Economic and Applied Economic Sciences
	results: magna cum laude (every year)
	master thesis: “Towards an integration of OLAP and Data Mining”
1992-1999	Greek–mathematics (8hrs)
	Onze-Lieve-Vrouwecollege Oudenaarde

professional experience

2008	IT management advisor (from October 2008)
	PricewaterhouseCoopers
2004-2008	researcher (Business Process Management)
	Katholieke Universiteit Leuven
2003	internship
	Cognos Benelux
	implementation of Cognos Powerplay and ReportNet at customer site

extra curricular

2004-2008	scuba diving: CMAS 3*Diver, NELOS duiker-hulpverlener, CMAS basic nitrox diver
2006-2008	lifeguard for scuba diving club Poseidon Leuven
2006	lifeguard (Vlaamse redderscentrale)
2002-2007	member of the VVW Nieuwpoort Bénéteau 25/7.5 sailing team
2002-2008	Vlaamse Zeezeilschool: kustvaart 1, 2, en 3. Algemeen stuurbrevet. Marifoonbrevet.

1997-1998	student council president
1996-1999	writing and editing the student's school paper
1991-1995	member of swimming club OZEKA Oudenaarde

doctoral program

2004-2005	**doctoral courses**: D650 Introduction to Research Methodologies H02A3A Programming Languages and Methodologies HD85 Machine learning and inductive inference D373 Multivariate statistics in business and industry H02c4a Artificial Neural Networks
2005	**comprehensive exam**: Rule-based process modeling and execution
2006	**research proposal**: On the use of business rules throughout the BPM life cycle.
2007	**1st seminar**: On the use of business rules throughout the BPM life cycle, Leuven, May 14, 2007.
2008	**2nd seminar**: Robust process discovery with artificial negative events, Leuven, February 14, 2008.
2008	**pre-defense**: Declarative techniques for modeling and mining business processes, Leuven, May 22, 2008.
2008	**defense**: Declarative techniques for modeling and mining business processes, Leuven, September 16, 2008.

academic experience

2008	ad-hoc *reviewer* for the Machine Learning special issue on "Swarm Intelligence"
2008	ad-hoc *reviewer* for the Information Systems special issue on "Business Process Modeling"
2008	ad-hoc *reviewer* for the Springer journal "LNCS Transactions on Petri Nets and Other Models of Concurrency" (ToPNoC)
2008	*program committee member* of RuleML-2008: The International RuleML Symposium on Rule Interchange and Applications
2007	*reviewer* for the Second International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2007), Zhengzhou (China), September 14-17 2007

teaching experience

2006	course: Business Intelligence class: decision trees & artificial neural networks
2007	course: Logica voor informatici: toepassingen in de beleidsinformatica class: An introduction to logic programming in Prolog, Leuven, Leuven, December 14, 2007.
2008	course: Knowledge Management & Business Intelligence

	class: Business process intelligence: quantifying process performance and process compliance, Leuven, April 8, 2008.

grants & awards

2008	*granted* the one-year fulltime position of postdoctoral researcher (PDMK) by KULeuven. Submitted research proposal: "Analysing Semi-Structured, Human Behavior with Probabilistic Sequence Analysis Techniques"
2007	*best paper award* for Goedertier, S., Mues, C., and Vanthienen, J. (2007d). Specifying process-aware access control rules in SBVR. In Paschke, A. and Biletskiy, Y., editors, *Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007)*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer. (Best Paper Award)

presentations

2008	Goedertier, S. (2008). Analyzing Business Processes using Data Mining: Data Mining Garden, workshop on New Frontiers in Data Mining, Leuven, Belgium, January 9, 2008. Presentation
2007	Goedertier, S. (2007c). Process Mining as First-Order Classification Learning on Logs with Negative Events, 3rd Workshop on Business Processes Intelligence (BPI'07), Brisbane, Australia, September 24, 2007. Presentation
2007	Goedertier, S. (2007a). A Vocabulary and Execution Model for Declarative Service Orchestration, 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Brisbane, Australia, September 24, 2007. Presentation
2007	Goedertier, S. (2007b). FETEW Doctoral Seminar, Declarative BPM: on the use of Business Rules in the BPM Life Cycle, Leuven, Belgium, May 14, 2007. Presentation
2005	Goedertier, S. and Vanthienen, J. (2005a). Rule-based Business Process Modeling and Execution. IEEE EDOC Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005)

publications

submitted for publication in international journals

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008c). Robust process discovery with artificial negative events. resubmitted for review to the *Journal of Machine Learning Research* on September 1, 2008

Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2008b). Benchmarking state-of-the-art algorithms for process discovery. submitted for review to the journal of *Informations Systems* on September 1, 2008

accepted for publication in international journals

Goedertier, S. and Vanthienen, J. (2008). Rule-based business process modeling and enactment. *International Journal of Business Process Integration and Management.* (accepted for publication)

published in international journals

Goedertier, S., Mues, C., and Vanthienen, J. (2007d). Specifying process-aware access control rules in SBVR. In Paschke, A. and Biletskiy, Y., editors, *Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007)*, volume 4824 of *Lecture Notes in Computer Science*, pages 39–52. Springer. (Best Paper Award)

Goedertier, S. and Vanthienen, J. (2007a). Declarative process modeling with business vocabulary and business rules. In Meersman, R., Tari, Z., and Herrero, P., editors, *OTM Workshops (1)*, volume 4805 of *Lecture Notes in Computer Science*, pages 603–612. Springer

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007c). Process mining as first-order classification learning on logs with negative events. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 42–53. Springer

Haesen, R., Goedertier, S., Van de Cappelle, K., Lemahieu, W., Snoeck, M., and Poelmans, S. (2007). A phased deployment of a workflow infrastructure in the enterprise architecture. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 270–280. Springer

Goedertier, S. and Vanthienen, J. (2007b). A vocabulary and execution model for declarative service orchestration. In ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., editors, *Proceedings of the 2nd Workshop on Advances in Semantics for Web services (semantics4ws'07), Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 496–501. Springer

Goedertier, S. and Vanthienen, J. (2006d). Designing compliant business processes with obligations and permissions. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer

