

2017

Data-driven conceptual modeling: how some knowledge drivers for the enterprise might be mined from enterprise data

Metta Santiputri
University of Wollongong

Follow this and additional works at: <https://ro.uow.edu.au/theses1>

University of Wollongong

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material.

Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Santiputri, Metta, Data-driven conceptual modeling: how some knowledge drivers for the enterprise might be mined from enterprise data, Doctor of Philosophy thesis, School of Computing and Information Technology, University of Wollongong, 2017. <https://ro.uow.edu.au/theses1/234>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au



UNIVERSITY
OF WOLLONGONG
AUSTRALIA

**DATA-DRIVEN CONCEPTUAL MODELING:
HOW SOME KNOWLEDGE DRIVERS FOR
THE ENTERPRISE MIGHT BE MINED FROM
ENTERPRISE DATA**

A Dissertation Submitted in Fulfilment of
the Requirements for the Award of the Degree of

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Metta Santiputri

School of Computing and Information Technology
Faculty of Engineering and Information Sciences

2017

© Copyright 2017

by

Metta Santiputri

ALL RIGHTS RESERVED

CERTIFICATION

I, Metta Santiputri, declare that this dissertation, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Metta Santiputri
21 March 2017

Table of Contents

List of Tables	v
List of Figures/Illustrations	vi
Abstract	vii
Acknowledgements	ix
List of Publications	x
1 Introduction	1
1.1 Motivation	1
1.2 Research questions	8
1.3 Research contributions	10
1.4 Thesis structure	12
2 Background	14
2.1 Goal-oriented requirements modeling	15
2.1.1 KAOS framework	16
2.1.2 i* frameworks	18
2.2 Goal elicitation	22
2.3 Business process modeling	26
2.3.1 BPMN	28
2.3.2 Semantically annotated process model	30
2.3.3 Process mining	32
2.4 The ArchiMate modeling language	34
2.5 Sequential pattern and sequential rule mining	37
2.5.1 Frequent pattern mining	38
2.5.2 Sequential pattern mining	45
2.6 Mining of enterprise models	52
2.7 Datasets	62
2.7.1 Event logs	62
2.7.2 Process logs	64
2.7.3 Effect logs	64
2.7.4 Message logs	65
2.7.5 Noise	66
2.7.6 Synthetic datasets	67
2.8 Research gap	69

3 Mining task post-conditions: Automating the acquisition of process semantics	71
3.1 Introduction	72
3.2 Example	75
3.3 An event ontology	79
3.4 Mining post-conditions	81
3.5 Validation	85
3.6 Abductive repair	89
3.7 Evaluation	92
3.8 Related works	96
3.9 Summary	100
4 Requirement model extraction	102
4.1 Introduction	102
4.2 The Dependency Extraction (DE) technique	105
4.3 The Task-Dependency Correlation Extraction (TDCE) technique	108
4.4 Evaluation	113
4.4.1 Evaluation of DE technique	114
4.4.2 Evaluation of TDCE technique	116
4.4.3 Improving requirements quality: Evaluation	119
4.5 Related works	123
4.6 Summary	125
5 Towards data-driven enterprise architectures: Discovering correlations between the business and application layers in ArchiMate	127
5.1 Introduction	128
5.2 Data-driven enterprise architectures: A general approach	130
5.3 Event logs	135
5.4 Mapping between logs and layer components	136
5.5 Frequent closed sequential pattern	138
5.6 Generating joined log	139
5.7 Mining the sequence patterns	142
5.8 Evaluation	149
5.9 Related works	153
5.10 Summary	155
6 Goal orchestrations: Modelling and mining flexible business processes	157
6.1 Introduction	158
6.2 Goal orchestration models and semantics	159
6.3 Executing goal orchestrations	162
6.3.1 Goal Consistency	165
6.4 Mining Goal Orchestrations	166
6.5 Evaluation	169

6.5.1	Evaluation with synthetic process models	169
6.5.2	Evaluation with real-life dataset	170
6.6	Summary	178
7	Conclusion and future work	179
7.1	Conclusion	179
7.2	Limitation	182
7.3	Future work	182
	References	212

List of Tables

2.1	An excerpt of <i>Holiday Booking</i> process event log	63
2.2	An excerpt of <i>Holiday Booking</i> process log	64
2.3	An excerpt of <i>Holiday Booking</i> process effect log	65
2.4	An excerpt of <i>Holiday Booking</i> process message log	66
3.1	Records of patient’s treatment	77
3.2	Records of patient’s conditions	78
3.3	An example of joined ProcessEvent-StateTransitionEvent table	86
3.4	An example of cumulative joined ProcessEvent-StateTransitionEvent table	86
3.5	The recall and precision measures from the evaluation	94
3.6	Excerpt from the process event log provided by the user	97
3.7	Excerpt from the state transition event log provided by the user	97
4.1	Example of process log	109
4.2	Candidate generation	110
4.3	Controlled Environment $k_{interaction} = 10\%$ and $k_{message} = 10\%$	115
5.1	Mapping Between Logs and Layer Components	138
5.2	Frequent sequence vs. frequent closed sequence	139
5.3	Unique task setting	143
5.4	Concurrent task setting	144
5.5	Concurrent task setting	147
5.6	Concurrent task setting	148
5.7	A small section of the telephone repair event log	150
5.8	The business processes, application functions, and application compo- nents and their correlations	150
5.9	Log_4 Result	153
6.1	An example of event log with corresponding non-deterministic cumula- tive effect sequence	168
6.2	Evaluation result with synthetic data	171
6.3	Ticket examples	173
6.4	Event sequences identified in the log	174
6.5	Goal assertions for the goal model	174

6.6 Goal sequence for effect trace 175

List of Figures

1.1	Overview of the models and the data generated in an enterprise	8
1.2	Overview of this thesis	13
2.1	Possible goal model with its goals and constraints for the Meeting Scheduler case study [157, 56, 256, 255]	19
2.2	Possible SD and SR model for the Meeting Scheduler case study [281, 50]	23
2.3	BPMN main elements	29
2.4	Three main types of process mining [250]	33
2.5	Core concepts of the ArchiMate language [240]	35
2.6	Summary of the concepts of the ArchiMate language [240]	35
2.7	Relationships between Business Layer and Application Layer Elements [240]	36
2.8	ArchiSurance, an ArchiMate example, taken from [135]	37
3.1	Clinical process for treatment of juveniles with head injuries [191]	76
3.2	Precision measures with noise in the effect log	95
3.3	Precision measures after validation with length parameter 2 and 3	95
3.4	A semantic annotated BPMN process model for <i>Holiday Booking</i> process	96
4.1	An i* SR model for a meeting scheduler system (adapted from [281])	114
4.2	Precision relative to log size	117
4.3	Precision relative to min support	117
4.4	Model from user	119
4.5	Extracted model-1	120
4.6	Extracted model-2	121
5.1	Microsoft Windows Event Viewer, an instance of event log in the application layer	137
5.2	ArchiMate Business and Application Layer	154
6.1	Treatment for children sustaining head injury with low blood sugar level	164
6.2	Goal orchestrations for business process model in Figure 6.1	164
6.3	Ticket handling process	172
6.4	Goal orchestrations for ticket handling process	176

Abstract

As organizations perform their business, they analyze, design and manage a variety of processes represented in models with different scopes and scale of complexity. Specifying these processes requires a certain level of modeling competence. However, this condition does not seem to be balanced with adequate capability of the person(s) who are responsible for the task of defining and modeling an organization or enterprise operation.

On the other hand, an enterprise typically collects various records of all events occur during the operation of their processes. Records, such as the start and end of the tasks in a process instance, state transitions of objects impacted by the process execution, the message exchange during the process execution, etc., are maintained in enterprise repositories as various logs, such as event logs, process logs, effect logs, message logs, etc. Furthermore, the growth rate in the volume of these data generated by enterprise process execution has increased manifold in just a few years.

On top of these, models often considered as the dashboard view of an enterprise. Models represents an abstraction of the underlying reality of an enterprise. Models also served as the "knowledge driver" through which an enterprise can be managed. Data-driven extraction offers the capability to mine these knowledge drivers from enterprise data and leverage the mined models to establish the set of enterprise data that conforms with the desired behaviour.

This thesis aimed to generate models or knowledge drivers from enterprise data to enable some type of dashboard view of enterprise to provide support for analysts. The rationale for this has been started as the requirement to improve an existing process or to create a new process. It was also mentioned models can also serve as a collection of effectors through which an organization or an enterprise can be managed.

The enterprise data refer to above has been identified as process logs, effect logs, message logs, and invocation logs. The approach in this thesis is to mine these logs to generate process, requirement, and enterprise architecture models, and how goals get fulfilled based on collected operational data.

The above a research question has been formulated as "whether it is possible to derive the knowledge drivers from the enterprise data, which represent the running operation of the enterprise, or in other words, is it possible to use the available data in the enterprise repository to generate the knowledge drivers?".

In Chapter 2, review of literature that can provide the necessary background knowledge to explore the above research question has been presented. Chapter 3 presents

how process semantics can be mined. Chapter 4 suggest a way to extract a requirements model. The Chapter 5 presents a way to discover the underlying enterprise architecture and Chapter 6 presents a way to mine how goals get orchestrated. Overall finding have been discussed in Chapter 7 to derive some conclusions.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Prof. Aditya Ghose and Dr. Hoa Khanh Dam, for the immeasurable support and guidance throughout my study. Their patience, encouragement, and insights have helped me immensely during the time of research and writing of this thesis and all related publications. I would also like to thank all my colleagues in Decision Systems Lab (DSL) research group for their support, discussions and comments on my work.

I would like to thank my institution, State Polytechnic of Batam, Indonesia, for giving me the opportunity to pursue PhD education. I am also obliged to the Government of Indonesia who provides me with the scholarship which allow me to finish this study.

I am eternally grateful to my family, my husband, Uuf Brajawidagda, and our daughter, Annika, for their love and support during our time here. Without them, none of this would be possible. Also my deepest gratitude to my parents in Indonesia who always believe in me and support me unconditionally.

Lastly, I would like to express my heartfelt thanks to all of those who supported me during the completion of this thesis.

List of Publications

- Metta Santiputri, Aditya Ghose, and Hoa Khanh Dam. Mining Task Post-Conditions: Automating the Acquisition of Process Semantics. In Paul Johannesson, Mong Li Lee, Stephen W. Liddle, Andreas L. Opdahl, and Óscar Pastor López, editors, *Data & Knowledge Engineering Journal* volume 109, pages 112-125. DOI 10.1016/j.datak.2017.03.007.
- Metta Santiputri, Aditya Ghose, and Hoa Khanh Dam. Goal Orchestrations: Modelling and Mining Flexible Business Processes. In Mayr H., Guizzardi G., Ma H., Pastor O, editors, *Conceptual Modeling*, volume 10650 of *Lecture Notes in Computer Science*, pages 373-387. Springer Cham, October 2017. DOI 10.1007/978-3-319-69904-2.
- Metta Santiputri, Aditya Ghose, Hoa Khanh Dam, and Xiong Wen. Mining Process Task Post-Conditions. In Paul Johannesson, Mong Li Lee, Stephen W. Liddle, Andreas L. Opdahl, and Óscar Pastor López, editors, *Conceptual Modeling*, volume 9381 of *Lecture Notes in Computer Science*, pages 514-527. Springer Berlin Heidelberg, October 2015. DOI 10.1007/978-3-319-25264-3.
- Suman Roy, Metta Santiputri, and Aditya Ghose. Annotation and Mining for Effects of Processes. In Isabelle Comyn-Wattiau, Katsumi Tanaka, Il-Yeol Song, Shuichiro Yamamoto, and Motoshi Saeki, editors, *Conceptual Modeling*, volume 9974 of *Lecture Notes in Computer Science*, pages 302-311. Springer Berlin Heidelberg, November 2016. DOI 10.1007/978-3-319-46397-1.
- Aditya Ghose, Metta Santiputri, Ayu Saraswati, and Hoa Khanh Dam. Data-driven Requirements Modeling: Some Initial Results with i*. In Georg Grossmann and Motoshi Saeki, editors, *Proceedings of the Tenth Asia-Pacic Conference on Conceptual Modelling (APCCM 2014)*. *Conferences in Research Practises in Information Technology (CRPIT)* Volume 154, pages 55-64. Auckland, New Zealand, January 2014.

Chapter 1

Introduction

In any organization, an assortment of models with variety of scopes and complexity are executed each day. The volume of data generating from these processes give an opportunity to mine many useful information to gain insight of the organization. Among many possible information that can be mined from the enterprise data are the models that describes the intended or correct behaviour of the organization.

This observation gives the motivation behind this thesis. More detail on this motivation presented in Section 1.1. This possibility to derive the knowledge from available data leads to the research question in Section 1.2. To address these questions, the contributions of this thesis is provided in Section 1.3. The structure of the remainder of this thesis is described in Section 1.4.

1.1 Motivation

The requirement to improve an existing process or to create a new process was one of the main push behind business process work in most organizations. Leading organizations invested more than \$1 million on business process analysis, process management, monitoring, redesign and improvement, some even reportedly spent over \$10 millions

in the year 2015 alone [113].

As organizations develop and deliver products and/or services, they build a number of functions and operations necessary to improve or create a new line of business. The range of functions and operations are varied and can be modeled with different scopes and scales of complexity [217]. This variety of models may also covers all different types of models from different phases in system and/or product development within the organization [217]. Several examples of models developed in an enterprise are listed below.

- Process models

Process models represent the flow of activities in a particular business or organizational unit in order to achieve some goals. A process model is a combination of activities, decisions, and sequence flow. To better understand the goals that the process aim to achieve, a process model is complemented with a declarative specification to provide process semantics. By adding semantic descriptions to the process model, analysts can perform reasoning over the annotated activities [88]. To construct semantically annotated business process models, analysts annotate activities in the model with descriptions of the changes that occur as a result or outcome of the activity execution (or *effect*) [123, 88, 184]. For each activity or event, state semantics can be annotated using a formal language. Though a lot of interest has been put to define the semantics of process modeling (such as [138, 149, 66, 27]), nevertheless to perform semantic annotation of process models is a complex and time-consuming task (more so with a complex process model), especially for non-expert modeler. We describe the process models using BPMN notations, which is explained further in Section 2.3 of Chapter 2.

Task outcome, represented as effects, and goal realizations can be further combined to represent the goals achieved during process enactment. Representing a

process model as a coordination of goals (referred as *goal orchestration*) allows the process to be executed in flexible and context sensitive ways. As the actors will be able to realize a goal in multiple different ways, it will enable a flexible process management.

- Requirement models

Requirement models define precise specifications of software behavior based on the real-world goals for, functions of, and constraints on software systems. They provide the basis for analyzing requirements, validating that they are indeed what stakeholders want, defining what designers have to build, and verifying that they have done so correctly upon delivery. One of the most prominent frameworks in goal-oriented requirement modeling is the i* framework. This framework is a framework for early-phase requirements modeling. It describes the requirements as a set of dependencies between actors and the tasks from each actor that contributes to that particular dependency. More details on the requirement modeling frameworks will be presented in Section 2.1 of Chapter 2. However, requirements acquisition/elicitation has been a well-known challenges (recognize as an instance of the *knowledge acquisition bottleneck*) which mostly originated from the difficulty to articulate users' requirements. This phenomenon leads to the requirements elicitation process becoming a time consuming human task of considerable complexity .

- Enterprise architecture

Enterprise architecture describes an overall view of an enterprise, including the structure and behavior of the system. One of the enterprise architecture modeling framework that has gained an increasing popularity is ArchiMate, first proposed by the Open Group in 2009. It describes the enterprise architecture in three different layers, i.e., business, application and technology layers, and connections

between elements from different layers to express the structure of the system from an abstract view to the implementation level. Section 2.4 of Chapter 2 presents an introduction to the ArchiMate modeling language.

However, a costly investment, both in effort and time, is required during data acquisition to describe the enterprise setting, especially in a complex and diverse environment, for instance when the environment covers both virtual and physical applications and infrastructures in several different locations [12, 68, 140]. Furthermore, in the event of rapid change, EA is difficult to maintain [12].

As a matter of fact, the requirement to improve an existing process or to create a new process was one of the main push behind business process work in most organizations. As process models were considered as a major value chain in these organizations, specific managers were assigned to specific processes, major or otherwise. However, only a fraction of these managers was frequently trained to analyze, design and manage business processes [113]. A survey conducted by BPTrends in 2016 [113] reported that their respondents (from a variety of industries, functions, organization's size, and locations) indicated that 99% of their organizations invested in business process management and the process models defined for 97% of the major value chains in the organization. Moreover, 86% of the respondents from the same survey also indicated that their organization has process managers who are responsible for process. However, almost 20% of these managers were never been trained to analyze, design and manage business processes. The survey also inquired about the organization willingness to hire outside consultants to help with their business process management. The answers given were 42% stated that they would hire consultants to develop an enterprise process architecture, 52% indicated that the consultants would be responsible for defining the relationship between strategy and process, and 49% would hire consultants to coordinate and manage the business process management projects

and programs.

From these results alone, it can be concluded that the business process management considered as highly crucial in an organization, however it does not follow that their manager were equipped with the capacity to analyze, design and manage business processes and that more organizations were willing to hire outside consultants to make up for the lack of this competency.

A large body of research have addressed this challenge of modeling as a complex, time-consuming process where the required competence of the analysts is not met. Most studies tackled this challenge by providing guidelines for analysts to build comprehensive models, such as SEQUAL [150], GoM [26], ISO-9126 compliant framework [182], 7PMG [177] and 4EM [217]. The emergence of these studies highlighted the fact that describing or defining the processes that need to be supported become the main predicament in system development, instead of the actual implementation or programming aspect of the system. However, despite a number of adequate guidance towards modeling, they typically required a certain level of modeling competence. What missing from these guides are elicitation approaches that can be adopted to help analysts to start with model building, especially for novices and non-expert modelers. 4EM [217] does include a selection of elicitation approaches, but they rely heavily on the approaches that have been applied mainly in requirements elicitations, such as interviews [4, 126, 93, 175, 209, 31], observation [21, 209, 32, 272], and workshops [274, 166, 96].

Many medium to large organizations run hundreds or even thousands of business process models in their operation. Using a miscellaneous of logging tools, these process executions can generate a multitude of data. Reports has shown that the number of data volume collected during the system operation has increased manyfold in just a few years [28, 83, 104, 132, 270]. Not just in volume, the data also comprises of various

types, such as:

- Event logs

Event logs have been utilized as a basis for process analysis in many different settings, most notably in the process mining [249, 251, 105, 267]. Essentially, event logs record any event occurrences during process execution. Each record is *time-stamped* and refers to an *event* of a particular *process instance*.

During a process execution, the events that occur can be viewed in general terms as being of two types: (1) events that correspond to the execution of the process tasks, i.e., start or end of a process task, and (2) events that correspond to the impact of a process task execution, i.e., the state changes in the objects as the outcome of the process execution. In general, the information enclosed in a record in the event logs usually consists of: (a) *instance/case identifier*, which describe the process instance an event is related to, (b) *event descriptor*, which contains the event description, such as a task ID or a state transition, and (c) a *time-stamp*. Other information, such as the originator or the person responsible for the event, can also be found though not as regular as the former.

- Process logs

We refer to the subset of event logs that record the events which signify the start or end of the process tasks execution as *process logs*. We make distinction of this type of logs with the next type of logs (i.e., effect logs) to emphasize the difference between the two types of events. A variety of business process management tools with event logging capability can be utilized to generate process logs.

- Effect logs

Effect log is also a subset of event logs. These logs record events that correspond to the outcome or impact of the process execution. The name *effect log* is preferred in view that the state transitions being recorded are considered as the

impact or *effect* of the process. In many settings, these logs can be generated by the same tools as process logs, however in other settings, an object state monitor has to be installed to specifically obtain them.