Dutch publications

Goedertier, S. and Vanthienen, J. (2006a). Bedrijfsregels voor conforme en flexibele bedrijfsprocessen. *Business InZicht*, 21

announcements made at international conferences

Martens, D., Vanthienen, J., Goedertier, S., and Baesens, B. (2007b). Placing process intelligence within the business intelligence framework. In *Proceedings* *of the 6th International Conference on Information and Management Sciences (IMS 2007)*

Goedertier, S. and Vanthienen, J. (2006c). Compliant and flexible business processes with business rules. In Regev, G., Soffer, P., and Schmidt, R., editors, *Proceedings of the 7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) at CAiSE'06*, pages 94–104. Presses Universitaires de Namur

Goedertier, S. and Vanthienen, J. (2006b). Business rules for compliant business process models. In Abramowicz, W. and Mayr, H. C., editors, *Proceedings of the 9th International Conference on Business Information Systems (BIS 2006)*, volume 85 of *Lecture Notes in Informatics*, pages 558–572. Gesellschaft für Informatik

Goedertier, S. and Vanthienen, J. (2005b). Rule-based business process modeling and execution. In *Proceedings of the IEEE EDOC Workshop on Vocabularies Ontologies and Rules for The Enterprise (VORTE 2005). CTIT Workshop Proceeding Series (ISSN 0929-0672)*, Enschede

internal reports (IR)

Goedertier, S., Haesen, R., and Vanthienen, J. (2007a). EM-BrA^{2}CE v0.1: A vocabulary and execution model for declarative business process modeling. FETEW Research Report KBI_0728, K.U.Leuven

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007b). A new approach for discovering business process models from event logs. FETEW Research Report KBI_0716, K.U.Leuven

Doctoral Dissertations from the Faculty of Business and Economics

From August 1, 1971.

1. GEPTS Stefaan (1971)
   Stability and efficiency of resource allocation processes in discrete commodity spaces. Leuven, K.U. Leuven, 1971. 86 pp.

2. PEETERS Theo (1971)
   Determinanten van de internationale handel in fabrikaten. Leuven, Acco, 1971. 290 pp.

3. VAN LOOY Wim (1971)
   Personeelsopleiding: een onderzoek naar investeringsaspekten van opleiding. Hasselt, Vereniging voor wetenschappelijk onderzoek in Limburg, 1971. VII, 238 pp.

4. THARAKAN Mathew (1972)
   Indian exports to the European community: problems and prospects. Leuven, Faculty of economics and applied economics, 1972. X, 343 pp.

5. HERROELEN Willy (1972)
   Heuristische programmatie: methodologische benadering en praktische toepassing op complexe combinatorische problemen. Leuven, Aurelia scientifica, 1972. X, 367 pp.

6. VANDENBULCKE Jacques (1973)
   De studie en de evaluatie van data-organisatiemethodes en data-zoekmethodes. Leuven, s.n., 1973. 3 V.

7. PENNYCUICK Roy A. (1973)
   The economics of the ecological syndrome. Leuven, Acco, 1973. XII, 177 pp.

8. KAWATA T. Bualum (1973)
   Formation du capital d'origine belge, dette publique et stratégie du développement au Zaire. Leuven, K.U. Leuven, 1973. V, 342 pp.

9. DONCKELS Rik (1974)
   Doelmatige oriëntering van de sectorale subsidiepolitiek in België: een theoretisch onderzoek met empirische toetsing. Leuven, K.U. Leuven, 1974. VII, 156 pp.

10. VERHELST Maurice (1974)
    Contribution to the analysis of organizational information systems and their financial benefits. Leuven, K.U. Leuven, 1974. 2 V.

11. CLEMEUR Hugo (1974)
Enkele verzekeringstechnische vraagstukken in het licht van de nutstheorie. Leuven, Aurelia scientifica, 1974. 193 pp.

12. HEYVAERT Edward (1975)
De ontwikkeling van de moderne bank- en krediettechniek tijdens de zestiende en zeventiende eeuw in Europa en te Amsterdam in het bijzonder. Leuven, K.U. Leuven, 1975. 186 pp.

13. VERTONGHEN Robert (1975)
Investeringscriteria voor publieke investeringen: het uitwerken van een operationele theorie met een toepassing op de verkeersinfrastructuur. Leuven, Acco, 1975. 254 pp.

14. Niet toegekend.

15. VANOVERBEKE Lieven (1975)
Microeconomisch onderzoek van de sectoriële arbeidsmobiliteit. Leuven, Acco, 1975. 205 pp.

16. DAEMS Herman (1975)
The holding company: essays on financial intermediation, concentration and capital market imperfections in the Belgian economy. Leuven, K.U.Leuven, 1975. XII, 268 pp.

17. VAN ROMPUY Eric (1975)
Groot-Brittannië en de Europese monetaire integratie: een onderzoek naar de gevolgen van de Britse toetreding op de geplande Europese monetaire unie. Leuven, Acco, 1975. XIII, 222 pp.

18. MOESEN Wim (1975)
Het beheer van de staatsschuld en de termijnstructuur van de intrestvoeten met een toepassing voor België. Leuven, Vander, 1975. XVI, 250 pp.

19. LAMBRECHT Marc (1976)
Capacity constrained multi-facility dynamic lot-size problem. Leuven, K.U. Leuven, 1976. 165 pp.

20. RAYMAECKERS Erik (1976)
De mens in de onderneming en de theorie van het producenten-gedrag: een bijdrage tot transdisciplinaire analyse. Leuven, Acco, 1976. XIII, 538 pp.

21. TEJANO Albert (1976)
Econometric and input-output models in development planning: the case of the Philippines. Leuven, K.U. Leuven, 1976. XX, 297 pp.

22. MARTENS Bernard (1977)
Prijsbeleid en inflatie met een toepassing op België. Leuven, K.U. Leuven, 1977. IV, 253 pp.

23. VERHEIRSTRAETEN Albert (1977)
Geld, krediet en intrest in de Belgische financiële sector. Leuven, Acco, 1977. XXII, 377 pp.

24. GHEYSSENS Lieven (1977)
International diversification through the government bond market: a risk-return analysis. Leuven, s.n., 1977. 188 pp.

25. LEFEBVRE Chris (1977)
Boekhoudkundige verwerking en financiële verslaggeving van huurkooptransacties en verkopen op afbetaling bij ondernemingen die aan consumenten verkopen. Leuven, K.U. Leuven, 1977. 228 pp.

26. KESENNE Stefan (1978)
Tijdsallocatie en vrijetijdsbesteding: een econometrisch onderzoek. Leuven, s.n., 1978. 163 pp.

27. VAN HERCK Gustaaf (1978)
Aspecten van optimaal bedrijfsbeleid volgens het marktwaardecriterium: een risico-rendements-analyse. Leuven, K.U. Leuven, 1978. IV, 163 pp.

28. VAN POECK André (1979)
World price trends and price and wage development in Belgium: an investigation into the relevance of the Scandinavian model of inflation for Belgium. Leuven, s.n., 1979. XIV, 260 pp.

29. VOS Herman (1978)
De industriële technologieverwerving in Brazilië: een analyse. Leuven, s.n., 1978. onregelmatig gepagineerd.

30. DOMBRECHT Michel (1979)
Financial markets, employment and prices in open economies. Leuven, K.U. Leuven, 1979. 182 pp.

31. DE PRIL Nelson (1979)
Bijdrage tot de actuariële studie van het bonus-malussysteem. Brussel, OAB, 1979. 112 pp.

32. CARRIN Guy (1979)
Economic aspects of social security: a public economics approach. Leuven, K.U. Leuven, 1979. onregelmatig gepagineerd

33. REGIDOR Baldomero (1979)
An empirical investigation of the distribution of stock-market prices and weak-form efficiency of the Brussels stock exchange. Leuven, K.U. Leuven, 1979. 214 pp.

34. DE GROOT Roger (1979)
Ongelijkheden voor stop loss premies gebaseerd op E.T. systemen in het kader van de veralgemeende convexe analyse. Leuven, K.U. Leuven, 1979. 155 pp.

35. CEYSSENS Martin (1979)
On the peak load problem in the presence of rationizing by waiting. Leuven, K.U. Leuven, 1979. IX, 217 pp.

36. ABDUL RAZK Abdul (1979)
Mixed enterprise in Malaysia: the case study of joint venture between Malysian public corporations and foreign enterprises. Leuven, K.U. Leuven, 1979. 324 pp.

37. DE BRUYNE Guido (1980)
Coordination of economic policy: a game-theoretic approach. Leuven, K.U. Leuven, 1980. 106 pp.

38. KELLES Gerard (1980)
Demand, supply, price change and trading volume on financial markets of the matching-order type. = Vraag, aanbod, koersontwikkeling en omzet op financiële markten van het Europese type. Leuven, K.U. Leuven, 1980. 222 pp.

39. VAN EECKHOUDT Marc (1980)
De invloed van de looptijd, de coupon en de verwachte inflatie op het opbrengstverloop van vastrentende finaciële activa. Leuven, K.U. Leuven, 1980. 294 pp.

40. SERCU Piet (1981)
Mean-variance asset pricing with deviations from purchasing power parity. Leuven, s.n., 1981. XIV, 273 pp.

41. DEQUAE Marie-Gemma (1981)
Inflatie, belastingsysteem en waarde van de onderneming. Leuven, K.U. Leuven, 1981. 436 pp.

42. BRENNAN John (1982)
An empirical investigation of Belgian price regulation by prior notification: 1975 - 1979 - 1982. Leuven, K.U. Leuven, 1982. XIII, 386 pp.

43. COLLA Annie (1982)
Een econometrische analyse van ziekenhuiszorgen. Leuven, K.U. Leuven, 1982. 319 pp.

44. Niet toegekend.

45. SCHOKKAERT Eric (1982)
Modelling consumer preference formation. Leuven, K.U. Leuven, 1982. VIII, 287 pp.

46. DEGADT Jan (1982)
Specificatie van een econometrisch model voor vervuilingsproblemen met proeven van toepassing op de waterverontreiniging in België. Leuven, s.n., 1982. 2 V.

47. LANJONG Mohammad Nasir (1983)
A study of market efficiency and risk-return relationships in the Malaysian capital market. s.l., s.n., 1983. XVI, 287 pp.

48. PROOST Stef (1983)
De allocatie van lokale publieke goederen in een economie met een centrale overheid en lokale overheden. Leuven, s.n., 1983. onregelmatig gepagineerd.

49. VAN HULLE Cynthia (1983)
Shareholders' unanimity and optimal corporate decision making in imperfect capital markets. s.l., s.n., 1983. 147 pp. + appendix.

50. VAN WOUWE Martine (2/12/83)
Ordening van risico's met toepassing op de berekening van ultieme ruïnekansen. Leuven, s.n., 1983. 109 pp.

51. D'ALCANTARA Gonzague (15/12/83)
SERENA: a macroeconomic sectoral regional and national account econometric model for the Belgian economy. Leuven, K.U. Leuven, 1983. 595 pp.

52. D'HAVE Piet (24/02/84)
De vraag naar geld in België. Leuven, K.U. Leuven, 1984. XI, 318 pp.

53. MAES Ivo (16/03/84)
The contribution of J.R. Hicks to macro-economic and monetary theory. Leuven, K.U. Leuven, 1984. V, 224 pp.

54. SUBIANTO Bambang (13/09/84)
A study of the effects of specific taxes and subsidies on a firms' R&D investment plan. s.l., s.n., 1984. V, 284 pp.

55. SLEUWAEGEN Leo (26/10/84)
Location and investment decisions by multinational enterprises in Belgium and Europe. Leuven, K.U. Leuven, 1984. XII, 247 pp.

56. GEYSKENS Erik (27/03/85)
Produktietheorie en dualiteit. Leuven, s.n., 1985. VII, 392 pp.

57. COLE Frank (26/06/85)
Some algorithms for geometric programming. Leuven, K.U. Leuven, 1985. 166 pp.

58. STANDAERT Stan (26/09/86)
A study in the economics of repressed consumption. Leuven, K.U. Leuven, 1986. X, 380 pp.

59. DELBEKE Jos (03/11/86)
Trendperioden in de geldhoeveelheid van België 1877-1983: een theoretische en empirische analyse van de "Banking school" hypothese. Leuven, KUL, 1986. XII, 430 pp.