- Message logs

Message logs record all form of communications in an organization, such as email or any other message exchange applications. In the enterprise context, these type of logs are frequently maintained and any process logging tool can be used to generate them. Because the message exchange occurs between actors in an organization, these logs usually documented in natural language and require NLP techniques to process them.

We acknowledge that there are other sources of data in an organization repository, for example enterprise level documents such as standard operating procedures of the organization; other process related logs such as exceptions/error logs, or provision/resourcing logs; or any other data sources. However, they are outside of the scope of this thesis.

The models developed by an enterprise and the data generated by the process execution is summarize in Figure 1.1.

System models often viewed as an abstract collections of *sensors* through which one can observe a complex underlying reality. On the other hand, models can also serve as a collection of *effectors* through which an organization or an enterprise can be managed. Models defines the “to-be” or desired enterprise operational, then leverage the mined models to determine specific organization data/repositories one would require such that these, if mines using the same approach, would yield the desired operation. Using these understanding, these models are referred as a collection of “*knowledge driver*”. One approach to achieve these is by performing data-driven extraction, which can enable a type of dashboard view of an organization where one can

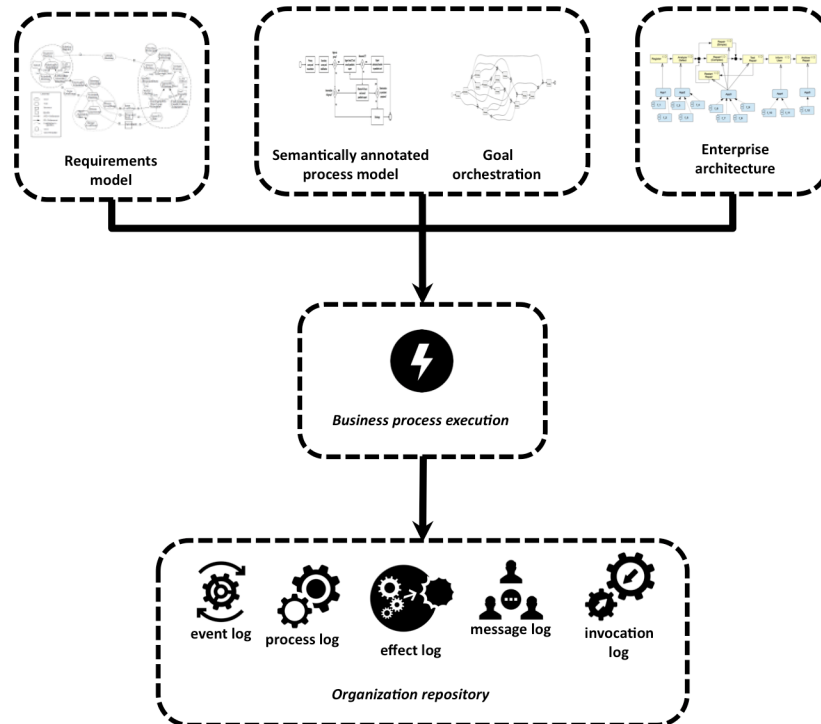


Figure 1.1: Overview of the models and the data generated in an enterprise

mine these knowledge drivers from enterprise data as an abstraction of the underlying reality and furthermore, the mined models can be leveraged to establish the set of enterprise data that conforms with the desired behaviour.

1.2 Research questions

Given the ability of system models as knowledge drivers to give a dashboard view of an enterprise and data-driven extraction approach as one possible mean to realize it, gives rise to the question *whether it is possible to derive the knowledge drivers from the enterprise data, which represent the running operation of the enterprise, or in other words, is it possible to use the available data in the enterprise repository to generate the knowledge drivers?*

In this thesis, we turn to the machine learning and data mining fields of research and leverage methods in both areas to mine the knowledge drivers. Relating to the

models and the data in Figure 1.1, the research questions are specifically identified as follow:

1. Mining process semantics

- (a) How the context-independent effects/outcomes of each task in the process model are mined from process logs and effect logs?
- (b) How the mined effects are validated?
- (c) What can be done when mined effects are found to be unsound or incomplete according to the validation result?

2. Requirement model extraction

- (a) How the inter-actor dependencies in the requirement model are mined from message logs?
- (b) How the tasks/goals associated with each (mined) dependency are mined from process logs and message logs?
- (c) How the methods are evaluated in regards to their effectiveness?

3. Enterprise architecture mining

- (a) What settings can be identified in relation to concurrent task execution in an enterprise?
- (b) How the correlation of the business and application layers in an ArchiMate model are mined from event logs?
- (c) How the methods are evaluated in regards to their effectiveness?

4. Goal orchestrations

- (a) How the business process model is represented as a coordination of goals?

- (b) How a goal orchestration model is extracted from event logs (specifically effect logs)?

1.3 Research contributions

The main contribution of this thesis is *the formulation of approaches to mine some knowledge drivers from data generated by the operational process execution by leveraging methods from the machine learning and data mining fields of research*. Support for the analysts are provided by generating “first-cut” models, which then can be adjusted to the analysts’ intentions and used as the base to improve the process. More specifically, the main contribution can be broken down into a number of points as outlined below.

- **Development of a method for mining process semantics from process logs and effect logs**

To describe the process semantics, the formalization of the task effects and the effect accumulation was adopted from previous works in [123, 88, 184, 86]. By leveraging sequential rule mining method, the context-independent effects were mined from process logs and effect logs. This contribution corresponds to research questions number 1(a). The available data, i.e., process logs and effect logs, were also leveraged to validate the mined effects, or in other words, to determine if the mined effects predict the state transitions seen in the data. This contribution relates to research questions number 1(b). The guidance to modify the mined effects were provided by formulating the problem as abductive problem. By formulating it as abductive problem, it can be determined what effects to be *augmented* (in the case where the mined effects are found to be incomplete) or *contracted* (in the case where the mined effects are found to be unsound). This contribution corresponds to research questions number 1(c).

- **Development of a method for mining inter-actor dependencies in a requirement model from process logs and message logs**

The i^* framework describes the requirements as a set of dependencies between actors in an organization. The approach mined the inter-actor dependencies and the tasks involved in a particular dependency from message logs and process logs. It mainly mined a domain model, or a model of the “as-is”, and does not intent to mine requirements or goals in the minds of stakeholders that have no manifestation in data. The approach leveraged the sequential pattern mining method to mine patterns in the logs that signify the dependency. This contribution relates to the research questions number 2(a) and 2(b). A proof-of-concept evaluation was provided using two settings, i.e., using synthetic data and by leveraging expert user, to determine the effectiveness of the method. This contribution corresponds to the research questions number 2(c).

- **Development of a method for mining the relationships between layers in an enterprise architecture from event logs**

Within an enterprise, four different settings were defined, i.e., complete timestamp-unique task setting, complete timestamp-concurrent task setting, partial timestamp-unique task setting, and partial timestamp-concurrent task setting, which relates to research question number 3(a). For each setting, the approach supported the enterprise architecture modeling by mining the relationship between business layer and application layer. It determine the relationships automatically by leveraging the event logs using the frequent closed sequential pattern mining method. This contribution corresponds to research question number 3(b). To evaluate the method, a proof-of-concept evaluation was provided. This contribution relates to research question number 3(c).

- **Development of a process model representation as a coordination of goals**

A goal-oriented representation of a business process as the sequence of goals achieved, called a goal orchestration model, was provided. Goal model, which contains available goals, provides the vocabulary for the effects which represents the task outcome. This contribution relates to research question number 4(a).

- **Development of a method for mining a goal orchestration model from event logs**

The sequence of effects, recorded in event logs, combined with a goal model are mined to extract a goal orchestration model. The approach adopted a similar method with the alpha algorithm to generate the model. If the goal model is specific to an actor or a process instance, then the goals will be recognized and mined specific to the process or actor in question. This contribution corresponds to research questions number 4(b).

The contributions of this thesis is presented as a framework in Figure 1.2.

1.4 Thesis structure

This chapter is concluded by summarizing the structure of this thesis.

Chapter 2 provides the background concepts that we use throughout this theses, including business process management, goal modeling, and sequential pattern mining and sequential rule mining.

Chapter 3 introduces the method to mine the effects or post-conditions of a process task. The definition of semantic effect annotation and the validation technique to exercise these mined effects are also introduced.

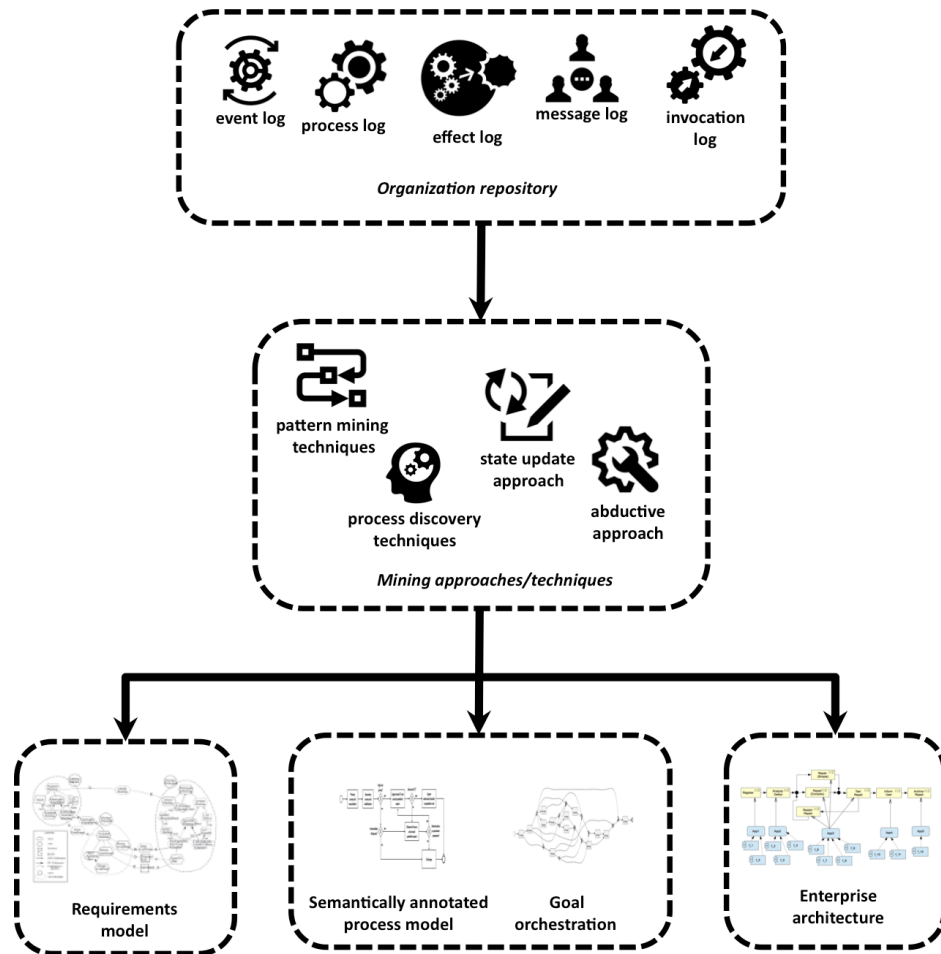


Figure 1.2: Overview of this thesis

Chapter 4 introduces an approach to mine the dependency in the i^* model.

Chapter 5 explains a method to mine the correlations between multiple layers in the enterprise architecture model, namely ArchiMate model.

Chapter 6 presents the concept of goal orchestration, a process model representation through sequence of goal satisfaction. In addition, it also provides the method to mine them using similar approach to the effect mining.

Chapter 7 finally concludes the thesis with some final remarks.

Chapter 2

Background

This thesis aims to explore the possibility to derive the knowledge drivers from the enterprise data. The first model intended to be mined is the requirement models, given the history that goal elicitation is considered to be a difficult process. Another possibility presented is to mine the process semantic or semantic annotated process model. Although there is a large volume of languages and frameworks in business process modeling, BPMN maintain as the most prominent framework to describe the process. An example of a field of research similar with this thesis is process mining. It mine the process model from event logs and the process model use BPMN notation, however it does not describe the semantic of the process. Architecture model is another possible model to be mined from available data. It describes an enterprise as layers of different abstractions. ArchiMate is the standard notation of enterprise architecture.

In this thesis, the mining of different models are performed using methods in sequential pattern and sequential rule mining. These methods takes an important role in this work considering that the analysis are based on the patterns or rules that are discovered from the data source.

Section 2.1 and Section 2.2 examines the goal-oriented requirement engineering as the basic approach in the requirement modeling that will be mined and the various

method for goal elicitation that has been employed to identify goals. These sections gives an understanding of the goal models and the methods to construct these models. Section 2.3 presents the introduction to BPMN and semantic annotation to process model in BPMN. Section 2.4 introduces the Archimate modeling language that was applied to represent an enterprise architecture. In section 2.5 sequential pattern mining and sequential rule mining are discussed and different algorithms in both areas are overviewed to showcase various alternatives of mining methods.

2.1 Goal-oriented requirements modeling

The key steps of the Requirements Engineering (RE) process are domain analysis, elicitation, negotiation and agreement, specification, specification analysis, documentation, and evolution [252, 189, 230, 148, 204]. They are intertwined and may span the entire life cycle of software development. The requirements for a software system maybe spread across the system, including problem owners, stakeholders, documentation, and other existing systems, which requires an elicitation process. The basic techniques for requirements elicitation includes many different methods such as interviews [4, 126, 93, 175, 209, 31], observation [21, 209, 32, 272], scenarios [236, 1, 141], workshops [274, 166, 96], focus groups [151], protocols [93, 187, 175], prototypes [229], and models [188, 24, 25, 216].

Requirements elicitation was started with the questions such as “what the system requirements are”. However, the RE research has been move towards addressing questions like “why the software is needed” and “why the design is justified” [252, 282]. By exploring these “why” questions, requirements analysts are able to model the intentions and purpose of the systems being designed. These research projects marked the recognition of goal significance in the RE process and formed the branch of RE research known as *Goal-Oriented Requirements Engineering* (GORE)

[254, 252, 253, 277, 210, 185, 142]. In this section we will explore further about KAOS methodology [54, 55, 56, 157] and i* framework [279, 281, 282], two most popular goal-oriented approaches [128].

2.1.1 KAOS framework

The KAOS methodology (stands for Knowledge Acquisition in autOMated Specification [55] or Keep All Objects Satisfied [257]) was proposed by Dardenne, van Lamswerde and Fickas [54, 55]. It was the first widespread approach to goal-oriented requirements engineering [79]. The methodology was aimed at supporting the requirements elicitation process [54, 55, 56, 157].

The KAOS methodology consists of three components, i.e., (a) *the specification language* that provides constructs to capture various types of concepts, (b) *the elaboration method* to elaborate high-level goals into KAOS specification systematically, and (c) *the meta-level knowledge* for guidance during the elaboration processes [157].

A *goal* is defined as “non-operational objective to be achieved by the composite system” [55, 56] or “an objective the composite system should meet, usually through the cooperation of multiple agents” [54, 57, 58, 157]. Each goal has a name, a natural language definition which describes the set of temporal sequence of states of the system that satisfying the goal, and an optional formal definition in temporal logic formula which describes the same set of temporal sequence of states [157].

Goals can be linked to a set of subgoals using *AND/OR-refinement* links. In an AND-refinement, the sufficient condition to satisfy a goal is by satisfying all of its subgoals while in an OR-refinement, the sufficient condition to satisfy a goal is by satisfying one of its subgoals [157, 56, 57, 58]. The goal refinement structure can be described using an AND/OR directed acyclic graph, denoted as *goal model* [30, 255, 157]. Figure 2.1 illustrate an example of a goal model for the Meeting Scheduler case

study adapted from [157, 56, 256, 255]. Goal refinement ends up when terminal goals are reached and these terminal goals are assignable to individual agents. Goals can be considered as requirements or assumptions dependent on whether they are assigned to the software agent or to an environment agent, respectively [54, 55, 157, 255].

Based on the concept being specified, the generic construct of a KAOS structure can be instantiated to specific types of links and assertion languages [54, 55, 157, 255]. For example, consider the specification for `ParticipantsConstraintsKnown` goal in Figure 2.1. The declaration part introduces a goal named `ParticipantsConstraintsKnown` with the required property that should eventually hold (denoted with `Achieve` verb) and refers to objects `Meeting`, `Participant`, `Scheduler`. The specification also contains the links to its parent goal, `MeetingRequestSatisfied`, and its refinement into subgoals, `ParticipantsConstraintsRequested`, `ParticipantsConstraintsProvided`, `ParticipantsAgendaUpToDate`, and `ParticipantsConstraintsKnownFromAgenda`. The goal is described using both informal statement and formal assertion. The formal assertion is written in a real-time temporal logic or using the bounded version of the temporal operators [30, 255, 157]. For example, in Figure 2.1, the operator $\diamond_{\leq d}$ refers to some time in the future within some deadline d .

Based on their temporal behavior, goals can be classified into four patterns [157, 56, 256], namely:

- *Achieve goals*, i.e., goals requiring that some property eventually holds, corresponds to temporal formula $P \Rightarrow \diamond Q$
- *Cease goals*, i.e., goals requiring that some property eventually stops to hold, corresponds to temporal formula $P \Rightarrow \neg \diamond Q$
- *Maintain goals*, i.e., goals requiring that some property always holds, corresponds to temporal formula $P \Rightarrow Q$
- *Avoid goals*, i.e., goals requiring that some property never holds, corresponds to temporal formula $P \Rightarrow \neg Q$

Goals are also classified according to the type of requirement they express with respect to the agents concerned (e.g., *SatisfactionGoals* are goals concerned with satisfying agent requests; *InformationGoal* are goals concerned with making an agent informed about object states; *SafetyGoal* are goals concerned with avoiding hazardous states) [157, 56, 256].

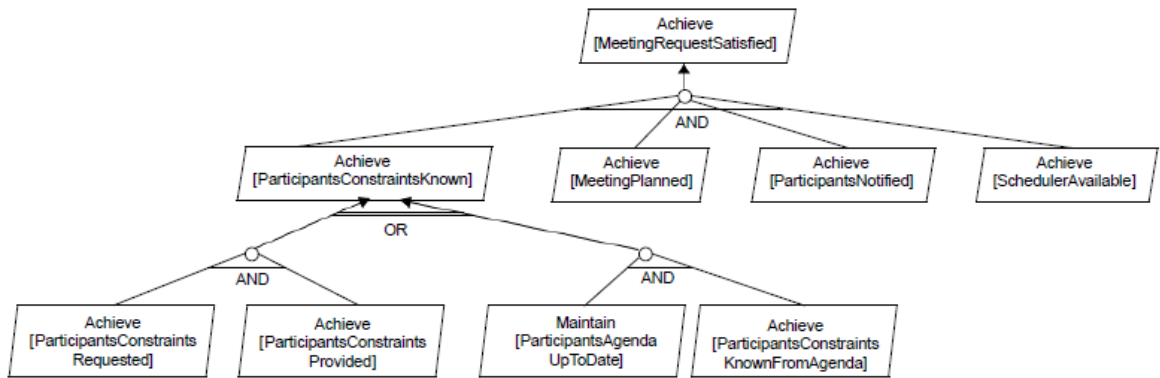
2.1.2 i* frameworks

The i* framework [279, 278, 282, 280, 281, 283] was proposed by Eric Yu. It was developed for modeling and reasoning about organizational environments and the stakeholders including their objectives and their relationships. The main concepts of i* were introduced in [282] and finalized in [280]. Since then, the version of i* has evolved and was updated into a wiki document, the iStar-wiki¹. The latest standard, the iStar 2.0 Language Guide, is published on June 2016 [50]. This standard contains a core language (focusing on concepts and relationships) to be spread for research, education and technology transfer purposes. According to this latest standard, “iStar” is preferred instead of “i*” to allow better indexing through search engines. In the remainder of this section, all the definitions refer to the iStar 2.0 Language Guide.