60. VANTHIENEN Jan (08/12/86)
Automatiseringsaspecten van de specificatie, constructie en manipulatie van beslissingstabellen. Leuven, s.n., 1986. XIV, 378 pp.

61. LUYTEN Robert (30/04/87)
A systems-based approach for multi-echelon production/inventory systems. s.l., s.n., 1987. 3V.

62. MERCKEN Roger (27/04/87)
De invloed van de data base benadering op de interne controle. Leuven, s.n., 1987. XIII, 346 pp.

63. VAN CAYSEELE Patrick (20/05/87)
Regulation and international innovative activities in the pharmaceutical industry. s.l., s.n., 1987. XI, 169 pp.

64. FRANCOIS Pierre (21/09/87)
De empirische relevantie van de independence from irrelevant alternatives. Assumptie indiscrete keuzemodellen. Leuven, s.n., 1987. IX, 379 pp.

65. DECOSTER André (23/09/88)
Family size, welfare and public policy. Leuven, KUL. Faculteit Economische en Toegepaste Economische Wetenschappen, 1988. XIII, 444 pp.

66. HEIJNEN Bart (09/09/88)
Risicowijziging onder invloed van vrijstellingen en herverzekeringen: een theoretische analyse van optimaliteit en premiebepaling. Leuven, KUL. Faculteit Economische en Toegepaste Economische Wetenschappen, 1988. onregelmatig gepagineerd.

67. GEEROMS Hans (14/10/88)
Belastingvermijding. Theoretische analyse van de determinanten van de belastingontduiking en de belastingontwijking met empirische verificaties. Leuven, s.n., 1988. XIII, 409, 5 pp.

68. PUT Ferdi (19/12/88)
Introducing dynamic and temporal aspects in a conceptual (database) schema. Leuven, KUL. Faculteit Economische en Toegepaste Economische Wetenschappen, 1988. XVIII, 415 pp.

69. VAN ROMPUY Guido (13/01/89)
A supply-side approach to tax reform programs. Theory and empirical evidence for Belgium. Leuven, KUL. Faculteit Economische en Toegepaste Economische Wetenschappen, 1989. XVI, 189, 6 pp.

70. PEETERS Ludo (19/06/89)
Een ruimtelijk evenwichtsmodel van de graanmarkten in de E.G.: empirische specificatie en beleidstoepassingen. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1989. XVI, 412 pp.

71. PACOLET Jozef (10/11/89)
Marktstructuur en operationele efficiëntie in de Belgische financiële sector. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1989. XXII, 547 pp.

72. VANDEBROEK Martina (13/12/89)
Optimalisatie van verzekeringscontracten en premieberekeningsprincipes. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1989. 95 pp.

73. WILLEKENS Francois (1990)
Determinance of government growth in industrialized countries with applications to Belgium. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1990. VI, 332 pp.

74. VEUGELERS Reinhilde (02/04/90)
Scope decisions of multinational enterprises. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1990. V, 221 pp.

75. KESTELOOT Katrien (18/06/90)
Essays on performance diagnosis and tacit cooperation in international oligopolies. Leuven, K.U. Leuven. Faculteit Economische en Toegepaste Economische Wetenschappen, 1990. 227 pp.

76. WU Changqi (23/10/90) Strategic aspects of oligopolistic vertical integration. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1990. VIII, 222 pp.

77. ZHANG Zhaoyong (08/07/91)
A disequilibrium model of China's foreign trade behaviour. Leuven, K.U.
Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1991. XII, 256 pp.

78. DHAENE Jan (25/11/91)
Verdelingsfuncties, benaderingen en foutengrenzen van stochastische grootheden geassocieerd aan verzekeringspolissen en -portefeuilles. Leuven, K.U.
Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1991. 146 pp.

79. BAUWELINCKX Thierry (07/01/92)
Hierarchical credibility techniques. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. 130 pp.

80. DEMEULEMEESTER Erik (23/3/92)
Optimal algorithms for various classes of multiple resource-constrained project scheduling problems. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. 180 pp.

81. STEENACKERS Anna (1/10/92)
Risk analysis with the classical actuarial risk model: theoretical extensions and applications to Reinsurance. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. 139 pp.

82. COCKX Bart (24/09/92)
The minimum income guarantee. Some views from a dynamic perspective. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. XVII, 401 pp.

83. MEYERMANS Eric (06/11/92)
Econometric allocation systems for the foreign exchange market: Specification, estimation and testing of transmission mechanisms under currency substitution. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. XVIII, 343 pp.

84. CHEN Guoqing (04/12/92)
Design of fuzzy relational databases based on fuzzy functional dependency. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1992. 176 pp.

85. CLAEYS Christel (18/02/93)
Vertical and horizontal category structures in consumer decision making: The nature of product hierarchies and the effect of brand typicality. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 348 pp.

86. CHEN Shaoxiang (25/03/93)
The optimal monitoring policies for some stochastic and dynamic production processes. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 170 pp.

87. OVERWEG Dirk (23/04/93)
Approximate parametric analysis and study of cost capacity management of computer configurations. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 270 pp.

88. DEWACHTER Hans (22/06/93)
Nonlinearities in speculative prices: The existence and persistence of nonlinearity in foreign exchange rates. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 151 pp.

89. LIN Liangqi (05/07/93)
Economic determinants of voluntary accounting choices for R & D expenditures in Belgium. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 192 pp.

90. DHAENE Geert (09/07/93)
Encompassing: formulation, properties and testing. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 117 pp.

91. LAGAE Wim (20/09/93)
Marktconforme verlichting van soevereine buitenlandse schuld door private crediteuren: een neo-institutionele analyse. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 241 pp.

92. VAN DE GAER Dirk (27/09/93)
Equality of opportunity and investment in human capital. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1993. 172 pp.

93. SCHROYEN Alfred (28/02/94)
Essays on redistributive taxation when monitoring is costly. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 203 pp. + V.

94. STEURS Geert (15/07/94)
Spillovers and cooperation in research and development. Leuven, K.U.
Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 266 pp.