The main notion of the i* framework is the *intentional actors* and *intentional dependency*. It models the actors’ goals, the means available to achieve these goals, and how it depends to other actors to achieve their goals. Actors depend on each other for goals to be achieved, tasks to be performed, resources to be furnished and performance measures to be optimized. Goals that are difficult or impossible for an actor, may be achieved by means of dependency with other actors. However, the downside of this dependency is that if the depended-on actors do not deliver, it becomes a vulnerability.

Actors are defined as “active, autonomous entities that aim at achieving their goals

¹<http://istarwiki.org>



Goal *Achieve* [MeetingRequestSatisfied]

Concerns Meeting, Initiator, Participant

RefinedTo ParticipantsConstraintsKnown, MeetingPlanned, ParticipantsNotified

InformalDef Every meeting request should be satisfied within some deadline associated with the request. Satisfying a request means proposing some best meeting date/location to the intended participants that fit their constraints, or notifying them that no solution can be found with those constraints.

FormalDef

(\forall r: Initiator, m: Meeting, p: Participant)

Requesting (r,m) \wedge Feasible (m) $\Rightarrow \diamond_{\leq d}$ Scheduled (m)

\wedge Invited (p,m) $\Rightarrow \diamond_{\leq d}$ Knows (p,m)

Goal *Achieve* [ParticipantsConstraintsKnown]

Concerns Meeting, Participant, Scheduler

Refines MeetingRequestSatisfied

RefinedTo ParticipantsConstraintsRequested, ParticipantsConstraintsProvided, ParticipantsAgendaUpToDate, ParticipantsConstraintsKnownFromAgenda

InformalDef A meeting scheduler should know the constraints of the various participants invited to the meeting within some deadline d after invitation.

FormalDef

(\forall m: Meeting, p: Participant, s: Scheduler)

Invited (p,m) \wedge Scheduling (s,m) $\Rightarrow \diamond_{\leq d}$ Knows (s,p.Constraints)

WeakConstraint *Maintain* [AgendaUpToDate]

InstanceOf ConsistencyConstraint

UnderResponsibilityOf Participant

FormalDef

(\forall p: Participant, tp: TimeInterval)

Agenda (p,-) $\wedge \neg$ Free (p,tp) \Leftrightarrow tp \in Agenda[p,-].BusyPeriods

StrongConstraint *Achieve* [BestSchedule]

Operationalizes *Achieve* [MeetingPlannedWithNegotiation], *Maximize* [ScheduleConvenience], *Minimize* [DeadEnds]

FormalDef

(\forall r: Initiator, m: Meeting, s: Scheduler)

Requesting (r,m) \wedge Scheduling (s,m)

$\Rightarrow \diamond_{\leq d}$ [Feasible (m) \Rightarrow Scheduled (m) \wedge Preferred (m)

\wedge NearlyFeasible (m) \Rightarrow ScheduledByNegotiation (m)

$\wedge \neg$ Feasible (m) $\wedge \neg$ NearlyFeasible (m) \Rightarrow DeadEnd (m)]

Figure 2.1: Possible goal model with its goals and constraints for the Meeting Scheduler case study [157, 56, 256, 255]

by exercising their know-how, in collaboration with other actors” [50]. Actors can be human, organizations, technical systems (hardware, software), or any combination of them. Actor can be used in the model without specialization (i.e., generic *actor*) or can be distinguished into two types, i.e., *Role* which represents an abstract characterization of an actor’s behavior within some context or domain, or *Agent* which represents an actor with concrete, physical manifestations. The relationships between actors are described using two types of *actor links*, i.e., *is-a* which represents the concept of generalization/specialization, and *participates-in* which represents any kind of association, other than generalization/specialization, between two actors.

Actor’s intention is identified by the *intentional elements* within their *boundary*. Four types of elements are defined, namely:

- *Goal*: a state of affairs that the actor wants to achieve and that has clear-cut criteria of achievement;
- *Quality*: an attribute for which an actor desires some level of achievement;
- *Task*: represents actions that an actor wants to be executed, usually with the purpose of achieving some goal;
- *Resource*: a physical or informational entity that the actor requires in order to perform a task.

A *dependency* represents a relationship between two actors with five arguments:

- *depender* is the actor that depends for something (the dependum) to be provided;
- *dependerElmt* is the intentional element within the dependers actor boundary where the dependency starts from, which explains why the dependency exists;
- *dependum* is an intentional element that is the object of the dependency;
- *dependee* is the actor that should provide the dependum;
- *dependeeElmt* is the intentional element that explains how the dependee intends to provide the dependum.

Based on the type of dependum, the dependencies can be classified into four types:

- *Goal dependency*: the dependee is expected to achieve the goal, and is free to choose how;
- *Quality dependency*: the dependee is expected to sufficiently satisfy the quality, and is free to choose how;
- *Task dependency*: the dependee is expected to execute the task in a prescribed way;
- *Resource dependency*: the dependee is expected to make the resource available to the depender.

Intentional elements can be related using four types of links:

- *Refinement* is an n-ary hierarchical relationship between goals and tasks. In *AND-refinement*, all the children must be fulfilled to fulfill the parent, while in *OR-refinement*, at least one child must be fulfilled to make the parent fulfilled.
- *NeededBy* relates a task with a resource to indicate that the actor needs the resource to execute the task.
- *Contribution* represents the effects of intentional elements on qualities. Positive effects will result in qualities being *fulfilled* or *satisfied*, on the other hand, negative effects will result in qualities being *denied*. Based on the effects, the Contribution link can be categorized in four types, i.e., *Make* (strong positive effects), *Help* (weak positive effect), *Hurt* (weak negative effect), and *Break* (strong negative effect).
- *Qualification* links a quality to its subject: a task, goal, or resource to express a desired quality over the execution of a task, the achievement of a goal, or the provision of the resource.

The iStar 2.0 framework introduces of three model views:

Strategic Dependency (SD) model is used to describe dependency relationships between actors within the boundary of an organization. It provides a set of

concepts for modeling processes in terms of the intentional dependency among actors.

Strategic Rationale (SR) model is used to describe the interests and concerns of actors in the model and how they can be addressed or impacted by different system configurations. It provides an intentional description of processes in terms of process elements and the rationales behind them.

Hybrid SD/SR is a combination of SD/SR views that focus on the strategic rationale of a particular set of actors.

Figure 2.2 shows an example of SD and SR model for Meeting Scheduler case study adapted from [281] with iStar 2.0 notations [50].

2.2 Goal elicitation

Goals are often difficult to identify [256, 17, 115, 212, 253, 155]. In some cases, goals are readily available and explicitly stated in the preliminary documents or other materials. However, in most cases, goals are often implicit. Therefore the process of goal elicitation is needed in order to identify goals.

Goal models have been transformed to and from a broad range of languages and artifacts. The technique to transform from business artifacts to goal models is beginning to rise in the last ten years, while from software artifacts to goal models is remained relatively low [128]. Goal models can be elicited from: business artifacts, such as business process [60, 80, 81], architecture enterprise [235], and scenarios [276, 211, 212, 258, 52, 53]; software artifacts, such as features [22], web services [137], and code [284, 264]; UML modeling language, such as use case [15]; non-UML modeling language, such as EPC [35, 186], BPMN [147, 61], CPMM [293], BGR [34], BPCM [80], DIS [101], HAM [101], Nomos [225], Future Wheels [202],

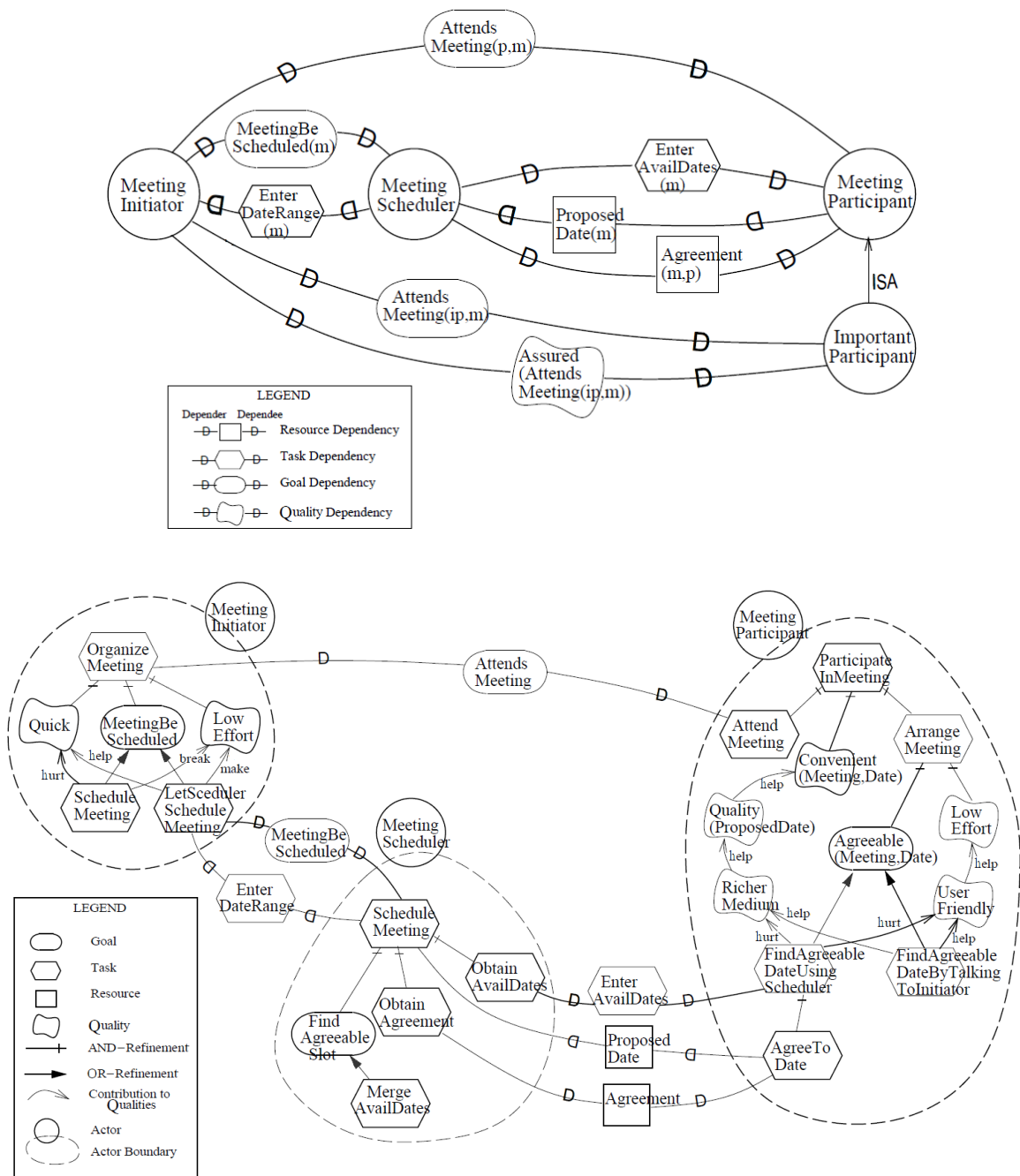


Figure 2.2: Possible SD and SR model for the Meeting Scheduler case study [281, 50]

and UCM [84]; requirements, such as other goal models [202], NFR [42], and natural language [14, 16, 171, 130, 134, 143, 156, 206, 137, 133, 137]; and architecture [235].

De la Vara, Sánchez and Pastor [60] and Gao and Krogstie [81, 80] investigate the correlation between goal models and business process models and define a set of guidelines to derive a goal model from a business process model. The guidelines maps elements and patterns in the business process model to elements in the goal model. Sunkle, Kulkarni and Roychoudhury [235] also provides the mapping from and to goals, but instead of business process model, they use the core elements of enterprise architecture and the mapping is performed through i* intentional metamodel. Another business artifact in the form of scenario also used in goal elicitation. The *CREWS-L'Ecritoire* approach by Rolland et al. [211, 212] used goal-scenario coupling (called “requirement chunk”) as a mean to discover goals. They performed AND (composition) and OR (alternative) operations to the goal-scenario pair to elaborate goals and build the goal model. Another method by Yang, Prasanna and King [276] called goal-directed information analysis (GDIA) translates scenarios to goal structure using seven clearly defined and repeatable steps, including task analysis. Formal specification of goals also obtained from scenarios by Van Lamsweerde et al. [258] and Damas [52, 53] using inductive inference.

Another approach to discover goals is to transform from software artifacts. Batista et al. [22] established method for software product line development which incorporates transformation from features to goals and softgoals. Jung et al. [137] used a set of NLP techniques to obtain functional-goals from web services’ descriptions. Source code is also used to elicitate goals. Yu et al. [284] and Wang et al. [264] proposed a method to discover goal models from source code using program slicing and refactoring techniques.

Antón et al. [15] deriving goals from these use cases created based on the Software

Requirements Specification (SRS) document. Bögl et al. [35] proposed an algorithm to construct a hierarchical goal tree from Event-driven Process Chains (EPC) and its annotation. Neiger and Churilov [186] presented a formalization of the relation between business process modeling and decision modeling, thus enabled transition between EPC process model and goal model within decision modeling framework. Koliadis et al. [147] mapped BPMN to i* model to support change propagation between the two models. The mapping includes a step to identify organizational objectives/goals that are not explicitly represented in the BPMN notation. González and Díaz [61] mapped business process goals into system goals for strategic alignment reason using certain heuristics. Zdravkovic, Svec and Giannoulis [293] transformed consumer preferences to i* models that represent consumer value of interest. The obtained i* models are then transformed to feature model that represent the product configurations. Bleistein, Cox and Verner [34] mapped Motivation Model entities, which describes the organization's business strategy, to i* entities, thus making the Motivational Model conceptual framework operational via i*. Grau, Franch and Maiden [101] propose PRiM (Process Reengineering i Method) that provides guidelines for the prescriptive construction of i models. The i* elements are obtained from Human Activity Models (HAMs) that represents the situated behaviour of human actors in the process and Detailed Interaction Script (DIS), which is a simplified notation for process scenarios. Siena et al. [225] proposed Nomos, a framework that includes a process to derive law-compliant system requirements to i* metamodel, including goals of the stakeholders. Pimentel et al. [202] proposed an approach that derives the goals and softgoals from an extended Future Wheel model and use it to enrich the goal model. Ghanavati, Amyot and Peyton [84] mapped the goal and business process models of the legislation and of organizations with Use Case Maps (UCM) that define the business processes that implement organizational policy.

Cardoso et al. [42] using the NFR catalogues as a tool in goal elicitation which is useful to identify goals that did not arise during initial interviews. Islam, Mouratidis and Wagner [134] introduced a framework to assist the elicitation and management of security and privacy requirements from relevant laws and regulations. Kiyavitskaya and Zannone [143] transformed requirements specifications expressed in natural language into semi-structured specifications and generate SI* models, an extended version of i* model. Lee and Liu [156] extracting user intention from the original Web service request terms, using lexical dictionary and domain ontology. Prat [206] discovered goals using semantic functions and formalized goals inside parameters. Ingolfo, Siena and Mylopoulos [133] improved requirements, expressed in i* and Nomos, through compliance checking and proposed a revision to non-compliant requirements. Anton et al. [14] performed comprehensive heuristics to discover goals from requirements and policy documents and converting them into operational requirements. They also defined a taxonomy to categorize the goals and construct a goal model [16]. To extract the goals from the documents, they use automatic text mining technique such as tf-idf and LDA [171]. Similar to the approach by Anton et al., is the implementation of information extraction techniques used by Hui et al. [130]. This approach uses information extraction techniques, such as frequency word, title-keyword, location and syntactic criteria to extract certain parts of a document.

2.3 Business process modeling

Different literatures provide variations on the definition of business process. In the dictionary, the word “process” means a series of actions that produce something or that lead to a particular result [65]. Curtis, Kellner and Over [48] defined process as “*a set of partially ordered steps intended to reach a goal*”. Hammer and Champy [107] defined business processes as “*a set of activities that, together, produce a result of*

value to the customer". Business process is also defined as "A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships" by The Workflow Management Coalition [242]. Weske [269] defined business process as a set of activities that are performed in coordination in an organizational and technical environment.

Ould [192] argues that business process modeling is useful to describe, analyze and enacting a process. Van der Aalst et al. [248] also shared similar view that business process modeling is important and may reduce the risk and the cost of corrections.

Since business processes is complex, different business process modeling was proposed to serve different purposes, represent different things and focus on different aspects [48, 269]. Curtis [48] identified four views on modeling business process, i.e., (1) *the functional view*—describes the functional dependencies between process elements, (2) *the dynamic (behavioural) view*—presents the sequence and control information of the process, (3) *the informational view*—describes the entities that produced, consumed, or manipulated by the process, and (4) *the organizational view*—provides the actor that performs each task and their position in the organization. However most modeling methods represent more than one view [179].

Mili et al. [179] classified the business process modeling language/framework into four categories, i.e.: (a) *traditional process modeling languages*—mostly come from information engineering and business process engineering, typically not formal including IDEF [214], Petri Nets [246, 247], Event Process Chains (EPC) [221], Role Activity Diagrams [192], Resource-Event-Agent (REA) [173], and Business Process Modeling Language [41], (b) *object-oriented languages*—use object-oriented paradigm and notations to represent business process, such as Unified Modeling Language (UML) 2.0 [238] and Enterprise Distributed Object Computing (EDOC) [237], (c) *dynamic*

process modeling language—focus on dynamic view, emphasize on a serialized format for model interchange, represent standardization by industrial body, including Workflow Process Definition Language (WPDL) [138], Business Process Modeling Notation (BPMN) [190], Web Services Business Process Execution Language (WS-BPEL) [136], and Business Process Definition MetaModel (BPDM) [239], and (d) *process integration languages*—intended for integrating the processes of two or more business partners, such as RosettaNet [213], Electronic Business XML (ebXML) [285], and Web Services Choreography Description Language (WS-CDL) [261].

Although there is a large volume of languages and frameworks in business process modeling, several survey reported that BPMN maintain as the most prominent framework among its peers [167, 168, 113, 201]. As such, it is preferred by the industry and become the de-facto standard for business process modeling. For most of our work, we also use BPMN to represent business process models. In the next section, we will briefly describe BPMN.

2.3.1 BPMN

BPMN describes business process using flowchart-based graphical models. The main building blocks of a BPMN model are *Flow Objects* (which consist of Activities, Events, and Gateways), *Data* (represented with Data Objects, Data Inputs, Data Outputs, and Data Stores), *Connecting Objects* that connects flow objects (that consists of Sequence Flows, Message Flows, Associations, and Data Associations), *Swimlanes* (with two alternatives: Pools or Lanes), and *Artifacts* (currently there are two types: Group and Text Annotation) [190].

An *Event* represents anything occurs during the execution of a business process that may influence the flow of the process. It usually has a trigger or an impact (result). The three types of events are Start, Intermediate, and End. An *Activity*

is the task that organization perform. It can be atomic or compound (composed of other activities). There are two types of activity: Task and Sub-Process. For Sub-Process, and additional small plus sign in the bottom center of the Activity notation is added. A *Gateway* controls the splitting and joining of Sequence Flow, as well as the decisions. The type of the gate is indicated by the sign or marker inside the notation. A *Sequence Flow* describes the sequence that activities will be executed in a process. Thus it represents the control flow of a business process. A *Message Flow* represents the message flow between two entities/roles, whose content is represented as a *Message*. An *Association* connects BPMN elements with artifacts or other information. A *Pool* or a *Lane* is used as a container to organize Activities in order to clarify the participants in a business process. A *Group* is also used to group elements in a BPMN model. A *Text Annotation* provides additional information for any element in a model. Figure 2.3 presents the notations for BPMN main elements.