95. BARAS Johan (15/09/94)
The small sample distribution of the Wald, Lagrange multiplier and likelihood ratio tests for homogeneity and symmetry in demand analysis: a Monte Carlo study. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 169 pp.

96. GAEREMYNCK Ann (08/09/94)
The use of depreciation in accounting as a signalling device. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 232 pp.

97. BETTENDORF Leon (22/09/94)
A dynamic applied general equilibrium model for a small open economy. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 149 pp.

98. TEUNEN Marleen (10/11/94)
Evaluation of interest randomness in actuarial quantities. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1994. 214 pp.

99. VAN OOTEGEM Luc (17/01/95)
An economic theory of private donations. Leuven. K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 236 pp.

100. DE SCHEPPER Ann (20/03/95)
Stochastic interest rates and the probabilistic behaviour of actuarial functions. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 211 pp.

101. LAUWERS Luc (13/06/95)
Social choice with infinite populations. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 79 pp.

102. WU Guang (27/06/95)
A systematic approach to object-oriented business modeling. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 248 pp.

103. WU Xueping (21/08/95)
Term structures in the Belgian market: model estimation and pricing error analysis. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 133 pp.

104. PEPERMANS Guido (30/08/95)
Four essays on retirement from the labor force. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 128 pp.

105. ALGOED Koen (11/09/95)
Essays on insurance: a view from a dynamic perspective. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 136 pp.

106. DEGRYSE Hans (10/10/95)
Essays on financial intermediation, product differentiation, and market structure. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 218 pp.

107. MEIR Jos (05/12/95)
Het strategisch groepsconcept toegepast op de Belgische financiële sector. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1995. 257 pp.

108. WIJAYA Miryam Lilian (08/01/96)
Voluntary reciprocity as an informal social insurance mechanism: a game theoretic approach. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 124 pp.

109. VANDAELE Nico (12/02/96)
The impact of lot sizing on queueing delays: multi product, multi machine models. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 243 pp.

110. GIELENS Geert (27/02/96)
Some essays on discrete time target zones and their tails. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 131 pp.

111. GUILLAUME Dominique (20/03/96)
Chaos, randomness and order in the foreign exchange markets. Essays on the modelling of the markets. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 171 pp.

112. DEWIT Gerda (03/06/96)
Essays on export insurance subsidization. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 186 pp.

113. VAN DEN ACKER Carine (08/07/96)
Belief-function theory and its application to the modeling of uncertainty in financial statement auditing. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 147 pp.

114. IMAM Mahmood Osman (31/07/96)
Choice of IPO Flotation Methods in Belgium in an Asymmetric Information Framework and Pricing of IPO's in the Long-Run. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 221 pp.

115. NICAISE Ides (06/09/96)
Poverty and Human Capital. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1996. 209 pp.

116. EYCKMANS Johan (18/09/97)
On the Incentives of Nations to Join International Environmental Agreements. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1997. XV + 348 pp.

117. CRISOLOGO-MENDOZA Lorelei (16/10/97)
Essays on Decision Making in Rural Households: a study of three villages in the Cordillera Region of the Philippines. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1997. 256 pp.

118. DE REYCK Bert (26/01/98)
Scheduling Projects with Generalized Precedence Relations: Exact and Heuristic Procedures. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. XXIV + 337 pp.

119. VANDEMAELE Sigrid (30/04/98)
Determinants of Issue Procedure Choice within the Context of the French IPO Market: Analysis within an Asymmetric Information Framework. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. 241 pp.

120. VERGAUWEN Filip (30/04/98)
Firm Efficiency and Compensation Schemes for the Management of Innovative Activities and Knowledge Transfers. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. VIII + 175 pp.

121. LEEMANS Herlinde (29/05/98)
The Two-Class Two-Server Queueing Model with Nonpreemptive Heterogeneous Priority Structures. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. 211 pp.

122. GEYSKENS Inge (4/09/98)
Trust, Satisfaction, and Equity in Marketing Channel Relationships. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. 202 pp.

123. SWEENEY John (19/10/98)
Why Hold a Job ? The Labour Market Choice of the Low-Skilled. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1998. 278 pp.

124. GOEDHUYS Micheline (17/03/99)
Industrial Organisation in Developing Countries, Evidence from Côte d'Ivoire. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 251 pp.

125. POELS Geert (16/04/99)
On the Formal Aspects of the Measurement of Object-Oriented Software Specifications. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 507 pp.

126. MAYERES Inge (25/05/99)
The Control of Transport Externalities: A General Equilibrium Analysis. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. XIV + 294 pp.

127. LEMAHIEU Wilfried (5/07/99)
Improved Navigation and Maintenance through an Object-Oriented Approach to Hypermedia Modelling. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 284 pp.

128. VAN PUYENBROECK Tom (8/07/99)
Informational Aspects of Fiscal Federalism. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 192 pp.

129. VAN DEN POEL Dirk (5/08/99)
Response Modeling for Database Marketing Using Binary Classification. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 342 pp.

130. GIELENS Katrijn (27/08/99)
International Entry Decisions in the Retailing Industry: Antecedents and Performance Consequences. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 336 pp.

131. PEETERS Anneleen (16/12/99)
Labour Turnover Costs, Employment and Temporary Work. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 207 pp.

132. VANHOENACKER Jurgen (17/12/99)
Formalizing a Knowledge Management Architecture Meta-Model for Integrated Business Process Management. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 1999. 252 pp.

133. NUNES Paulo (20/03/2000)
Contingent Valuation of the Benefits of Natural Areas and its Warmglow Component. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. XXI + 282 pp.

134. VAN DEN CRUYCE Bart (7/04/2000)
Statistische discriminatie van allochtonen op jobmarkten met rigide lonen. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. XXIII + 441 pp.

135. REPKINE Alexandre (15/03/2000)
Industrial restructuring in countries of Central and Eastern Europe: Combining branch-, firm- and product-level data for a better understanding of Enterprises' behaviour during transition towards market economy. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. VI + 147 pp.

136. AKSOY, Yunus (21/06/2000)
Essays on international price rigidities and exchange rates. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. IX + 236 pp.

137. RIYANTO, Yohanes Eko (22/06/2000)
Essays on the internal and external delegation of authority in firms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. VIII + 280 pp.