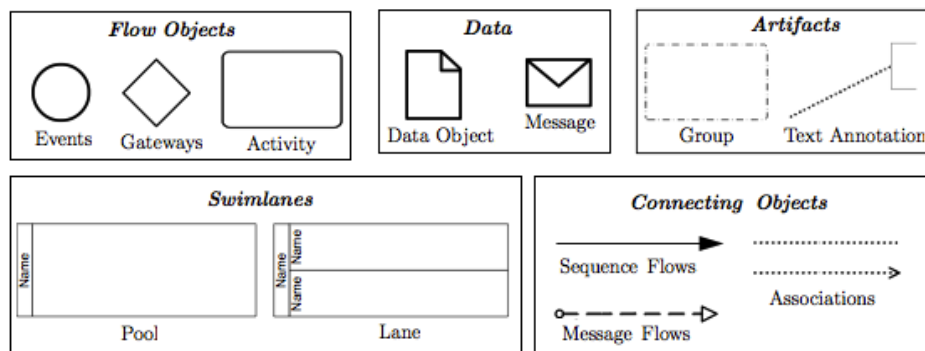


Figure 2.3: BPMN main elements

Ghose et al. [86] and Hinge et al. [123] provide the semantic descriptions of business process modeled in BPMN. In their work, the tasks in the process model are annotated with the effects of the tasks. We use this semantic annotation in our work for reasoning over a BPMN model, particularly for validation of mining results and detection of goal satisfactions. A detailed description of the semantic effect annotation is provided in the next section.

2.3.2 Semantically annotated process model

Our approach relies on the semantically annotated process model where task or event in a process is associated with effects. Previous works on semantic process effects [86, 123, 184] have defined the semantic effect and the accumulation of the semantic effects in the business process model.

A semantic effect (or *effect*) is the result (i.e. product or outcome) of an activity being executed. Effects are written as conjunctive normal form sentences in the underlying formal state description language, which might be propositional or first-order. For convenience, multiple effects (e.g. $\alpha \wedge \beta$) are expressible as a set of effects interchangeably (e.g. $\{\alpha, \beta\}$). Each task or event has context-independent *immediate effects* e that can be contextualized via iterated applications of a *state update operator* as in [86] and [123]. The contextualized effects of an activity is referred as *cumulative effects*, denoted by E .

The contextualized effects are non-deterministic—at any given point in a process, the actual effects that might accrue would be one of a set of *effect scenarios* $\{es_1, es_2, \dots\}$. This support for non-determinism is necessary for two reasons. First, in any process with XOR-branches, one might arrive at a given task via multiple paths, and the contextualized effects achieved must be contingent on the path taken. Since this analysis is done at design time, we need to admit the possibility of non-deterministic effects since the specific path taken can only be determined at runtime. Second, many state update operators generate non-deterministic outcomes, since inconsistencies (that commonly appear in state update) can be resolved in multiple different ways. Of the two well-known state update operators in the literature—the Possible Models Approach (PMA) and the Possible Worlds Approach (PWA)—our work leverages the PWA [90]. Specifically, we use the operator \oplus defined below.

In the following, we assume that all consistency checks implicitly include a back-

ground knowledge base (\mathcal{KB}) containing rules and axioms. Thus, the statement that $e'_i \cup e_j$ is consistent effectively entails the statement that $e'_i \cup e_j \cup \mathcal{KB}$ is consistent. We omit references to \mathcal{KB} for ease of exposition.

For two effects e_i and e_j , and the knowledge base \mathcal{KB} , if $e_i \not\models \perp$ and $e_j \not\models \perp$, then the *pair-wise effect accumulation* (or *state update*) $e_i \oplus e_j$ is defined as:

$$e_i \oplus e_j = \{e_j \cup e'_i \mid e'_i \subseteq e_i \wedge e'_i \cup e_j \cup \mathcal{KB} \not\models \perp \wedge \\ \text{there does not exist } e''_i \text{ such that } e'_i \subset e''_i \subseteq e_i \wedge \\ e''_i \cup e_j \cup \mathcal{KB} \not\models \perp\}$$

The outcome of the state update operation is not a unique effect specification, but a set of non-deterministic *effect scenarios*. To see why this might be the case, consider a task T with a single associated effect scenario given by $\{p, q\}$ which is followed by task T' whose immediate effect is to make r true. Given a background knowledge base consisting of a single rule $r \rightarrow (\neg p \vee \neg q)$, the \oplus operator would give us two distinct outcomes: $\{p, r\}$ and $\{q, r\}$.

To obtain a complete annotation of a process model, we repeatedly apply the \oplus operator over pairs of contiguous tasks in a process model, with the first argument being an effect scenario associated with the prior task and the second argument being the immediate effect of the later task. Special techniques are provided for dealing with XOR and AND gateways in proposals such as [86], [123] and [265]. We briefly explain the technique to accumulate effects by using 2-way joins as an example. The technique can be generalized to handle n -way joins.

Let T and T' be two tasks immediately preceding a join. Let their cumulative annotations be $E_T = \{es_{T_1}, \dots, es_{T_m}\}$ and $E_{T'} = \{es_{T'_1}, \dots, es_{T'_n}\}$, respectively. Let T'' be the task that immediately follow the join with $e_{T''}$ as its immediate effect and $E_{T''}$ as its cumulative effect.

For **AND-joins**, we define $E_{T''} = \{(es_{T_i} \oplus e) \cup (es_{T'_j} \oplus e)\}$, where $es_{T_i} \in E_T$ and $es_{T'_j} \in E_{T'}$. The result of the effect accumulation in this setting is denoted as $ANDacc(E_T, E_{T'}, e)$. Note that we do not consider the possibility of a pair of effect scenarios in AND-joins being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models. For **XOR-joins**, we define $E_{T''} = es_k \oplus e$, where $es_k \in E_T$ or $es_k \in E_{T'}$. The result of the effect accumulation in this setting is denoted as $XORacc(E_T, E_{T'}, e)$. For **OR-joins**, the result of the effect accumulation is denoted by $ORacc(E_T, E_{T'}, e) = ANDacc(E_T, E_{T'}, e) \cup XORacc(E_T, E_{T'}, e)$.

2.3.3 Process mining

Alongside the exploration of the semantic aspect of a business process, there is a vast area of research in mining the structural of the process itself, namely process mining. Unlike the approach of traditional BPM technologies which aimed at improving the effectiveness of a business process through its artifacts (models, data, and systems), the main goal of process mining is to improve the operational processes themselves [250]. Process mining has emerged in the last decade [249] as a comprehensive discipline that offers insights into business processes and supports for process improvements.

Van der Aalst [250] defines three main types of process mining as illustrated in Figure 2.4, they are:

- **discovery:** Given an event log that represents example executions, a discovery method produces a model without the need of any additional information. An example of discovery method is α -algorithm [249]. The input for this algorithm is an event log. The algorithm then constructs a Petri net without any prior knowledge. In addition to petri net, a discovery technique may also produces other resource-related models, such as social network showing interactions be-

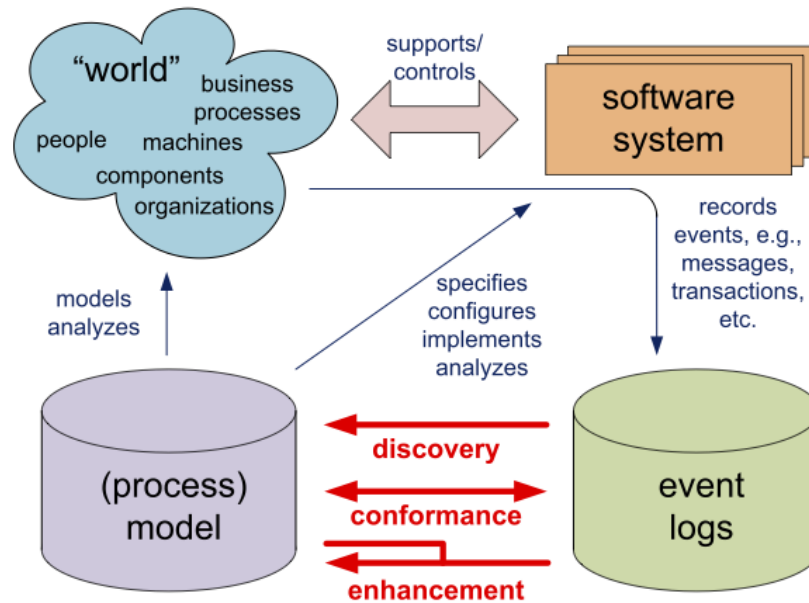


Figure 2.4: Three main types of process mining [250]

tween people in an organization. Other examples of discovery technique are heuristic miner [267] and fuzzy miner [105].

- **conformance:** Conformance checking is used to investigate if the process executions in reality, recorded in an event log conforms to the existing process model, and vice versa. A conformance checking technique compares the event log of a process with the model of the corresponding process. The result of this comparison may discover deviations, furthermore a conformance checking technique can also locate and measure the severity of these deviations. An example of conformance checking is the algorithm in [215].
- **enhancement:** The main objective with enhancement is utilizing information about the actual process enclosed in some event log to improve the existing process model. The improvement may be one of two types, i.e., repair (namely modifying the model with reference to the log) and extension (which is adding a new information to the process model in relation with the reality recorded in

the log).

2.4 The ArchiMate modeling language

The ArchiMate language was realized as part of a collaborative research project on enterprise architecture, funded partly by the Dutch government and involving several Dutch research institutes, as well as governmental and financial institution [154]. In February 2009, the ArchiMate language was transferred to the Open Group who adopted it as a technical standard and published its first version specification. The latest version, the ArchiMate 3.0 Specification [240], was released in June 2016.

ArchiMate [240, 154] is a standardized notation for describing enterprise architectures. In an ArchiMate model, an enterprise architecture is represented through multiple distinct layers, and via relations between elements in adjoining layers. ArchiMate separates the architecture into three layers that are connected to each other through services where the higher layers make use of services that are provided by the lower layers. The layers in ArchiMate are: (1) *Business layer* represents the products and services offered to the external customer, which will be performed by business processes; (2) *Application layer* represents the software applications that supports the business layer and realized by (software) application components; (3) *Technology layer* represents the infrastructure needed to run applications in application layer and realized by computer and communication devices and system software.

The core concept in each layer is illustrated in Figure 2.5. The ArchiMate distinguishes the structural or static aspect and the behavioral or dynamic aspect. There is a close relationship between these aspects: behavioral concepts are assigned to structural concepts, to depict who or what performs the behaviour. The active structure elements on the right side show the actual behavior. On the left side, the passive structural elements, which represents objects on which behavior is performed in terms

of information objects or physical objects. ArchiMate also makes distinction between an external view (on the top) and an internal view (on the bottom) on systems. The service concept represents a unit of essential functionality that some entity e.g. system, organization or department, makes available to its environment. Services are accessible through interfaces, which illustrated in Figure 1 as the external view on the structural aspect. Figure 2.6 presents the structure of the ArchiMate language and the relevant layer-specific concepts.

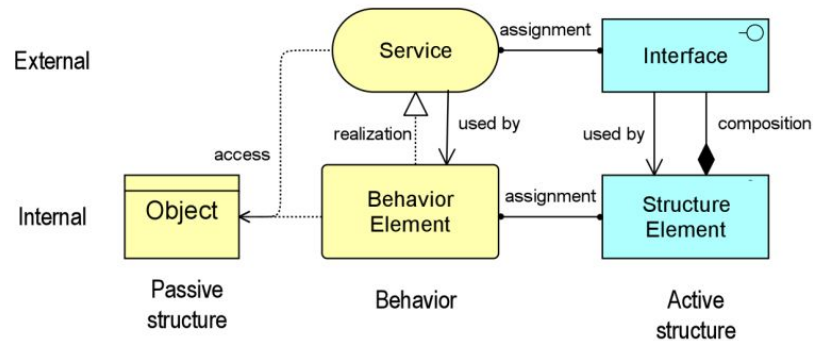


Figure 2.5: Core concepts of the ArchiMate language [240]

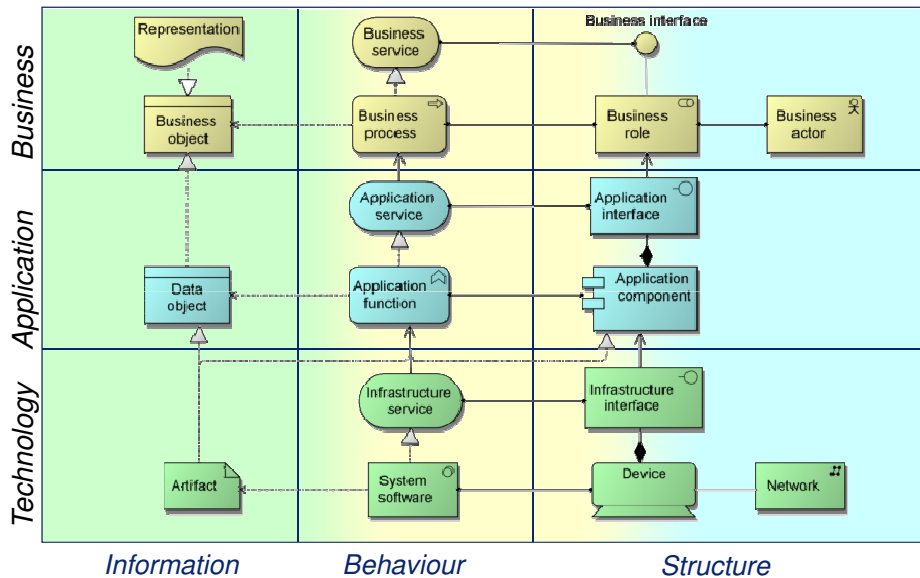


Figure 2.6: Summary of the concepts of the ArchiMate language [240]

There are two main types of relationships between these layers as illustrated in

Figure 2.7:

1. *Serving* relationships which illustrate the support from the applications for the business, both in behavioral and structural aspects. Examples of the serving relationships including relationships between application service and business behavior elements, and between application interface and business role; relationships between business service and application behavior elements, and between business interface and application components.
2. *Realization* relationships which indicate that an object is a realization or a representation of another object. Examples of the realization relationships including relationships between an application process or function and a business process or function; relationships between a data object or a technology object with a business object, which indicate that the data object is a digital representation of the corresponding business object, or the technology object is a physical representation of the business object.

In addition, there may be an aggregation relationship between a product and an application or technology service, and a data or technology object, to indicate that these services or objects can be offered directly to a customer as part of the product.

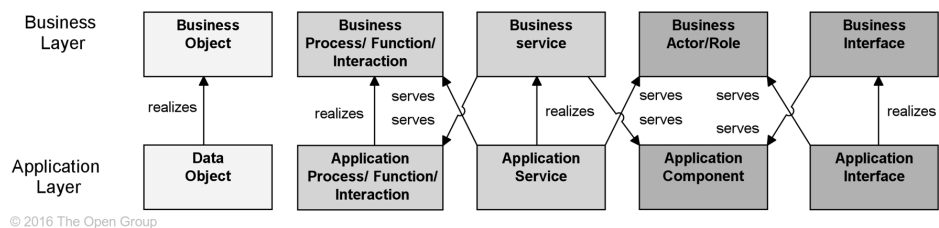


Figure 2.7: Relationships between Business Layer and Application Layer Elements [240]

Figure 2.8 illustrate these relationships in an example of an ArchiMate model. The application services in the application layer serves business processes in the business

layer. For instance, the application layer provides *Claim Administration* services that serves *Notify Additional Stakeholder*, *Validate* and *Investigate* business processes. In turns, the application services are realized by the application components. Continuing with our example, the *Claim Administration* service is realized by the *Home & Away Policy Administration*. The application components use services in the technology layer, which are realized by system softwares. In our example, the *Home & Away Policy Administration* component is supported by *Messaging Service* and *Data Access Service*, which are realized by the *Message Queueing* system and *DBMS*, respectively.

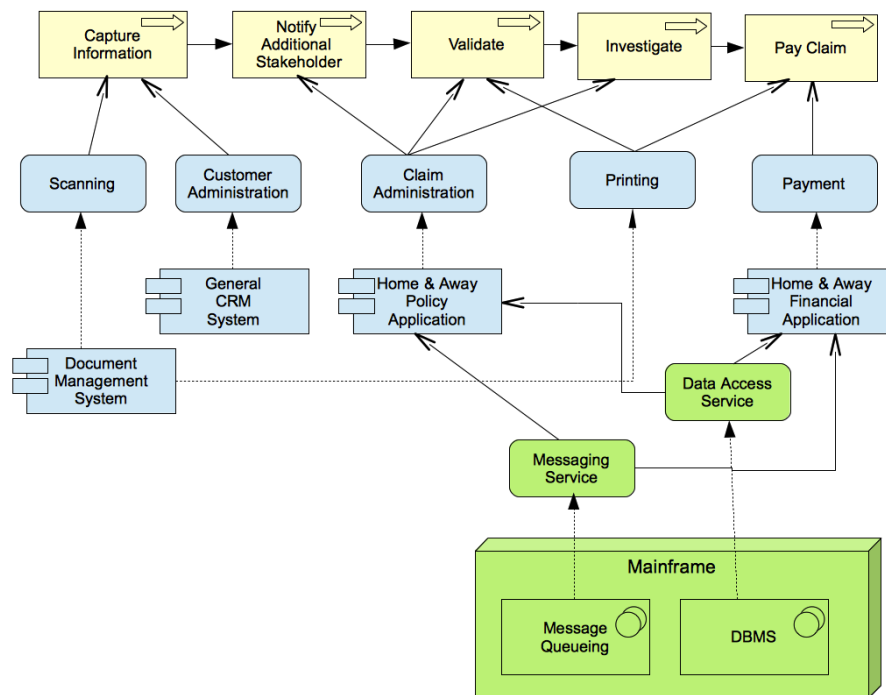


Figure 2.8: ArchiSurance, an ArchiMate example, taken from [135]

2.5 Sequential pattern and sequential rule mining

Since first presented by Agrawal, Imieliński, and Swami in [8], the problem of finding the frequent pattern of items and the relationships among the items are one of the most investigated field in data mining [5, 43, 92, 108, 162, 183]. As an intensively

researched problem, a vast amount of studies have contributed to its progress in terms of methodologies and applications development.

In [8], Agrawal et al. formulated the problem of market basket analysis to understand purchasing behaviour as an *association rule* mining problem [5, 43, 92, 108]. The analysis consists of two parts [5, 43]. First, finding the frequent itemsets or patterns within the dataset. Second, finding the relationship between items within these itemsets. However, the majority of the research in this area concentrated on discovering the frequent patterns as the first step since the level of complexity is more challenging than determining the associations [5, 43, 92, 108].

Closely related to frequent pattern mining is the problem of sequential pattern mining where the items follows a temporal order. Agrawal and Srikant first presented the problem of discovering sequential patterns in [10] and later in [232]. As in the area of frequent itemsets mining, a large number of research have investigated numerous techniques in sequential pattern mining, where some of the algorithms are modifications of known frequent pattern mining methods [5, 108, 162, 183]. Also similar to association rules, sequential rules derived from sequential patterns are considered as “second-stage” output [5].

2.5.1 Frequent pattern mining

The problem of frequent pattern mining is defined as follows [5, 8, 43, 92, 108].

Let $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ be a transaction database, where each $T_i \in \mathcal{T}$, $\forall i = \{1 \dots n\}$ consists of a set of items, say $T_i = \{i_1, i_2, i_3, \dots, i_l\}$. A set $P \subseteq T_i$ is called an itemset. The number of transactions containing P is referred to as the *support* of P . A pattern P is defined to be frequent if its support is at least equal to the minimum threshold.

Research in frequent pattern mining can be categorized into four different fields [5]:

(1) research that investigates a more efficient algorithms for frequent pattern mining, i.e., *technique-centered*; (2) research that specifically study on managing the scalability of the data, i.e., *scalability management*; (3) studies on numerous variants of algorithms to handle different data types and various tasks, i.e., *advanced data types*; (4) studies on the applications of frequent pattern mining in different domains such as chemical and biological domains, i.e., *applications*. Our work falls into the fourth category, applications, since we apply different algorithms into our specific domain, which is business process management.

Aggarwal, Bhuiyan and Hasan in [5] defined the baseline algorithm for frequent pattern mining as presented in Algorithm 1. The inputs to the algorithm are the database that contains the transactions \mathcal{T} and a user-defined minimum support threshold s . First, all 1-item frequent patterns are generated and included into \mathcal{FP} , a data store to hold all the frequent patterns. Then the algorithm generates a candidate pattern based on the frequent patterns already in \mathcal{FP} and computes its support. If the support is equal or higher than s , then the candidate pattern is considered as frequent and stored in \mathcal{FP} . This process continues until all frequent patterns are explored.

Algorithm 1: Baseline Frequent Pattern Mining [5] (Database: \mathcal{T} , Minimum support: s)

```

begin
   $\mathcal{FP} = \{\}$ 
  insert length-one frequent pattern in  $\mathcal{FP}$ 
  for all frequent patterns in  $\mathcal{FP}$  do
    generate a candidate pattern  $\mathcal{P}$  from one (or more) frequent pattern(s)
    in  $\mathcal{FP}$ 
    if  $support(\mathcal{P}, \mathcal{T}) \geq s$  then
      | add  $\mathcal{P}$  to frequent pattern set  $\mathcal{FP}$ 
    end
  end
end

```

Most frequent pattern mining algorithms follows this baseline algorithm. The dif-

ference among these algorithms mainly lies in the pattern exploration and the support calculation strategy. While some algorithms generate the pattern by *level-wise* or breadth-first exploration where all frequent patterns of k -length must be generated first before generating $(k+1)$ -length patterns [11], others prefer *enumeration trees* as their exploration method which enables different strategies for exploration, such as depth-first, breadth-first, or other hybrid strategies [3]. This exploration strategy also has an impact on how the irrelevant and redundant or duplicate candidate patterns are eliminated or *pruned*. Additionally, the exploration strategy also determines how an algorithm counts the support of each pattern, such as whether the calculation on a level can be reused in another level, which will reduce the effort needed. Based on the candidate generation and pattern exploration, Aggarwal and Han [5] categorized the frequent pattern mining algorithms into three groups: (1) join-based algorithms, (2) prefix-tree-based algorithms, and (3) suffix-tree-based algorithms.

(1) Join-based algorithms

Included in this group are all algorithms where the candidate is generated by joining frequent patterns to form a new pattern. The Apriori algorithm [9] is the most basic join-based algorithm. It mainly exploits the anti-monotone Apriori property of frequent patterns [6, 8, 11] that every subset of a frequent pattern is also frequent, or in other words, if any k -length pattern is not frequent, then none of its $(k+1)$ -length super-patterns can be frequent.

Based on this property, the Apriori algorithm generates the candidate patterns by joining k -length frequent patterns to form a $(k+1)$ -length candidate patterns. A candidate pattern may be *pruned* if not *all* its k -length subsets are frequent. The candidate pattern must also be validated by counting its support against the minimal support threshold. Accordingly, the basic Apriori algorithm consists of four steps: (i)

joining the already discovered frequent patterns to generate $(k+1)$ -length candidate patterns, (ii) pruning the $(k+1)$ -length candidates whose all of its subsets are not frequent, (iii) determining if the $(k+1)$ -length pattern is frequent by validating them against the minimum support threshold. The algorithm is terminated when there are no frequent patterns discovered in a given iteration (which means that there are no more candidate patterns that can be generated).

Computing the support of the candidate pattern is the most computationally expensive part of the Apriori algorithm. Several optimizations have been suggested to improve the efficiency of the algorithm, including the AprioriTid and AprioriHybrid algorithm from the same author [9], Direct Hashing and Pruning (DHP) algorithm by Park, Chen and Yu [195] and Apriori_LB by Bayardo [23], while other research proposed alternatives on the implementation level, such as studies by Borgelt and Kruse [36], by Mannila, Toivonen and Verkamo [164].

(2) Tree-based algorithms

All the algorithms in this group explicitly introduce an *enumeration tree* or *prefix tree* or *lexicographic tree* [3] to generate the candidates. As the name suggested, the lexicographic tree is developed based on the lexicographic ordering of the items in the database and built on the prefixes. The root of the tree is the empty set. All 1-length patterns are attached to the root node. Any k -length pattern node is attached to its $(k-1)$ -prefix node. The tree can be grown both ways, either in breadth-first or depth-first order.

By introducing enumeration tree explicitly, algorithms are able to explore candidates in a more flexible way. It also enable the algorithm to achieve more efficiency in counting strategy by avoiding re-doing the counting work. With enumeration tree, the only factor that effects the difference of the number of candidates between different

algorithms is the pruning method.

In IAS algorithm by Agrawal et al. [6, 8], the tree is constructed in level-wise fashion and a transaction database is implemented to calculate the number of any corresponding itemsets at a given level without any optimization in counting strategy. Meanwhile other algorithms use recursive database projections called *TreeProjection* [2, 3]. The number of counting work can be scaled down by performing projection to limit the database size that is used for support calculation. Another method to achieve more efficient counting strategy is by using a vertical representation of the transaction database, such as Eclat by Zaki [288] (and its variant, dEclat by Zaki and Gouda [291]) and VIPER by Shenoy, Haritsa, Sudarshan, Bhalotia, Bawa, and Shah [223].

(3) Recursive suffix-based growth

The suffix-based algorithms also using enumeration tree, but instead of using prefix-based method to build the tree, they apply suffix-based method using extended frequent patterns suffixes.

The FP-Growth method by Han, Pei, Yin and Mao [111] does not require any candidate generation to discover the complete set of frequent itemsets due to the implementation of FP-Tree to represents the conditional transaction database that is used to store the frequent items. The FP-Tree starts with length-1 pattern. The conditional FP-tree is constructed by combining the set of prefix paths in the FP-tree that co-occurred with the suffix pattern to produce the conditional pattern base. This frequent patterns from the conditional FP-tree is then concatenated with the suffix pattern. This operation is performed recursively to discover all frequent patterns. There have been many studies for more efficient runtime and space [193, 287, 95, 98, 99, 152, 158, 208, 207, 233, 234], especially since database keeps growing in volume.

Maximal and closed frequent patterns

Frequent pattern mining often produces a large volume of frequent patterns because every subpattern of a frequent pattern is also frequent itself. This leads to a considerable effort spent to count redundant patterns. To resolve this problem, closed frequent pattern mining and maximal frequent pattern mining were proposed. Closed frequent pattern mining was first introduced by Pasquier et al. in [196], while maximal frequent pattern mining was first presented by Bayardo in [23].

Let $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ be a transaction database. A pattern $P \subseteq T_i$ is a *maximal frequent pattern* if P is frequent and there exists no frequent superset of P in \mathcal{T} . A pattern $P \subseteq T_i$ is a *closed frequent pattern* if P is frequent and there exists no superset pattern of P that has the same support as P in \mathcal{T} .

The first algorithm to mine the maximal frequent patterns was proposed by Bayardo in the same paper [23] called the MaxMiner algorithm. This algorithm is based on the Apriori algorithm with level-wise, breadth-first exploration method and additional optimization in pruning strategy by introducing superset frequency pruning and subset infrequency pruning. On the other hand, the DepthProject algorithm by Agarwal [3], another maximal frequent pattern mining algorithm, is a depth-first algorithm based on the lexicographic tree. The algorithm uses a pruning strategy where a subtree is pruned because all patterns in them are frequent and therefore reduces the search space and improves the counting efficiency. Burdick et al. [39] introduced another method called MAFIA algorithm that improve the counting efficiency by adopting vertical bitmap to represent an itemset.

The first algorithm to mine frequent closed itemsets was also based on the Apriori algorithm, called Close by Pasquier et al. in [196]. To reduce the search space, this algorithm uses the closed itemset lattice and applies the closure function (all subset of

a frequent closed pattern are frequent) iteratively in candidate generation and support counting. CHARM by Zaki and Hsiao [139] uses the closure checking operation introduced in Eclat [288]. On the other hand, to avoid candidate generation, CLOSET by Pei, Han and Mao [199] and CLOSET+ by Wang, Han and Pei [263] use FP-Tree structure proposed in FP-Growth method [111], as well as its mining procedure. FP-Tree structure is also used in FPclose algorithm by Grahne and Zhu [100], although the later combined it with FP-array technique and various optimization techniques. DCI.Closed by Lucchese, Orlando and Perego [160] introduced a searching strategy that can detect and eliminate duplicate patterns during runtime.

Association rule mining

The problem of association rule mining is closely related to finding frequent pattern and introduced together by Agrawal et al. in [6, 8]. In general, most studies derive association rules from frequent patterns and therefore take frequent pattern mining as the imperative step in association rule mining.

The problem of association rule mining is defined as follows [5, 8, 6, 43, 108].

Let $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ be a transaction database, P is a frequent pattern where $P \subseteq T_i$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X, Y \subset P$ and $X \cap Y = \emptyset$. The *confidence* of an association rule is defined as the ratio of the support of the pattern $X \cup Y$ to the support of X . The rule $X \Rightarrow Y$ holds in \mathcal{T} if its confidence is at least equal to the minimum threshold.

The most basic algorithm for association rule mining, follows from the algorithm for frequent pattern mining [8, 6, 290], presented in Algorithm 2. To generate rules, for every frequent pattern P , find all non-empty subpattern of P . For every such subset SP , output a rule of the form $SP \Rightarrow (P - SP)$ if ratio of support P to support SP is at least equal to minimum threshold c .

Algorithm 2: Baseline Association Rule Mining [8, 6, 290] (Database: \mathcal{T} , Frequent patterns: \mathcal{FP} , Minimum confidence: c)

```

begin
  for all frequent pattern  $\mathcal{P}$  in  $\mathcal{FP}$  do
    for all subpattern  $\mathcal{SP}$  of  $\mathcal{P}$  do
      if  $\text{confidence}(\mathcal{P}, \mathcal{SP}) \geq c$  then
        | output  $\mathcal{SP} \Rightarrow (\mathcal{P} - \mathcal{SP})$ 
      end
    end
  end
end

```

2.5.2 Sequential pattern mining

The sequential pattern mining was first addressed by Agrawal and Srikant in [10]. The problem of sequential pattern mining is similar to the frequent pattern mining, however the main difference is that in sequential pattern mining, the mining methods are applied over temporal database [6, 8, 10, 43, 162, 183].

The problem of sequential pattern mining is defined as follows [10, 183, 162, 43].

Let $\mathcal{I} = \{i_1, i_2, i_3, \dots, i_n\}$ be a set of all *items*. An *event* (or an *itemset*) is a non-empty unordered collection of items (without loss of generality, items of an event are sorted in lexicographic order), denoted as $(i_1 i_2 \dots i_m)$, where i_j is an item. A *sequence* is an ordered list of events. A sequence α is denoted as $\langle a_1 a_2 \dots a_q \rangle$ where a_i is an event. A sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ is a *subsequence* of another sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$ and β is a super-sequence of α , denoted by $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$.

The database \mathcal{D} for sequential mining consists of a set of input-sequences. The *support* of a sequence α , denoted $\text{support}_{\mathcal{D}}(\alpha)$, is defined as the total number (or proportion) of input sequences in the database \mathcal{D} that contain α . Given a user-specified threshold called *minimum support*, denoted min_sup , a sequence α is said to be *frequent* if $\text{support}_{\mathcal{D}}(\alpha) \geq \text{min_sup}$. A sequence with k items, where $k = \sum_j |a_j|$ is called a

k-sequence. The set of frequent *k*-sequence is denoted as \mathcal{F}_k .

Several studies on sequential pattern mining algorithms group the algorithms into two categories [5, 162, 183]: (1) Apriori-based algorithms and (2) Pattern growth algorithms. Most of the algorithms are Apriori-based and depend on the Apriori property, while the pattern growth algorithms improve the efficiency of the Apriori-based algorithms by avoiding candidate generation, as also the case in the frequent pattern mining.

(1) Apriori-based algorithms

AprioriAll algorithm [10] was proposed by Agrawal and Srikant. It consists of four steps, namely: (1) sorts the database with sequence id as the major key and timestamp as the minor key, (2) finds the frequent itemsets (the set of all itemsets that satisfy minimum support), (3) transforms each transaction to a set of frequent itemsets from previous step, (4) generates candidate sequence using similar method as in Apriori algorithm, where a candidate sequences is formed from previously mined frequent sequences, until either no candidates are generated or no candidates meet the minimum support. GSP algorithm [232] was also introduced by Srikant and Agrawal. It extends the AprioriAll algorithm by adding time constraints, sliding time windows and taxonomies. However, it still employs the same multiple-pass, candidate-generation-and-test approach. This general approach is also adopted by the PSP algorithm introduced by Masegla, Cathala, and Poncelet [170], however, for retrieval efficiency, PSP uses a prefix-tree for organizing candidate sequences. Any branch of the prefix-tree, from the root to a leaf, represents a candidate sequence, and the terminal node of any branch provides the support of the corresponding sequence.

The previous three algorithms use horizontal data format that requires them to maintain support count for each subsequence being mined in each iteration. There-

fore to improve efficiency, some Apriori-based algorithms uses vertical representation. SPADE [290] and its variant cSPADE [289] by Zaki applies a vertical data format to represent the sequence database combined with lattice-based search techniques. In these algorithms, to find $(k+1)$ -sequence, they join two frequent k -sequence that share the same identifier and their timestamps are sequentially ordered. To discover all frequent sequences, both SPADE and cSPADE break down the search space (called *lattice*) into smaller segments (called *sub-lattices*). These sub-lattices are then searched either by breadth-first or depth-first algorithm. SPAM (Sequential PAttern Mining) by Ayres, Flannick, Gehrke, and Yiu [20] improves the efficiency in the counting process by representing data using a vertical bitmap representation. It introduces an effective pruning method into a depth-first search algorithm. The candidate sequences are stored in a lexicographic tree and generated by extending the sequence either with a new transaction consisting of a single item in the end or with an item in the last itemset. IBM (Indexed Bit Map for mining frequent sequences) by Savary and Zeitouni [220] maps distinct sequences to a bit map and stores its frequency in an NB table. All ordered combinations of sequences are encoded using an SV vector. Candidate generation is conducted in the same manner as GSP, PSP and SPAM. The candidate support is determined by first accessing the cell where the size of the sequence in question is encoded and then using the SV vector to determine if the candidate is contained in subsequent lines of the IBM.

(2) Pattern-growth algorithms

The main disadvantage of Apriori-based algorithms is its high-consumption of memory to store large number of candidate sequence during the mining process, especially when the sequence database or the sequence patterns are large either in volume or in length [109, 111]. Pattern growth algorithms solve this problem by eliminating

the candidate generation and prune steps and dividing the search space into smaller section which are mined separately. This way the algorithm can perform faster when given large volumes of data.

FreeSpan (Frequent pattern-projected Sequential pattern mining) by Han, Pei, Mortazavi-Asl, Chen, Dayal and Hsu [110] uses projection of frequent items into a sequence database to produce a smaller one and generates subsequence fragments in each projected database. PrefixSpan (Prefix-projected Sequential Pattern mining) by Pei, Han, Mortazavi-Asl, Pinto, Chen, Dayal, and Hsu [200], based on FreeSpan, but only the prefix subsequences are checked and only their corresponding suffix subsequences are projected into the database. As a result, only local frequent sequences that are required to be explored to generate sequential patterns in each projected database. The major advantage is that it does not generate and test any candidate sequences that do not exist in a projected database. SLPMiner (Sequential pattern mining with Length-decreasing support) by Seno and Karypis [222] also uses the projection-based approach but it employs length-decreasing support constraint to find both short sequences with high support and long sequences with a lower support.

Maximal and closed sequential patterns

In the frequent pattern mining, when the candidate generation and test techniques and a very low support threshold is used, the algorithm performance degrades. Several algorithms then proposed to mine *frequent closed itemsets* which has been explored in Section 2.5.1. Similar situation also occurs in the sequential pattern mining, which led to the introduction of algorithms to mine the *closed sequential patterns*.

Let \mathcal{FS} be a set of *frequent sequences*. A sequence $\alpha \in \mathcal{FS}$ is a *maximal sequence* if there exists no supersequence of α in \mathcal{FS} . A sequence $\alpha \in \mathcal{FS}$ is a *closed sequence* if there exists no supersequence of α that has the same support as α in \mathcal{FS} .

CloSpan (Closed Sequential pattern mining) by Yan, Han and Afshar [275] based on the PrefixSpan algorithm. It stores the candidate sequences using a lexicographic tree. The candidate sequence is generated using the same method as in SPAM, either by extending the sequence by adding a new transaction consisting of a single item in the end or adding an item in the last itemset. Then to filter out any non-closed sequences, the algorithm uses post-pruning. BIDE by Wang and Han [262] applies a closure checking method called *BI-Directional Extension* to grow the prefix patterns and its closure both in the forward and backward direction. ClaSP (Closed Sequential Patterns algorithm) by Gomariz, Campos, Marin, and Goethals [94] mine the closed sequences based on the vertical database format. It uses the same procedure as SPADE but with additional step to eliminate non-closed patterns. The elimination procedure uses a hash function with the support of a pattern as key and the pattern itself as value. If two patterns have the same support, the algorithm check if one is subsequence of the other, and if this condition is satisfied, the shorter pattern is removed. The ClaSP algorithm is further improved by CM-ClaSP by Fournier-Viger, Gomariz, Campos, and Thomas [74]. It integrates a pruning mechanism called co-occurrence pruning in the ClaSP algorithm [94] during the SEARCH procedure.

For dense database or database with long sequences, sometimes the set of closed patterns is still too large [78, 77]. Therefore the set maximal sequential patterns is introduced as the set of all closed sequential pattern that is not a subsequence of another closed sequential pattern. However, the mining of this set is computationally expensive and a number of algorithms has been proposed [78, 77].

MSPX by Luo and Chung [161] discovers maximal sequential patterns by determining the potentially infrequent candidates using various samples. The new candidates are generated using the remaining candidates after removing all potentially infrequent candidates. DIMASP (Discover all the Maximal Sequential Patterns) algorithm by

García-Hernández, Martínez-Trinidad and Carrasco-Ochoa [82] stores all the distinct pairs of contiguous items and its frequency, then uses this data structure to extract the maximal frequent patterns based on the user-specified threshold. By storing the contiguous items and its frequency, when a new sequence is added, the algorithm does not require to repeat all the work to discover all the maximal sequential pattern, but only only preprocesses the related part. While MSPX and DIMASP need to maintain the intermediate candidates in the memory during mining process, MaxSP (Maximal Sequential Pattern miner) by Fournier-Viger, Wu and Tseng [78] discovers all maximal sequential patterns without storing intermediate candidates in main memory. It uses a checking mechanism to determine if a pattern can be extended either in forward or backward direction. If a pattern can be extended in either way, then it is not maximal, otherwise the pattern is maximal. Therefore with this method, a maximal pattern can be discovered without having to compare with previously found patterns. VMSP (Vertical mining of Maximal Sequential Patterns) by Fournier-Viger, Wu, Gomariz and Tseng [77] uses different approach. This algorithm uses a vertical representation with a depth-first approach of the search space. It stores the set of discovered maximal patterns and then each time a candidate is generated, the algorithm checks whether a super-pattern and/or a sub-pattern of the candidate is already found. The algorithm also uses the item co-occurrence information for pruning the search space.

Sequential rule mining

Similar to the association rule mining problem, once the frequent sequences are known, they can be used to obtain the sequential rules to describe the relationship between different sequence item.

The problem of sequential rule mining is defined as follows [290, 76, 59, 163, 114, 75].