138. HUYGHEBAERT, Nancy (20/12/2000)
The Capital Structure of Business Start-ups. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2000. VIII + 332 pp.

139. FRANCKX Laurent (22/01/2001)
Ambient Inspections and Commitment in Environmental Enforcement. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. VIII + 286 pp.

140. VANDILLE Guy (16/02/2001)
Essays on the Impact of Income Redistribution on Trade. Leuven, K.U.Leu-ven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. VIII + 176 pp.

141. MARQUERING Wessel (27/04/2001)
Modeling and Forecasting Stock Market Returns and Volatility. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. V + 267 pp.

142. FAGGIO Giulia (07/05/2001)
Labor Market Adjustment and Enterprise Behavior in Transition. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. 150 pp.

143. GOOS Peter (30/05/2001)
The Optimal Design of Blocked and Split-plot experiments. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. X + 224 pp.

144. LABRO Eva (01/06/2001)
Total Cost of Ownership Supplier Selection based on Activity Based Costing and Mathematical Programming. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. 217 pp.

145. VANHOUCKE Mario (07/06/2001)
Exact Algorithms for various Types of Project Scheduling Problems. Nonregular Objectives and time/cost Trade-offs. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. 316 pp.

146. BILSEN Valentijn (28/08/2001)
Entrepreneurship and Private Sector Development in Central European Transition Countries. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. XVI + 188 pp.

147. NIJS Vincent (10/08/2001)
Essays on the dynamic Category-level Impact of Price promotions. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001.

148. CHERCHYE Laurens (24/09/2001)
Topics in Non-parametric Production and Efficiency Analysis. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. VII + 169 pp.

149. VAN DENDER Kurt (15/10/2001)
Aspects of Congestion Pricing for Urban Transport. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. VII + 203 pp.

150. CAPEAU Bart (26/10/2001)
In defence of the excess demand approach to poor peasants' economic behaviour. Theory and Empirics of non-recursive agricultural household modelling. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. XIII + 286 pp.

151. CALTHROP Edward (09/11/2001)
Essays in urban transport economics. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001.

152. VANDER BAUWHEDE Heidi (03/12/2001)
Earnings management in an Non-Anglo-Saxon environment. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2001. 408 pp.

153. DE BACKER Koenraad (22/01/2002)
Multinational firms and industry dynamics in host countries : the case of Belgium. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. VII + 165 pp.

154. BOUWEN Jan (08/02/2002)
Transactive memory in operational workgroups. Concept elaboration and case study. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. 319 pp. + appendix 102 pp.

155. VAN DEN BRANDE Inge (13/03/2002)
The psychological contract between employer and employee : a survey among Flemish employees. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. VIII + 470 pp.

156. VEESTRAETEN Dirk (19/04/2002)
Asset Price Dynamics under Announced Policy Switching. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. 176 pp.

157. PEETERS Marc (16/05/2002)
One Dimensional Cutting and Packing : New Problems and Algorithms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. IX + 247 pp.

158. SKUDELNY Frauke (21/05/2002)
Essays on The Economic Consequences of the European Monetary Union. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002.

159. DE WEERDT Joachim (07/06/2002)
Social Networks, Transfers and Insurance in Developing countries. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. VI + 129 pp.

160. TACK Lieven (25/06/2002)
Optimal Run Orders in Design of Experiments. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. XXXI + 344 pp.

161. POELMANS Stephan (10/07/2002)
Making Workflow Systems work. An investigation into the Importance of Task-appropriation fit, End-user Support and other Technological Characteristics. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. 237 pp.

162. JANS Raf (26/09/2002)
Capacitated Lot Sizing Problems : New Applications, Formulations and Algorithms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002.

163. VIAENE Stijn (25/10/2002)
Learning to Detect Fraud from enriched Insurance Claims Data (Context, Theory and Applications). Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. 315 pp.

164. AYALEW Tekabe (08/11/2002)
Inequality and Capital Investment in a Subsistence Economy.Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. V + 148 pp.

165. MUES Christophe (12/11/2002)
On the Use of Decision Tables and Diagrams in Knowledge Modeling and Verification. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. 222 pp.

166. BROCK Ellen (13/03/2003)
The Impact of International Trade on European Labour Markets. K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002.

167. VERMEULEN Frederic (29/11/2002)
Essays on the collective Approach to Household Labour Supply. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. XIV + 203 pp.

168. CLUDTS Stephan (11/12/2002)
Combining participation in decision-making with financial participation: theoretical and empirical perspectives. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002. XIV + 247 pp.

169. WARZYNSKI Frederic (09/01/2003)
The dynamic effect of competition on price cost margins and innovation. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

170. VERWIMP Philip (14/01/2003)
Development and genocide in Rwanda ; a political economy analysis of peasants and power under the Habyarimana regime. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

171. BIGANO Andrea (25/02/2003)
Environmental regulation of the electricity sector in a European Market Framework. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003. XX + 310 pp.

172. MAES Konstantijn (24/03/2003)
Modeling the Term Structure of Interest Rates Across Countries. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003. V+246 pp.

173. VINAIMONT Tom (26/02/2003)
The performance of One- versus Two-Factor Models of the Term Structure of Interest Rates. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen. 2003.

174. OOGHE Erwin (15/04/2003)
Essays in multi-dimensional social choice. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003. VIII+108 pp.

175. FORRIER Anneleen (25/04/2003)
Temporary employment, employability and training. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

176. CARDINAELS Eddy (28/04/2003)
The role of cost system accuracy in managerial decision making. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003. 144 pp.

177. DE GOEIJ Peter (02/07/2003)
Modeling Time-Varying Volatility and Interest Rates. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003. VII+225 pp.