Let \mathcal{D} is a database consists of a set of input-sequences, S is a frequent sequence

where $S \subseteq \mathcal{D}$. A *sequential rule* $X \Rightarrow Y$ is defined as a relationship between two sequence $X, Y \subseteq S$ such that $X \cap Y = \emptyset$ and $X, Y \neq \emptyset$.

The interpretation of a rule $X \Rightarrow Y$ is that if the items of X occur in a sequence, then items in Y will occur afterward in the same sequence. The *support* of a rule $X \Rightarrow Y$ is how many sequences contains the items from X followed by the items from Y . The *confidence* of a rule $X \Rightarrow Y$ is the support of the rule divided by the number of sequences containing the items from X .

Given a user-specified minimum support and minimum confidence, several algorithms has been proposed to generate sequential rules. The algorithm of Das et al. [59] and MOWCATL (Minimal Occurrences With Constraints And Time Lags) algorithm by Harms et al. [114] both mine rules occurring frequently in sequences but are inadequate for discovering rules shared by different sequences. CMRules algorithm by Fournier-Viger, Faghihi, Nkambou and Nguifo [73] was built based on the observation that if the temporal information of a sequence database is removed, then all rules discovered also holds in the original sequence database. Therefore, it applies an association rule mining algorithm such as Apriori [8] after removing the temporal information to discover the sequential rules. In the same paper, the authors also introduced another algorithm, CMDeo [73], a variant of an algorithm by Deogun and Jiang [62]. Using the same level-wise method similar to Apriori [8], the algorithm recursively finds larger candidate rules by combining smaller rules. Fournier-Viger also proposed another algorithm using pattern-growth approach called RuleGrowth [76] based on the method in PrefixSpan [200]. It starts with rules between two single items and then recursively expands them by adding single items from the database, either to the left or right part of the rule. To avoid repeated database projection operation, the author then introduced algorithm for dense database or database with long sequences that uses a vertical representation of the database, ERMiner (Equivalence

class based sequential Rule Miner) algorithm [75]. It explores the search space of rules using equivalence classes of rules having the same antecedent or consequent. A slightly different approach by Lo, Khoo and Wong [159] and Zang, Xu and Li [292] mines the sequential rules from the closed set of sequential patterns.

2.6 Mining of enterprise models

Several requirements elicitation techniques and methods that has been used to mine goals and construct goal model including scenarios [12,13,47,48,53], textual documents such as requirements or policy documents [1,3], and source code [56,61].

The Crews-LEcritoire approach by Rolland et al. [47,48] used goal-scenario coupling (called "requirement chunk") as a mean to discover goals. They performed AND (composition) and OR (alternative) operations to the goal-scenario pair to elaborate goals and build the goal model. This way goals are discovered simultaneously with scenarios. However, the goal coupling can only be applied if the complete goal and scenario are defined; otherwise the composition, alternative and refinement operations cannot be performed. Once the goal and scenario is defined, the software tool LEcritoire will be able to discover new (goal,scenario) pair using the composition and alternative operations. Users then can choose the appropriate goals and scenarios based on their preferences.

The approach by Van Lamsweerde et al. [53] and Damas [12,13] covers both positive or desired scenarios and negative or undesired scenarios to infer a temporal logic specification. They use scenario because it is considered as an instance of system usage and it provides sequences of interaction steps between the intended software and its environment. This method takes scenarios as examples/counterexamples, inductively infers a set of candidate goals/requirements that cover all example scenarios and exclude all counterexample scenarios and generates a set of goal specifications in

temporal logic that covers all positive scenarios and excluding all negative ones. Scenarios are expressed as event trace diagram and the result specifications are expressed in the KAOS goal-based language.

Anton et al. performed several heuristics to discover goals from requirements and policy documents [1]. They take the assumption that goals have not been documented or explicitly elicited therefore to identify goals, the analysts must work from existing documentations, such as process flow diagrams, transcript interviews, etc. Therefore they introduce comprehensive heuristics for discovery and identification of goals and converting goal into operational requirements, although it lacks formal semantics. They also defined a taxonomy to categorize the goals and construct a goal model [2]. To extract the goals from the documents, they use automatic text mining technique such as tf-idf and LDA [37].

Similar to the approach by Anton et al. is the implementation of information extraction techniques used by Hui et al. [37]. This approach uses information extraction techniques, such as frequency word, title-keyword, location and syntactic criteria to extract certain parts of a document. The research takes scientific research papers and patents documents as input and extracts conceptual models from these papers. The three steps in the research were preprocessing, segmentation, and merging. In the preprocessing step, excerpts from a document, which included abstract, introduction and conclusion sections, were extracted. In the next step, these sections were categorized into pre-defined template slots. In the last step, these categorized sentences were refined further by removing unused parts, such as common cue word, and merging similar sentences. Conceptual model were then produced, based on these sentences.

Another approach to discover goal and build the goal model is by Yu et al. [61] and Wang et al. [56]. This techniques based on concepts in the NFR framework where goal has an intended function (intention) and associated topic (subject matter). In

their approach, the goal models are discovered from source code using program slicing and refactoring techniques. These techniques convert source code into a more abstract form by delimiting comments and produce a Hammock graph for each statement block. At the entry and exit of the Hammock graph, pre- and post-condition define allowable classes of input/output states. These states and transitions form a statechart. In this statechart, the action of the a state transition is considered as its function and the contextual state of the statechart is considered as its topic. Therefore a structured statechart can be viewed as a goal model. The goal model produced from this technique closely resembles a class diagram where it shows the connection between methods in the codes rather than the actual goals.

Dalpiaz et al. [10] also introduce a method to build goal model. Their approach is slightly different because they separate the goal model into two different model: design-time goal models to design a system and runtime goal models to analyze a system's runtime behavior with respect to its requirements. They build the runtime goal model based on the design-time goal model with runtime artifacts such as states and fulfillment of goals. Since system behavior is characterized a sequence of events, related to goal instances, therefore the runtime goal model annotated with runtime event traces and constraint. Later on these traces can be used to monitor goal fulfillment.

Another approach for goal mining involve several techniques in the knowledge discovery and data mining such as association rule mining, process mining, and techniques from information extraction [1]. In this approach, to determine relations between two actors in a goal model, association rule mining can be used to perform the search for any patterns which correlates between one set on events with another set of events in a log. But in a dependency between two actors in a model, one actor acts as dependee and another one will act as depender, therefore the order they appear in a sequence is important. Thereof the sequential pattern mining technique is more appropriate in

this case. Any pattern discovered can be interpreted as indication of the existence of a dependency between two actors or tasks. One of the technique available for sequential pattern mining is Generalized Sequential Pattern (GSP) algorithm by Srikant et al. [51]. Information extraction techniques is also another approach to discover goals [1,30], such as tf-idf which can determine the importance of a word in a corpus. The most important keywords then can be interpreted as the goal or the intention of the whole corpus. For both sequential pattern mining and information extraction method, we use off-the-shelf readily available tools, such as StanfordNLP [27], and incorporate the libraries available in the tools for our purpose.

The growing ubiquity of data, the ability to access large-scale sensor instrumentation and the availability of "big data" tools gives another opportunity to perform requirements elicitation through "mining" these data to capture the requirements of the system. Both data and the tools required to instrument the data infrastructures that can generate vast quantities of data are now available, either within the enterprise boundaries or publicly available [23]. For example consider logs from within the enterprise, such as process logs that describe the execution of business process of the enterprise, interaction/message logs that represents communications that occurred either internally or with external parties, and object state monitors that represents the range of objects and their states in the enterprise. Moreover open data source such as the textual content of the web or government-mandate logs can also serve as important source.

Furthermore by utilizing requirements model as dashboard of the system, we can gain multiple benefits. First, the requirements model serves as a visualization of the system's behaviour. In a system, goals are sometimes abstract and implicit [34]. The ability to "mine" these goals and visualize them into a model that is understandable to the user is important, not just to understand the behaviour of the system itself, but

also to be able to discover when the system behaviour deviates from its requirements model [49]. Second, the requirements model can also act as effector, where the changes in the models are reflected to the corresponding data in the system (when we use the word change, we refer to insertions, updates, or deletions that may occur to any of goals). With changes or modification at the model level we would like to be able to reflect only the necessary changes to the system.

Utilizing requirements model as dashboard also implies that the requirements must be represents during run-time. Moreover if the system has the ability to hold the requirements models in memory, when changes occur in the model, the system can react or adapt according to the new model [4]. Sawyer et al. in [49] proposed that such issues can be resolved off-line, but it requires the developers to have full access to the requirements models which they can reason and reach resolution decisions.

One of the approach in representing and modeling requirements during run-time was by the insertion of code into a running system. These codes takes form as monitors [20,46], annotation [11,57], claim [59], or a boolean value [26] and monitored during the system run-time to determine whether the running system complies with the requirements model. However, although these are easy to monitor because they capture actual program behavior, they are difficult to trace back to original stakeholder requirements, and therefore is not easy to adapt when there are changes in the goal model.

Sawyer et al. [49] and Johnson et al. [32] suggested to provide the system with primitives for the goal-based requirements meta-model to cope with the changes, such as primitive for add goal, delete goal, replace goal, obtain agent from goal, and assign agent to goal. To achieve this, they introduce two different layers, the base layer consisting of requirements models and meta-level where dynamic access and manipulation of the requirements objects happens. The requirement objects refers to all objects

related to requirements, such as stakeholders' goals, goal refinements, and domain assumption. Therefore, the primitives are defined to allow the meta-level to modify the goal-based requirements models in the base-level.

Inverardi et al. [31] implemented this approach in their framework where they build a generic meta-layer to manage requirements entities at run-time. This meta-layer is generic and independent from the language used in the implementation. It defines the operations to manage manipulations on the requirements models defined during design time. The operations are to add and delete a requirement and to check satisfiability of a requirement. On top of that, this framework also able to determine whether any inconsistency arise when a requirement is added or deleted from the system. Therefore it also provides operation to validate a requirement with respect to the correspondent implementation.

Instead of defining the primitives to adapt to the changes in the goal models, Goldsby et al. [25] handle adaptation by enumerating all alternative paths at design time. This approach called Levels of RE for Modeling (LoREM). In this approach, during the first phase (goal and requirements phase), the system developer identifies a set of steady-state systems, such that each steady-system is suitable and satisfies the goals, and a requirement model for each steady-state system by describing requirements that the steady-state system should satisfy to achieve goals.

Later on, during system execution, when any failures of any requirements detected, the system will be able to select between existing alternative systems to continually meet its goals.

The previous approach by Goldsby et al. [25] is able to choose among existing correct alternatives that are already defined, but on the other hand it does not define the solutions for unanticipated changes. Ernst et al. [17] propose to overcome these unanticipated changes by using a knowledge base in a logic-based goal-oriented

requirements modeling language called *Techne*. It finds solutions to requirements changes and minimize the effort to implement new solution by re-using as much of the old solution as possible. Every time a changes happens, which include changes to all aspects of requirement including the goals, tasks, etc., the solution is searched incrementally, which means that it starts from the current solution and try to move towards one that meets the problem captured in the new requirement. The obvious benefit of this approach is that the complexity of the calculations is not repeated more than is necessary.

While Ernst et al. [17] using logic-based approach to minimize the effort for new solution, another approach by Nakagawa et al. [41] perform similar task of minimizing effort by localizing the impact of changes only to corresponding part of the system represents as control loops. By localizing the impact of changes, they can minimize increases in code complexity and help to limit influence of changes. They use elaboration process for impact analysis on implementation artifacts once the changes has been identified. This elaboration process involves adding entities to the goal model, merging similar goals and extracting control loops for these goals. If requirements are changed or added, the goal model is updated to accommodate them and perform the elaboration process on it again. As a result, the corresponding control loops acquire new or modified goals. The impact of the changes is analyzed by checking changes in relevant control loops and their dependencies.

Cleland-Huang et al. [9] implement different approach to manage changes. Instead of handling the changes when they occur in the system, they predict the effect before any changes are implemented and based on this prediction, the stakeholders will be able to decide whether the change is carried out or not. Using this approach, goals that effected by the changes are identified along with the level of the impact and appropriate strategies to manage them are developed. To do this, they build a proba-

bilistic network model links between classes and elements. Using these links, whenever a change is proposed, the impact will be propagated throughout related regions and evaluated to determine its effect on system goals. The evaluation result then used by the stakeholders to determine if the change should be implemented or not.

Similar approach using probabilistic network is also used by Cailliau et al. in [7], but instead of using it to predict the effect of changes, they use it to determine the impact in case there is obstacle to a goal in the system. The impact is calculated from leaf obstacles (correspond to obstacles to a leaf goal in the goal model) up to root obstacles (corresponds to obstacles to the root goal in the goal model). The refinement patterns defined in [15] are used to determine the probabilities of the obstacle consequences from a leaf goals to higher-level goals. This shows a method to propagate any impact upwards in a goal model. Just as the method from Cleland-Huang et al. [9], this method is also able to determine any obstacles to the system's goals and calculate the impact therefore the stakeholders can implement the most appropriate countermeasures to be integrated in the system goal model.

Another approach is to explicitly build separate goal model during design and runtime. This approach is used by Dalpiaz et al. [10]. They separate the goal model into two different model: design-time goal models to design a system and runtime goal models to analyze a system's runtime behavior with respect to its requirements. They build the runtime goal model based on the design-time goal model with runtime artifacts such as states and fulfillment of goals. In the runtime goal model, each non-leaf goal is annotated with goal annotation that describe the expected behavior of its subgoals. Therefore any invalid trace in the runtime goal model will indicate a failure in the system.

Di Francescomarino [5] leverage the semantically labelled business processes to automatically verify if business processes fulfill a set of given constraints, and to formu-

late queries that involve both knowledge about the domain and the process structure. Ghose and Koliadis [11] provide a semantic characterization of a minimal revision strategy that capable to detect and partially automate compliance resolution using a notion of compliance patterns and obtain compliant process models from models that might be initially non-compliant. Happer and Stojanovic [15] propose a semantic business process management, Ontoprocess, to provide means for automatically checking the compliance of business processes with business rules by combining semantically described business processes with SWRL rules by a set of shared ontologies. Hoffmann et al. [24] propose a framework where processes are annotated to capture the semantics of task execution, and compliance is checked against a set of constraints posing restrictions on the desirable process states. Morrison et al. [] propose a framework for strategic alignment to understanding of the relationship between a set of processes and the realization of a set of strategies and the optimal set of processes that can achieve these strategies using a semantically annotated process model.

Weber et al. [42] suggest Semantic Business Process Validation (SBPV), an approach that take take the annotations and the underlying ontology into account in order to determine whether the tasks are consistent with respect to each other, and with respect to the underlying workflow structure. Taking inspiration from the semantic of Petri Net, logical information is propagated through the workflow. Wong and Gibbons [44] propose a relative-timed semantic model for BPMN by introducing the notion of relative time in the form of delays to their model. The semantics is defined in the language of Communicating Sequential Processes (CSP). The annotated process model allows behavioural properties of BPMN diagrams to be mechanically verified. Koliadis et al. [30] propose an approach to analyse change against high-level models of the organization. The proposed approach use model annotations to asses relationships between business process and organizational models to improved analy-

sis against higher-level organizational structures, motivations, inter-dependencies and capabilities. Born and Dörr [2] extend the SAP Research modeling tool "Maestro for BPMN" with a flexible annotating semantics in a user-friendly way. The extension exposes ontological knowledge to the business user in appropriate forms and employs matchmaking and filtering techniques to display options with high relevance only. ProcessSEER [23] by Hinge et al. provides a user-friendly framework for analysts to explicitly annotate business process model and automatically computes the post-conditions associated with tasks selected by the user. Hornung et al. [25] propose a recommender system that suggests a list of correct and fitting process fragments for an edited business process model, which can be used to complete the process model being edited.

In regards of assisting designers and analysts, many studies have emerged in harnessing historical data, specifically on software repositories, to discover useful information from various data sources. MI (Mining programmer Interaction histories) [31], a recommendation system by Lee et al., mining finer-grained association rules in software revision histories to recommend files to edit using programmer interaction histories. ROSE (Reengineering of Software Evolution) tool [46] by Zimmermann et al. mining association rules from version histories in order to guide programmers along related changes. Ying et al. [45] propose an approach that applies data mining techniques to determine change patterns (sets of files that were changed together frequently in the past) from the change history of the code base. Remail [1] a plugin for Eclipse by Bacchelli, is a recommendation system for emails that integrates email archives in the IDE and allows developers to easily retrieve discussions related to the chosen code entities. Reverb [37] proposed by Sawadsky et al. is also a recommendation system for developer that recalls and displays pages that are similar to code under active development by the developer based on code-related web pages perused by the

developer in a web browser. Logical Structural Diff (LSDiff), a tool built by Kim and Notkin to support software engineers on inspecting program differences by inferring systematic structural differences as logic rules and noting anomalies as exceptions to the logic rules.

2.7 Datasets

As previously discussed in Chapter 1, various types of data are generated by organizations using a miscellaneous of logging tools. The followings are some of the logs used in this thesis.

2.7.1 Event logs

Event logs have been utilized as a basis for process analysis in many different settings, most notably in the process mining [249, 251, 105, 267]. Essentially, event logs record any event occurrences during process execution. Each record is time-stamped and refers to an event of a particular process instance.

In general, the information enclosed in a record in the event logs usually consists of: (a) instance/case identifier, which describe the process instance an event is related to, (b) event descriptor, which contains the event description, such as a task ID or a state transition, and (c) a time-stamp. Other information, such as the originator or the person responsible for the event, can also be found though not as regular as the former.

During a process execution, the events that occur can be viewed in general terms as being of two types: (1) events that correspond to the execution of the process tasks, i.e., start or end of a process task, and (2) events that correspond to the impact of a process task execution, i.e., the state changes in the objects as the outcome of the process execution.

An example of event log is illustrated in Table 2.1. It shows an event log generated by a *Holiday Booking* process followed by a travel agent. Each record represents one event (either a start of an activity or an observed state). For instance, the first event in the log is the start of activity Receive Itinerary of cust4 on January 1st, 2011 on 11:09.06. The second event represents a state where airline preferences of a customer becomes known (represented by the *airline-preferences-known* state).

Table 2.1: An excerpt of *Holiday Booking* process event log

CustID	Timestamp	Event
cust4	21-01-2015 11:09.06	Receive Itinerary
	21-01-2015 11:11.04	<i>airline-preferences-known</i>
cust3	21-01-2015 11:16.01	Receive Itinerary
cust3	21-01-2015 11:16.32	Check Flight Availability
	21-01-2015 11:17.31	<i>airline-preferences-known</i>
	21-01-2015 11:17.43	<i>airline-classoftravel-known</i>
	21-01-2015 11:18.23	<i>departure-preferences-known</i>
cust1	21-01-2015 11:19.07	Receive Itinerary
	21-01-2015 11:21.10	<i>arrival-preferences-known</i>
cust4	21-01-2015 11:35.02	Check Flight Availability
	21-01-2015 11:35.22	<i>airline-classoftravel-known</i>
cust1	21-01-2015 11:40.06	Check Hotel Availability
	21-01-2015 11:40.20	<i>departure-preferences-known</i>
	21-01-2015 11:40.23	<i>arrival-preferences-known</i>
cust3	21-01-2015 11:41.25	Check Hotel Availability
	21-01-2015 11:43.08	<i>hotel-available-known</i>
	21-01-2015 11:43.25	<i>hotel-available-known</i>
cust3	21-01-2015 11:44.04	Check Tour Availability
cust4	21-01-2015 11:47.09	Determine Feasible Itineraries
	21-01-2015 11:52.00	<i>feasible-itinerary-known</i>
cust4	21-01-2015 11:53.18	Consult Customer
	21-01-2015 11:54.24	<i>customer-confirmation</i>
cust3	21-01-2015 11:57.05	Consult Customer
cust6	21-01-2015 12:02.49	Receive Itinerary
	21-01-2015 12:05.23	<i>customer-confirmation</i>
	21-01-2015 12:07.31	<i>hotel-booking</i>
	21-01-2015 12:09.08	<i>tour-available</i>

2.7.2 Process logs

We refer to the subset of event logs that record the events which signify the start or end of the process tasks execution as process logs. We make distinction of this type of logs with the next type of logs (i.e., effect logs) to emphasize the difference between the two types of events. A variety of business process management tools with event logging capability can be utilized to generate process logs.

Continuing with the example from the previous section, Table 2.2 shows an example of process log. It includes only the events that signify the start of an activity.

Table 2.2: An excerpt of *Holiday Booking* process log

CustID	Timestamp	Activity
cust4	21-01-2015 11:09.06	Receive Itinerary
cust3	21-01-2015 11:16.01	Receive Itinerary
cust3	21-01-2015 11:16.32	Check Flight Availability
cust1	21-01-2015 11:19.07	Receive Itinerary
cust4	21-01-2015 11:35.02	Check Flight Availability
cust1	21-01-2015 11:40.06	Check Hotel Availability
cust3	21-01-2015 11:41.25	Check Hotel Availability
cust3	21-01-2015 11:44.04	Check Tour Availability
cust4	21-01-2015 11:47.09	Determine Feasible Itineraries
cust4	21-01-2015 11:53.18	Consult Customer
cust3	21-01-2015 11:57.05	Consult Customer
cust6	21-01-2015 12:02.49	Receive Itinerary

2.7.3 Effect logs

Effect log is also a subset of event logs. These logs record events that correspond to the outcome or impact of the process execution. The name effect log is preferred in view that the state transitions being recorded are considered as the impact or effect of the process. In many settings, these logs can be generated by the same tools as process logs, however in other settings, an object state monitor has to be installed to specifically obtain them.

As with the previous example, Table 2.3 illustrate the subset of the records in Table 2.1 that are identified as the observed states of the process.

Table 2.3: An excerpt of *Holiday Booking* process effect log

Timestamp	Effect
21-01-2015 11:11.04	<i>airline-preferences-known</i>
21-01-2015 11:17.31	<i>airline-preferences-known</i>
21-01-2015 11:17.43	<i>airline-classoftravel-known</i>
21-01-2015 11:18.23	<i>departure-preferences-known</i>
21-01-2015 11:21.10	<i>arrival-preferences-known</i>
21-01-2015 11:35.22	<i>airline-classoftravel-known</i>
21-01-2015 11:40.20	<i>departure-preferences-known</i>
21-01-2015 11:40.23	<i>arrival-preferences-known</i>
21-01-2015 11:43.08	<i>hotel-available-known</i>
21-01-2015 11:43.25	<i>hotel-available-known</i>
21-01-2015 11:52.00	<i>feasible-itinerary-known</i>
21-01-2015 11:54.24	<i>customer-confirmation</i>
21-01-2015 12:05.23	<i>customer-confirmation</i>
21-01-2015 12:07.31	<i>hotel-booking</i>
21-01-2015 12:09.08	<i>tour-available</i>

2.7.4 Message logs

Message logs record all form of communications in an organization, such as email or any other message exchange applications. In the enterprise context, these types of logs are frequently maintained, and any process logging tool can be used to generate them. Because the message exchange occurs between actors in an organization, these logs usually documented in natural language and require NLP techniques to process them.

Message logs, along with process logs, are widespread. A corporate email repository can be viewed as a message log, although the messages are entirely unstructured. In many cases, messages are structured such as in a variety of Electronic Data Exchange (EDI) languages, or in more recent standards such as RosettaNet and ebXML. The general message structure consists of: (a) Message ID (MID): a universally unique nu-

merical identifier that represents a specific message; (b) Response ID (RID): a number to identify if a message is original or as a reply or forward of another message. If the message is original, then RID is 0; else RID is identical to the MID of the message it replied or forwarded; (c) Time (T): the date and time when the message was sent; (d) Sender (S): the actor that sends the message; (e) Recipient(s) (R): the actor(s) that receive(s) the message; (f) Content: The content of the message.

Table 2.4 illustrates a message log of a travel agent during the Holiday Booking process.

Table 2.4: An excerpt of *Holiday Booking* process message log

MID	RID	Timestamp	Sender	Receiver	Content
id0	0	21-01-2015 11:11.04	cust4	travel agent	Could you find any interesting holiday destination for us?
id1	id0	21-01-2015 11:17.31	travel agent	cust4	I will send you several interesting destinations with available packages.
id3	0	21-01-2015 11:17.43	cust9	travel agent	Could you find any holiday destination in Europe?
id2	id1	21-01-2015 11:18.23	travel agent	cust4	Please find attached list of interesting holiday destination.
id4	id3	21-01-2015 11:21.10	travel agent	cust9	I will send you a list of destinations in Europe.
id6	0	21-01-2015 11:35.22	travel agent	cust4	Please send the preferred date to travel.
id5	id4	21-01-2015 11:40.20	travel agent	cust9	Please find attached list of interesting holiday destinations in Europe.
id7	id0	21-01-2015 11:40.23	cust4	travel agent	The travel date is around June.
id16	0	21-01-2015 11:43.08	travel agent	cust9	Please send me the preferred date to travel.
id17	id16	21-01-2015 11:43.25	cust9	travel agent	The travel date is around August.

2.7.5 Noise

The evaluations in the following sections involved exercises where noises are introduced into the logs. The intent was to simulate the execution of an imperfect system/process,

whose behaviour would be represented by the interleaved logs. In each case, a log was generated using any established machinery. These generated logs are considered as the correct behaviour of a process model, where all instances are precisely following any trace determined in the model. Later on, noise is introduced into these correct logs by: (a) randomly selecting any number of traces in the log; (b) changing either the sequence of activity; or (c) changing the observed state; (d) interleaved the correct and incorrect traces. Therefore, the log contains error, whether in the structural (wrong sequence of activity) or semantical (wrong states are observed) aspect.

The number of noise introduced in the log can also be increased from one exercise to another. Considering the number of records in the log, the number of noise is calculated in percentage. For example, in the log with 100 records, a 10% noise is introduced into the log, which means 10 records are modified such that it represents incorrect behaviour.

2.7.6 Synthetic datasets

Some of the evaluations performed in this thesis use synthetic datasets. These datasets were generated to reflect the correct behavior of a process model, or in other words, it simulates the execution of the process model without any deviation or without any error. The main purpose of this synthetic dataset is for the proof-of-method evaluation, therefore to show that the machinery built to implement the methods is performed as expected and to show that the method produces the result as expected.

The synthetic datasets were generated using a simple machinery of Java programming, taking advantage of the multi-thread capability of the language to simulate the running of multiple instances of one or more process models. By varying the number of running instances or the number of process models, the dataset can be generated to illustrate different temporal settings.

A thread is a light-weight smallest part of a process that can run concurrently with the other parts (other threads) of the same process. Threads are independent because they all have separate path of execution that is the reason if an exception occurs in one thread, it does not affect the execution of other threads. All threads of a process share the common memory. A multi-threaded program contains two or more parts that can run concurrently, and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application. Multi-threading enables to write in a way where multiple activities can proceed concurrently in the same program.

Therefore, in the machinery to generate the dataset, the process models are defined and will be executed. Since a thread executes according to its main program, each thread represents one instance of the process model. With multi-thread programming, it is possible to have several independent instances of the same process model running simultaneously. It is also possible to have multiple process models defined and when the multi-thread programming is executed, each thread chooses randomly which trace or process model it is executed. For each thread or each process instance, an ID is given as the identifier and the start time is recorded to represent the timestamp in the event log. The result is a dataset in the form of event log which consists of multiple process instances, running independently of each other. Each record consists of an event identifier and a timestamp that signify the start time of the event.

2.8 Research gap

The idea to utilize history data of process execution as a data source is not new. One research area where this idea grows rapidly is process mining. Process mining is a research discipline that discovers, monitors and improves real processes by extracting knowledge from event logs readily available from today's system. It links the modeled behavior on one hand and the observed behavior on the other hand. There are three types of process mining techniques: discovery, conformance, and enhancement. Process discovery techniques take an event log as input and produces model that best described the behavior observed in the log, mostly to provide insights into what occurs in reality. Conformance checking techniques takes a process model and an event log of the same process as input and compares the observed behavior in the log with the behavior allowed by the model to identify where and when deviations occur and measure the severity of such deviations. Enhancement techniques take a process model and an event log as input to extend and improve the model with information extracted from the log.

In regards of assisting designers and analysts, many studies have emerged in harnessing historical data, specifically on software repositories, to discover useful information, with data such as programmer interaction history [23], software revision history [36,35], email history [1], visited web pages [27], and bug reports [19]. Particularly in business process modeling, there are extensive studies on process mining that exploit the historical data of process model executions, i.e. event logs. Process mining algorithms—such as alpha algorithm [31], heuristic miner [33], and fuzzy miner [12]—extract the structure of the process model.

The mining of the historical execution data is the starting point of this thesis, which looks at different models that are possible to be mined and addresses one specific aspect: the data-driven conceptual modeling. This thesis explores different methods to

assist analysts in developing different models for an enterprise by leveraging the already available data. Furthermore, this thesis complements the findings from previous research concerning the process model described in BPMN. This research introduces the mining of process semantic to build a semantic-annotated process model.

In summary, most of the prior research on the developing mining from the enterprise data either has focused on one model of the enterprise nor does it leverage the execution data from the running process. On the other hand, the previous research which has used the execution data, in the form of event logs, are mainly interested in mining the structural aspect of the process. Furthermore, none of these methods is concerning on mining the goal satisfaction of a running process and modeling the enterprise based on this insight.

Chapter 3

Mining task post-conditions: Automating the acquisition of process semantics

A large and growing body of work explores the use of semantic annotation of business process designs [70, 123, 227, 265, 63, 86]. A large body of work also addresses the problem of semantic annotation of web services in a similar fashion [169, 178, 181, 226]. Common to all of these approaches is the idea that semantic annotation of process tasks or services provides value in ways that the process or service model alone cannot. The focus in this chapter is on *post-conditions* of tasks in the context of process models (pre-conditions are also of interest and we believe that an extension of the machinery presented here can address these, but are outside the scope of the present work). However, the modeling and acquisition of these post-conditions poses a particularly difficult challenge. It is generally recognized that process modeling involves significant investment in time and effort, which would be multiplied manyfold if there were an additional obligation to specify semantic annotations.

Section 3.1 provides an introduction into the approach. Section 3.2 introduces the

data sources that were exploited in the method. The data-driven approach to mining the semantic annotations is provided in Section 3.4 and to validate these annotations in Section 3.5. An abductive approach to repair any incomplete or unsound mined effects is presented in Section 3.6. An empirical evaluation to the method is provided in Section 3.7. Lastly, this chapter is summarized in Section 3.9.

3.1 Introduction

Ideally process designs annotated with post-conditions help answer the following question for any part of a process design: *what changes will have occurred in the process context if the process were to execute upto this point?* Arguably, a sufficiently detailed process model (for instance one that decomposes tasks down to the level of individual read or write operations) will require no additional information to answer this question. However, process models are most valuable when described at higher levels of abstraction, in terms of concepts and activities that stakeholders are familiar with. Processes annotated with post-conditions thus serve a crucial modeling function, providing an effective summary of a substantial body of knowledge regarding the “lower-level” workings of a process. Annotation with post-conditions can also help solve a range of problems such as process compliance management [86], change management [144], enterprise process architectures [106] and the management of the business process life cycle [146].

The modeling and acquisition of these post-conditions poses a particularly difficult challenge. It is generally recognized that process modeling involves significant investment in time and effort, which would be multiplied manyfold if there were an additional obligation to specify semantic annotations. Analysts also tend to find semantic annotation difficult, particularly if the intent is to make these formal (as is required by all of the use cases referred to above). This chapter seeks to address this

challenge by offering a set of techniques that mine readily available data associated with process execution to generate largely accurate “first-cut” post-conditions for process tasks or activities (we use the terms “task” and “activity” interchangeably in this chapter).

The approach leverages the generally understood notion of *event logging*. The events that occur in a process execution context can be viewed in general terms as being of two types: (1) events that describe the start or end of the execution of process tasks and (2) events that describe state changes in the objects impacted by a process. In many settings, the existing event logging machinery is capable of logging both kinds of events. In other settings, we need to instrument *object state monitors* (for either physical objects or computational objects, or both) to obtain events of the second kind. One such approach on event logging is the event processing framework for business process management by Herzberg et al. [118, 119, 120, 121, 122].

These two types of logs were leveraged in juxtaposition, and the time-stamped sequences of task execution events and state-change events thus obtained, to generate the *sequence database* taken as input by a sequential rule miner (CMRules [73] in this instance, but others could be used instead). The key idea is to identify commonly occurring patterns of activity execution events, followed by sequences of state change events. As we show, the approach is generally quite effective. We also define techniques which leverage a *state update operator* (that defines how a specification of a state of affairs is updated as a consequence of the execution of an action) and the actual history of process execution provided by the juxtaposed activity executions and state changes to determine whether the mined post-conditions, if accumulated using the state update operator, would indeed generate the available execution histories. This forms a validation step for the mined results.

Our intent is to mine the *context-independent post-conditions* (or immediate out-

come) of each activity. These are *contextualized* via iterated applications of the state update operator to obtain the *context-dependent post-conditions* of each activity (in the context of a process model)—a complete collection of these for each activity or event provides a semantically annotated process model. For instance, the outcome of turning a switch on is to complete a circuit. In the context of a light bulb circuit, the context-dependent post-conditions of this activity would be to turn the bulb on. In the context of a switching circuit for a chemical reactor, the context-dependent post-conditions of that same activity would be to bring the chemical reactor to an operational state. We envisage the machinery we present below being used in the following manner: given as input a set of events that describe the execution of activities, a set of state-change events, a process model (or a set of process models in the event that the logs describe the execution of instances of multiple process designs) and a state update operator, the machinery would generate the post-conditions of each activity referred to in the recorded events. These post-conditions could be used directly in annotating process models, or might be viewed as “first-cut” specifications, to be edited and refined by expert analysts.

The problem we solve can be summarized as follows. Given: (1) a log of process events, (2) a log of object state transition events, (3) a process model or models whose execution generated these logs and (4) a state update operator, the context-independent post-conditions of every task/activity referred to in the process event log. Inputs (1) and (2) are used in the mining phase, while inputs (3) and (4) are used in the validation phase.

This chapter extends the results presented in [218] in a number of important ways. First, this work presents a more sophisticated approach to validation. Second, it offers a novel abductive framework for repairing mined post-conditions, based on soundness and completeness analysis contained in the validation approach. Third, the work

presents more extensive empirical analysis.

3.2 Example

Process designs are intended to be abstract, enabling users to get a handle on a complex underlying reality. Thus the *effects* or *impact* of a process is often not directly reflected in the high-level abstractions contained in a process design. The proposal offers a means of mining these effects and correlating these with elements of a process design. Compelling examples of such processes can be found in domains such as medicine, logistics, financial services and so on. We will use a clinical process as the running example in this chapter.

Specifically, we will focus on a clinical process for the treatment of juveniles with head injuries, drawn from [191]. Figure 3.1 illustrates the complete process of head injury treatment. Initial evaluation aims to quickly determine the severity of injury and to initiate the appropriate treatment immediately. After the primary and secondary survey, the patient with head injury is treated according to the risk category. Patients with high risk of intracranial injury have to undergo a head CT scan and a consultation with a paediatric expert. Any abnormalities observable on a CT scan should be treated according to neurosurgical advice. In the absence of abnormalities, a period of prolonged observation is required due to the risk of cerebral oedema or delayed bleeding. This extended period of observation also applies for any patients displaying features of an intermediate risk group. If an acute deterioration or any persistent symptoms (vomiting, headache, irritability, abnormal behavior or unsteady gait) is detected at six hours after injury, a head CT is indicated. Otherwise, the patient may be discharged.

Consider four patients with different conditions. We describe the process instances for two of these patients below, while Table 3.1 describes the task sequence that applied

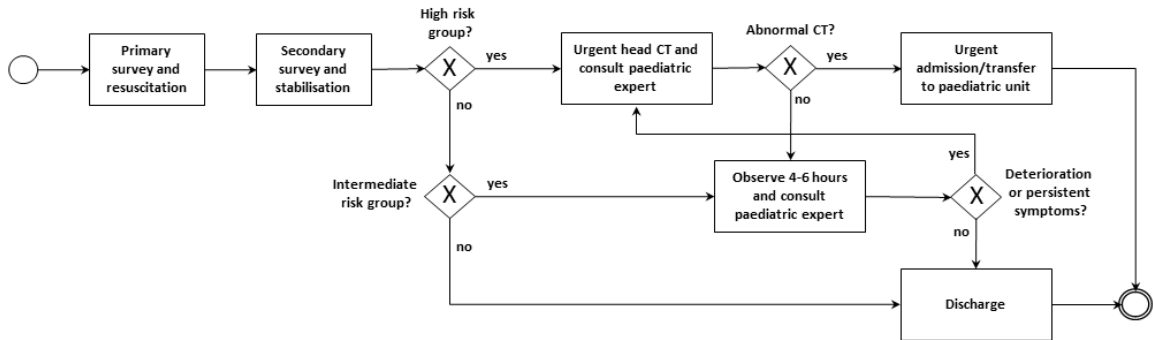


Figure 3.1: Clinical process for treatment of juveniles with head injuries [191]

to all four patients:

patient1 Patient presented as a member of the high risk group (abnormal cardio-respiratory function, loss of consciousness for more than 5 minutes, retrograde amnesia more than 5 minutes, abnormal behaviour, abnormal drowsiness, seizure although the patient is non-epileptic, non-accidental injuries, persistent headache, co-morbidity, fall from higher than 3 m height, laceration on the head, low GCS, oxygen saturation less than 95%, intubated). After the patient had undergone a head CT, the results indicated intra-cerebral bleeding, therefore the patient was transferred to the paediatric unit.

patient2 Patient presented as a member of the high risk group (normal cardio, abnormal respiratory, loss of consciousness for more than 5 minutes, retrograde amnesia more than 5 minutes, with abnormal behaviour, abnormal drowsiness, seizure although the patient is non-epileptic, non-accidental injuries, persistent headache, co-morbidity, victim of motor vehicle accident, swelling and laceration on the head, low GCS, oxygen saturation less than 95%, intubated). After the patient underwent a head CT, the results came back as normal, therefore the patient was put under observation for 4-6 hours. During the observation period, there was no further deterioration and the symptoms resolved, therefore

the patient was discharged.

Table 3.1 shows an event log that records the sequence of clinical interventions for each of *patient1*, *patient2*, *patient3* and *patient 4*.

Table 3.1: Records of patient's treatment

Time	Patientid	Treatment
t1	patient1	primary survey and resuscitation
t5	patient2	primary survey and resuscitation
t27	patient2	secondary survey and stabilisation
t30	patient1	secondary survey and stabilisation
t54	patient3	primary survey and resuscitation
t77	patient4	primary survey and resuscitation
t82	patient3	secondary survey and stabilisation
t84	patient4	secondary survey and stabilisation
t105	patient1	urgent head CT and consult paediatric expert
t124	patient4	discharge
t126	patient2	urgent head CT and consult paediatric expert
t135	patient3	observe 4-6 hours and consult paediatric expert
t141	patient2	observe 4-6 hours and consult paediatric expert
t148	patient1	urgent admission/transfer to paediatric unit
t154	patient2	discharge
t162	patient3	urgent head CT and consult paediatric expert
t173	patient3	urgent admission/transfer to paediatric unit

Table 3.2 stores the condition of each patient (for ease of exposition, we only show the records for *patient1* and *patient2*). Every change in a patients condition is recorded in this table together with a time-stamp. We use an underlying clinical vocabulary (or a state description language) to represent a patients condition. For instance, in the first record, at time *t1*, *patient1s* heart rate and blood pressure are measured and categorized as normal (represented as $heart_rate(patient1, normal) \wedge blood_pressure(patient1, normal)$). The condition of *patient2* is much the same when assessed at time *t5* (represented as $heart_rate(patient2, normal) \wedge blood_pressure(patient2, normal)$). At time *t11*, *patient1* is intubated. The most obvious effect of this clinical intervention is recorded in the table as $intubated(patient1)$.