178. LEUS Roel (19/09/2003)
The generation of stable project plans. Complexity and exact algorithms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

179. MARINHEIRO Carlos (23/09/2003)
EMU and fiscal stabilisation policy : the case of small countries. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

180. BAESENS Bart (24/09/2003)
Developing intelligent systems for credit scoring using machine learning techniques. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

181. KOCZY Laszlo (18/09/2003)
Solution concepts and outsider behaviour in coalition formation games. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

182. ALTOMONTE Carlo (25/09/2003)
Essays on Foreign Direct Investment in transition countries : learning from the evidence. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

183. DRIES Liesbeth (10/11/2003)
Transition, Globalisation and Sectoral Restructuring: Theory and Evidence from the Polish Agri-Food Sector. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

184. DEVOOGHT Kurt (18/11/2003)
Essays On Responsibility-Sensitive Egalitarianism and the Measurement of Income Inequality. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

185. DELEERSNYDER Barbara (28/11/2003)
Marketing in Turbulent Times. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

186. ALI Daniel (19/12/2003)
Essays on Household Consumption and Production Decisions under Uncertainty in Rural Ethiopia. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2003.

187. WILLEMS Bert (14/01/2004)
Electricity networks and generation market power. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

188. JANSSENS Gust (30/01/2004)
Advanced Modelling of Conditional Volatility and Correlation in Financial Markets. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

189. THOEN Vincent (19/01/2004)
On the valuation and disclosure practices implemented by venture capital providers. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

190. MARTENS Jurgen (16/02/2004)
A fuzzy set and stochastic system theoretic technique to validate simulation models. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

191. ALTAVILLA Carlo (21/05/2004)
Monetary policy implementation and transmission mechanisms in the Euro area. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

192. DE BRUYNE Karolien (07/06/2004)
Essays in the location of economic activity. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

193. ADEM Jan (25/06/2004)
Mathematical programming approaches for the supervised classification problem. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

194. LEROUGE Davy (08/07/2004)
Predicting Product Preferences : the effect of internal and external cues. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

195. VANDENBROECK Katleen (16/07/2004)
Essays on output growth, social learning and land allocation in agriculture : micro-evidence from Ethiopia and Tanzania. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

196. GRIMALDI Maria (03/09/2004)
The exchange rate, heterogeneity of agents and bounded rationality. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

197. SMEDTS Kristien (26/10/2004)
Financial integration in EMU in the framework of the no-arbitrage theory. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

198. KOEVOETS Wim (12/11/2004)
Essays on Unions, Wages and Employment. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

199. CALLENS Marc (22/11/2004)
Essays on multilevel logistic Regression. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

200. RUGGOO Arvind (13/12/2004)
Two stage designs robust to model uncertainty. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2004.

201. HOORELBEKE Dirk (28/01/2005)
Bootstrap and Pivoting Techniques for Testing Multiple Hypotheses. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

202. ROUSSEAU Sandra (17/02/2005)
Selecting Environmental Policy Instruments in the Presence of Incomplete Compiance. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

203. VAN DER MEULEN Sofie (17/02/2005)
Quality of Financial Statements : Impact of the external auditor and applied accounting standards. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

204. DIMOVA Ralitza (21/02/2005)
Winners and Losers during Structural Reform and Crisis : the Bulgarian Labour Market Perspective. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

205. DARKIEWICZ Grzegorz (28/02/2005)
Value-at-risk in Insurance and Finance : the Comonotonicity Approach. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

206. DE MOOR Lieven (20/05/2005)
The Structure of International Stock Returns : Size, Country and Sector Effects in Capital Asset Pricing. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

207. EVERAERT Greetje (27/06/2005)
Soft Budget Constraints and Trade Policies : The Role of Institutional and External Constraints. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

208. SIMON Steven (06/07/2005)
The Modeling and Valuation of complex Derivatives : The Impact of the Choice of the term structure model. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

209. MOONEN Linda (23/09/2005)
Algorithms for some Graph-Theoretical Optimization Problems. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

210. COUCKE Kristien (21/09/2005)
Firm and industry adjustment under de-industrialisation and globalization of the Belgian economy. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

211. DECAMPS Marc (21/10/2005)
Some actuarial and financial applications of generalized diffusion processes. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

212. KIM Helena (29/11/2005)
Escalation games: an instrument to analyze conflicts. The strategic approach to the bargaining problem. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2005.

213. GERMENJI Etleva (06/01/2006)
Essays on the Economics of Emigration from Albania. Leuven, K.U.Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

214. BELIEN Jeroen (18/01/2006)
Exact and Heuristic Methodologies for Scheduling in Hospitals: Problems, Formulations and Algorithms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

215. JOOSSENS Kristel (10/02/2006)
Robust discriminant analysis. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

216. VRANKEN Liesbet (13/02/2006)
Land markets and production efficiency in transition economies. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

217. VANSTEENKISTE Isabel (22/02/2006)
Essays on non-linear modelling in international macroeconomics. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

218. WUYTS Gunther (31/03/2006)
Essays on the liquidity of financial markets. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

219. DE BLANDER Rembert (28/04/2006)
Essays on endogeneity and parameter heterogeneity in cross-section and panel data. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

220. DE LOECKER Jan (12/05/2006)
Industry dynamics and productivity. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

221. LEMMENS Aurélie (12/05/2006)
Advanced classification and time-series methods in marketing. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

222. VERPOORTEN Marijke (22/05/2006)
Conflict and survival: an analysis of shocks, coping strategies and economic mobility in Rwanda, 1990-2002. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

223. BOSMANS Kristof (26/05/2006)
Measuring economic inequality and inequality aversion. Leuven, K.U.
Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

224. BRENKERS Randy (29/05/2006)
Policy reform in a market with differentiated products: applications from the car market. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

225. BRUYNEEL Sabrina (02/06/2006)
Self-control depletion: Mechanisms and its effects on consumer behavior. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

226. FAEMS Dries (09/06/2006)
Collaboration for innovation: Processes of governance and learning. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

227. BRIERS Barbara (28/06/2006)
Countering the scrooge in each of us: on the marketing of cooperative behavior. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

228. ZANONI Patrizia (04/07/2006)
Beyond demography: Essays on diversity in organizations. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

229. VAN DEN ABBEELE Alexandra (11/09/2006)
Management control of interfirm relations: the role of information. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

230. DEWAELHEYNS Nico (18/09/2006)
Essays on internal capital markets, bankruptcy and bankruptcy reform. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

231. RINALDI Laura (19/09/2006)
Essays on card payments and household debt. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

232. DUTORDOIR Marie (22/09/2006)
Determinants and stock price effects of Western European convertible debt offerings: an empirical analysis. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

233. LYKOGIANNI Elissavet (20/09/2006)
Essays on strategic decisions of multinational enterprises: R&D decentralization, technology transfers and modes of foreign entry. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

234. ZOU Jianglei (03/10/2006)
Inter-firm ties, plant networks, and multinational firms: essays on FDI and trade by Japanse firms. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

235. GEYSKENS Kelly (12/10/2006)
The ironic effects of food temptations on self-control performance. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

236. BRUYNSEELS Liesbeth (17/10/2006)
Client strategic actions, going-concern audit opinions and audit reporting errors. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

237. KESSELS Roselinde (23/10/2006)
Optimal designs for the measurement of consumer preferences. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

238. HUTCHINSON John (25/10/2006)
The size distribution and growth of firms in transition countries. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

239. RENDERS Annelies (26/10/2006)
Corporate governance in Europe: The relation with accounting standards choice, performance and benefits of control. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

240. DE WINNE Sophie (30/10/2006)
Exploring terra incognita: human resource management and firm performance in small and medium-sized businesses. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

241. KADITI Eleni (10/11/2006)
Foreign direct investments in transition economies. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

242. ANDRIES Petra (17/11/2006)
Technology-based ventures in emerging industries: the quest for a viable business model. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

243. BOUTE Robert (04/12/2006)
The impact of replenishment rules with endogenous lead times on supply chain performance. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

244. MAES Johan (20/12/2006)
Corporate entrepreneurship: an integrative analysis of a resource-based model. Evidence from Flemish enterprises. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

245. GOOSSENS Dries (20/12/2006)
Exact methods for combinatorial auctions. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

246. GOETHALS Frank (22/12/2006)
Classifying and assessing extended enterprise integration approaches. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

247. VAN DE VONDER Stijn (22/12/2006)
Proactive-reactive procedures for robust project scheduling. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2006.

248. SAVEYN Bert (27/02/2007)
Environmental policy in a federal state. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

249. CLEEREN Kathleen (13/03/2007)
Essays on competitive structure and product-harm crises. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

250. THUYSBAERT Bram (27/04/2007)
Econometric essays on the measurement of poverty. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

251. DE BACKER Manu (07/05/2007)
The Use of Petri Net Theory for Business Process Verification. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

252. MILLET Kobe (15/05/2007)
Prenatal testosterone, personality, and economic behavior. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

253. HUYSMANS Johan (13/06/2007)
Comprehensible predictive models: New methods and insights. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

254. FRANCKEN Nathalie (26/06/2007)
Mass Media, Government Policies and Economic Development: Evidence from Madagascar. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

255. SCHOUBBEN Frederiek (02/07/2007)
The impact of a stock listing on the determinants of firm performance and investment policy. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

256. DELHAYE Eef (04/07/2007)
Economic analysis of traffic safety. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

257. VAN ACHTER Mark (06/07/2007)
Essays on the market microstructure of financial markets. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

258. GOUKENS Caroline (20/08/2007)
Desire for variety: understanding consumers' preferences for variety seeking. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

259. KELCHTERMANS Stijn (12/09/2007)
In pursuit of excellence: essays on the organization of higher education and research. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

260. HUSSINGER Katrin (14/09/2007)
Essays on internationalization, innovation and firm performance. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

261. CUMPS Bjorn (04/10/2007)
Business-ICT alignment and determinants. Leuven, K.U. Leuven, Faculteit Economische en Toegepaste Economische Wetenschappen, 2007.

262. LYRIO Marco (02/11/2007)
Modeling the yield curve with macro factors. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2007.

263. VANPEE Rosanne (16/11/2007)
Home bias and the implicit costs of investing abroad. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2007.

264. LAMBRECHTS Olivier (27/11/2007)
Robust project scheduling subject to resource breakdowns. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2007.

265. DE ROCK Bram (03/12/2007)
Collective choice behaviour: non parametric characterization. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2007.

266. MARTENS David (08/01/2008)
Building acceptable classification models for financial engineering applications. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

267. VAN KERCKHOVEN Johan (17/01/2008)
Predictive modelling: variable selection and classification efficiencies. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

268. CIAIAN Pavel (12/02/2008)
Land, EU accession and market imperfections. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

269. TRUYTS Tom (27/02/2008)
Diamonds are a girl's best friend: five essays on the economics of social status. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

270. LEWIS Vivien (17/03/2008)
Applications in dynamic general equilibrium macroeconomics. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

271. CAPPELLEN Tineke (04/04/2008)
Worldwide coordination in a transnational environment: An inquiry into the work and careers of global managers. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

272. RODRIGUEZ Victor (18/04/2008)
Material transfer agreements: research agenda choice, co-publication activity and visibility in biotechnology. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

273. QUAN Qi (14/04/2008)
Privatization in China: Examining the endogeneity of the process and its implications for the performance of newly privatized firms. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

274. DELMOTTE Jeroen (30/04/2008)
Evaluating the HR function: Empirical studies on HRM architecture and HRM system strength. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

275. ORSINI Kristian (05/05/2008)
Making work pay: Insights from microsimulation and random utility models. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

276. HOUSSA Romain (13/05/2008)
Macroeconomic fluctuations in developing countries. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

277. SCHAUMANS Catherine (20/05/2008)
Entry, regulation and economic efficiency: essays on health professionals. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008.

278. CRABBE Karen (21/05/2008)
Essays on corporate tax competition in Europe. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

279. GELPER Sarah (30/05/2008)
Economic time series analysis: Granger causality and robustness. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

280. VAN HOVE Jan (20/06/2008)
The impact of technological innovation and spillovers on the pattern and direction of international trade. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

281. DE VILLE DE GOYET Cédric (04/07/2008)
Hedging with futures in agricultural commodity markets. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

282. FRANCK Tom (15/07/2008)
Capital structure and product market interactions: evidence from business start-ups and private firms. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

283. ILBAS Pelin (15/09/2008)
Optimal monetary policy design in dynamic macroeconomics. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008

284. GOEDERTIER Stijn (16/09/2008)
Declarative techniques for modeling and mining business processes. Leuven, K.U. Leuven, Faculteit Economie en Bedrijfswetenschappen, 2008