Table 3.2: Records of patient's conditions

Time	Patientid	Conditions
t1	patient1	heart_rate(patient1, normal) \wedge blood_pressure(patient1, normal)
t2	patient1	normothermia(patient1)
t3	patient1	\neg oxygen_saturation(patient1, normal) \wedge \neg PaO2_level(patient1, normal) \wedge \neg PaCO2_level(patient1, normal)
t4	patient1	GCS(patient1, low)
t5	patient2	heart_rate(patient2, normal) \wedge blood_pressure(patient2, normal)
t6	patient2	normothermia(patient2)
t7	patient2	\neg oxygen_saturation(patient2, normal) \wedge \neg PaO2_level(patient2, normal) \wedge \neg PaCO2_level(patient2, normal)
t8	patient2	GCS(patient2, low)
t9	patient2	cervical_spine(patient2, immobilise)
t10	patient1	cervical_spine(patient1, immobilise)
t11	patient1	intubated(patient1)
t12	patient1	systemic_blood_pressure(patient1, adequate)
t13	patient1	maintenance_fluids_administered(patient1)
t14	patient1	opiates_administered(patient1)
t15	patient1	sedation_score(patient1, high)
t16	patient1	blood_glucose(patient1, normal)
t17	patient1	analgesia_administered(patient1)
t18	patient1	anti_emetics_administered(patient1)
t19	patient2	intubated(patient2)
t20	patient2	systemic_blood_pressure(patient2, adequate)
t21	patient2	maintenance_fluids_administered(patient2)
t22	patient2	opiates_administered(patient2)
t23	patient2	sedation_score(patient2, high)
t24	patient2	blood_glucose(patient2, normal)
t25	patient2	analgesia_administered(patient2)
t26	patient2	\neg anti_emetics_administered(patient2)
t27	patient2	loss_of_consciousness(patient2)
t28	patient2	\neg anterograde_amnesia(patient2) \wedge retrograde_amnesia(patient2)
t29	patient2	mild_agitation(patient2) \wedge altered_behaviour(patient2) \wedge \neg abnormal_drowsiness(patient2)
t30	patient1	loss_of_consciousness(patient1)
t31	patient1	\neg anterograde_amnesia(patient1) \wedge retrograde_amnesia(patient1)
t32	patient1	\neg mild_agitation(patient1) \wedge \neg altered_behaviour(patient1) \wedge abnormal_drowsiness(patient1)
t33	patient1	vomiting_without_other_cause(patient1)
t34	patient1	seizure(patient1) \wedge non_epileptic(patient1)
t35	patient2	vomiting_without_other_cause(patient2)
t36	patient2	seizure(patient2) \wedge non_epileptic(patient2)
t37	patient2	non_accidental_injury(patient2)
t38	patient2	headache(patient2)
t39	patient2	co-morbidities(patient2)
t40	patient2	\neg age_under_1yr(patient2)
t41	patient2	motor_vehicle_accident(patient2) \wedge \neg fall(patient2)
t42	patient2	GCS(patient2, low)
t43	patient2	focal_neurological_abnormality(patient2)
t44	patient2	\neg penetrating_injury(patient2)
t45	patient2	\neg suspected_depressed_skull_fracture(patient2) \wedge \neg suspected_depressed_base_of_skull_fracture(patient2)
t46	patient2	\neg scalp_bruise(patient2) \wedge swelling(patient2) \wedge laceration(patient2)
t47	patient2	\neg tense_fontanelle(patient2)
t48	patient1	non_accidental_injury(patient1)
t49	patient1	headache(patient1)
t50	patient1	co-morbidities(patient1)

3.3 An event ontology

The approach was derived from the event processing framework for business process management by Herzberg et al. [118, 119, 120, 121, 122]. In this framework, a process model is correlated with a set of data objects and each data object has a defined life cycle. The notion of a *data object* permits us to abstract information (of various kinds including information that reflects states in the life-cycle of real-world objects) being processed or manipulated during process execution [121].

During process execution, a wide variety information about changes or exceptions in the business process environment can be represented through event objects, e.g. the start of a ceratain activity, the state change of certain data object, etc. In this work, we focus on only two types of event objects: (1) *process events* which record the start of the execution of a task or activity, and (2) *object state transition events* that describe the impact of process execution via state changes in the impacted objects (which could be computational objects, such as data items, or real-world objects, such as a piece of machinery or a switch). We are only interested in recording the state of these objects that are the result of the state transitions, and do not record the prior states.

Since object state transition events represent the effects of executing a process, we will on occasion use the terms *object state transition* and *effect* interchangeably.

We can now relate these event types to the running example from the previous section. The process events in that example are recorded in Table 3.1. The object state transition events in that example are recorded in Table 3.2. It is useful to note that these latter events essentially describe the condition of a patient. For example, the first row at Table 3.1 indicates that activity *primary survey and resuscitation* was started at time t_1 . The first row in Table 3.2 indicates at time t_1 the condition of *patient1* as $heart_rate(patient1, normal) \wedge blood_pressure(patient1, normal)$. In this

example, the objects are the patients and the object state transition events describe various aspects of the state of a patient after a particular activity/medical intervention has been performed.

These event objects can be obtained by instrumenting the process environment with object state monitors (both for physical objects as well as for computational/business objects). For this purpose, we describe the state changes or transitions using the state description language that might involve propositional state variables—the changes to describe would then be propositions becoming true or false, or more generally as disjunctions (in case state monitors have limited sensing capabilities). The underlying language might also admit non-Boolean state variables, in which case the states recorded would be the new value assignments to these objects. When annotating a process model with object state transitions cause by each task, it is convenient to use first-order sentence schemas. Thus, we would use a sentence schema such as *heart_rate(Patient, Status)*, which would be instantiated with a ground sentence such as *heart_rate(patient1, normal)* in a log of object state transitions.

In the head injury treatment example, the “*primary survey and resuscitation*” activity would lead to a ground instance of the sentence schema *normothermia(patient1)* becoming available. In this setting, the precise grounding of the *Patient* objects are not of particular interest. Indeed, recording the actual values of these objects would lead to the procedure treating different groundings as distinct objects, when in fact we are only interested in recording the fact that a ground instance of that sentence schema has become available. For states of this sort, we only record a propositional effect of the form *normothermia-known*. In a similar fashion, it is sufficient to record *patient-heart-rate-known* rather than the fact that *patient1* has a *normal* heart-rate (as described in *heart_rate(patient1, normal)*). In other settings, we are interested in the precise instantiations of the objects in a sentence schema of the form $p(X, Y)$,

in which case the full ground instance of $p(X, Y)$ is recorded in the object transition events table.

The approach to mine the activity post-conditions involves (1) correlating process events and object state transition events as represented in the database (in this section), and then (2) filtering these by validating them (in the next section).

3.4 Mining post-conditions

The approach to post-conditions mining is predicated on the observation that the state transitions of objects impacted by executing an activity occur soon after the execution of the activity. State transitions that manifest a long period after the execution of an activity are typically not the effect of that activity alone, but of that activity plus some others (e.g., one may think of the arrival of a traditional “snailmail” letter 3 days after posting as an outcome of the action of letter-posting, when it actually involves several other activities executed by the postal service). The key pattern we leverage in mining post-conditions is the *sequence* that involves the execution of an activity and the manifestations of its *object state transitions*, using a sequential pattern miner. We are interested in identifying all the state transitions that occur always (or most of the time) after each activity is executed. Since the process executions are recorded as event objects and the state transitions occurrences are recorded as object transition events, we must first establish the correlations between the two tables that records both events to obtain a joined table that serves as the *sequence database* for a sequential rule miner. We use the CMRules algorithm [73] although a number of other candidates exist (see Section 2.5), and the framework is flexible enough to allow the use of any of these.

While the focus is on the sequential patterns that relate event objects to object state transitions, we are not interested in the relative sequencing amongst state transitions. Indeed, it is undesirable for this purposes to have the sequential rule miner to view

the sequences $\langle T, p, q \rangle$ and $\langle T, q, p \rangle$ as being distinct. We therefore enforce the rule that a contiguous sequence of state transitions in the sequence database must always be represented in lexicographic order (this would require the second sequence above to be re-written as the first sequence).

We consider the problem of post-conditions mining in two settings: (1) Settings characterized by the **unique activity assumption** which stipulates that only one activity may be performed at any point in time. This permits us to correlate all of the state transitions observed between the execution of a given activity and the start of the next activity with the first activity. (2) Settings characterized by the **concurrent activity assumption** which admits the possibility of multiple activities executing concurrently (these could be activities associated with distinct instances of the same process or associated with different processes). The second setting is more general, but the first setting simplifies the post-conditions mining problem, and is worth considering if appropriate. We will apply the CMRules algorithm in both settings.

In general, a sequential rule $X \rightarrow Y$ consists of two parts: the antecedent X and the consequent Y , which are both assumed to be sequences of transactions. The rule states that if the elements of X occur in a given sequence in the sequence database being mined, then the elements of Y will follow in the same sequence and in a manner that preserves the sequential relations between the elements of X and between the elements of Y . All sequential rules must also satisfy certain criteria regarding their accuracy (minimum confidence) and the proportion of the data that they actually represent (minimum support). The CMRules algorithm takes as input a sequence database along with a user-specified thresholds: minimum support ($minSeqSup$) and minimum confidence ($minSeqConf$). It outputs the set of all sequential rules that satisfy the $minSeqSup$ and $minSeqConf$ thresholds. The algorithm consists of two steps. The first step involves obtaining a transaction database from a sequence database without

considering the sequential information. The algorithm then finds all association rules from the transaction database using an association rule mining algorithm, such as Apriori [8]. All association rules discovered must satisfy the minimum support and minimum confidence thresholds which is set equal to $minSeqSup$ and $minSeqConf$. In the next step, the algorithm then scans the original sequence database to calculate the support and confidence of each association rule, and eliminates the rules that do not satisfy $minSeqSup$ or $minSeqConf$. The rules that satisfy both thresholds are considered as sequential rules. To apply this algorithm in this setting, we must first combine the *process event log* and the *object state transition log* into a *sequence database*.

Both in settings with the *unique activity assumption* and in settings with concurrent activities, we create a joined table from the event objects and the object transition events of the form: $\langle\langle T_1, \langle\langle e_{11} \rangle, \dots, \langle e_{1n} \rangle \rangle \rangle, \dots, \langle T_i, \langle\langle e_{i1} \rangle, \dots, \langle e_{im} \rangle \rangle \rangle, \dots, \langle T_p, \langle\langle e_{p1} \rangle, \dots, \langle e_{pk} \rangle \rangle \rangle$ where each $\langle T_i, T_{i+1} \rangle$ pair represents contiguous activities and each e_{ij} represents the j -th state transition observed after the start of activity T_i and before the start of activity T_j . We shall henceforth refer to this table as the *Joined ProcessEvent-StateTransitionEvent table*. This table serves as the sequence database provided as input to the sequential rule miner. A special provision is needed for the last activity in case it does not have any subsequent activity. Instead of using the last record in the event objects table as the end timestamp, we assume that we have prior information about the maximal time of process execution, ϵ , and use it as the end time of the last activity in any case.

We then apply the CMRules algorithm, with the best results obtained when the values of $minSeqSup$ and the $minSeqConf$ are bounded from below by the number of distinct case-ID in which a specific activity occurs (as with any association rule min-

ing technique, *minSeqSup* and *minSeqConf* represent the support and confidence respectively—higher values of these can give us more reliable results but rule out potentially interesting rules and vice versa). In *unique activity* settings with no noise, the sequence of state transitions following the execution of each activity and prior to the execution of the next activity in the process instance should be largely identical if the process design is fixed—we apply CMRules mainly to mitigate the effects of noise. In *concurrent activity* settings, these could vary significantly since the state transitions that follow an activity might not be the output of that particular activity but those of a distinct concurrent activity. In these settings, the sequential rule miner is essential to identify the commonly occurring patterns of state transitions following a given activity.

For example, consider *patient1* in Table 3.2. The first activity, *primary survey and resuscitation* has timestamp t_1 and the next activity for the same patient, *secondary survey and stabilisation*, has timestamp t_{30} ; therefore, we associate activity *primary survey and resuscitation* with all state transitions observed between the timestamps t_1 until t_{30} . This gives us the sequence (*primary survey and resuscitation*)(*heart-rate-known*)(*blood-pressure-known*)(*normothermia-known*)(*oxygen-saturation-known*)(*PaO2-level-known*)(*PaCO2-level-known*)(*GCS-known*), etc. Similarly, activity *secondary survey and stabilisation* is associated with all state transitions with timestamps between t_{30} until t_{105} , and so on. Applying the same process to all the other cases, we obtain the sequences for all activities in the process instance for *patient1*. Next, these sequences are grouped into a sequence databases based on their activity name. For example, the sequence for activity *primary survey and resuscitation* for *patient1* goes into the same sequence database with the sequence for the activity *primary survey and resuscitation* sequence for *patient2* (along with activity *primary survey and resuscitation* sequences for other

patients).

Although the CMRules algorithm is able to generate all sequential rules from the sequence databases, further post-processing is required. Since we are interested only in relations between an activity and state transitions, only rules with a single activity name as antecedent are included in the results and all other rules are discarded.

3.5 Validation

We can use the state update operator and the available data to *validate* the mined post-conditions. The intuition is to leverage available data to determine if the mined post-conditions predict the object state transitions seen in the data. We offer tests for soundness and completeness, and an abductive framework to guide the repair of mined post-conditions. We consider two settings, the first mainly for testing purposes and the second because it reflects real-life operations.

Unique activity assumption: The analysis described below can be performed in settings satisfying the **unique activity assumption** which precludes multiple concurrently executing process instances. Note that such settings are rare in practice. This analysis is nonetheless useful for two reasons. First, it is possible to create *test runs* of processes that satisfy this assumption. Second, this affords the opportunity to develop the overall validation approach, which is subsequently specialized for the more practical setting.

A joined ProcessEvent-StateTransitionEvent table associates with each task a set of effects that occurred after the execution of that task and prior to the execution of the next task in the execution sequence. We use the following procedure to obtain, from a given joined ProcessEvent-StateTransitionEvent table, a *cumulative joined ProcessEvent-StateTransitionEvent table*. Each row in the latter associates with each task the set of accumulated effects of all tasks executed upto this point. Note that the

remainder of the exposition ignores the initial state that accrued at the start of the process (mainly to reduce the complexity of the formalization), but this can be trivially added if required. Let each row in the joined ProcessEvent-StateTransitionEvent table be of the form $\langle T_i, E_i \rangle$ where E_i is a set of literals (i.e., indicators of object state transition events). We assume that there is also a background knowledge base \mathcal{KB} defined in the same language as that in which the effects are described.

The procedure involves the following steps:

- We set the first entry of the cumulative joined ProcessEvent-StateTransitionEvent table to be $\langle T_i, \{E_i\} \rangle$.
- We obtain each subsequent entry in the cumulative joined ProcessEvent-StateTransitionEvent table (of the form $\langle T_i, \mathbb{E}_i \rangle$) from the prior entry using the following rule: $\mathbb{E}_{i+1} = \mathbb{E}_i \oplus E_{i+1}$.

The following example illustrates how this is done (we use this procedure to obtain Table 3.4 from Table 3.3).

Table 3.3: An example of joined ProcessEvent-StateTransitionEvent table

T_i	E_i
T_1	p, q
T_2	r, s
T_3	t

Table 3.4: An example of cumulative joined ProcessEvent-StateTransitionEvent table

$\mathcal{KB} : t \rightarrow \neg(p \wedge r)$	
T_i	\mathbb{E}_i
T_1	$\{\{p, q\}\}$
T_2	$\{\{p, q, r, s\}\}$
T_3	$\{\{p, q, s, t\}, \{r, q, s, t\}\}$

An element of the joined ProcessEvent-StateTransitionEvent table can be viewed as a *semantic execution trace* (i.e., a sequence of tasks interleaved with observed effects after each task), or part of one, of the form:

$$\langle\langle\langle T_1, \langle\langle e_{11} \rangle, \dots, \langle e_{1n} \rangle \rangle \rangle, \dots, \langle T_i, \langle\langle e_{i1} \rangle, \dots, \langle e_{im} \rangle \rangle \rangle, \dots, \langle T_p, \langle\langle e_{p1} \rangle, \dots, \langle e_{pk} \rangle \rangle \rangle\rangle$$

for a process instance (case) with p activities, with each T_i representing an activity ID and each e_{ij} representing the result of the j -th state transition associated with activity T_i . An element of the cumulative joined ProcessEvent-StateTransitionEvent table associates with each task both the effects observed after the execution of that task and the effects of prior tasks that persist. These are obtained, as shown above, by applying the state update operator. Since the outcome of the application of the state update operator can be non-deterministic in general, we associate with each task a set of sets of effects (as illustrated with task T_3 in Table 3.4 above). We shall refer to the sequence of activities $\langle T_1, \dots, T_p \rangle$ as the *signature* of the semantic execution trace above, and note that multiple semantic execution traces might be obtained for the same signature (due to the fact that we might find the process in one of many possible non-deterministic states after the execution of a sequence of activities).

To validate the post-conditions mined using the procedure described in the previous section, it is useful to establish:

- *Soundness*: The soundness condition states that the mined post-conditions are correct, i.e., observed in the data. In other words, mined post-conditions, accumulated via the state update operator upto a given point in a process must be included in the observed set of accumulated post-conditions at that point in the process. Formally, for each semantic execution trace manifested in a cumulative joined ProcessEvent-StateTransitionEvent table and for each activity T_i there must exist an associated set of observed (accumulated) effects es_i (the entry in the cumulative joined ProcessEvent-StateTransitionEvent table corresponding to

T_i), such that the following holds: $es_i \cup \mathcal{KB} \models e$ for some $e \in e_{T_1} \oplus e_{T_2} \oplus \dots \oplus e_{T_i}$ where each e_{T_i} denotes the mined post-conditions of activity T_i (recall that the application of the \oplus operator can lead to multiple non-deterministic outcomes, making $e_{T_1} \oplus e_{T_2} \oplus \dots \oplus e_{T_i}$ a set). or a sufficiently extensive collection of process and object state transition events, we may also require that there must exist, for every $e \in e_{T_1} \oplus e_{T_2} \oplus \dots \oplus e_{T_i}$, some entry in the cumulative joined ProcessEvent-StateTransitionEvent table with an es_i associated with T_i such that $es_i \cup \mathcal{KB} \models e$.

- *Completeness:* The completeness condition requires that all observed post-conditions are mined. This is essentially the reverse of the previous entailment relation (i.e., $e \cup \mathcal{KB} \models es_i$).

Concurrent activities: In settings which permit multiple active process instances and where multiple activities might be concurrently executed, we cannot guarantee that the post-conditions observed between the start of an activity and the start of the next activity in the same process instance are necessarily the post-conditions of the former activity (since concurrent activities from other process instances might have led to these). In such settings, we validate by creating modified sequence databases, parameterized by an activity sequence length parameter n for use with CMRules. For instance, when the activity sequence length parameter is 2, for each contiguous pair of activities $\langle T_i, T_j \rangle$, we take sequences of the form $\langle T_i, e_{i1}, \dots, e_{in} \rangle$ and $\langle T_j, e_{j1}, \dots, e_{jm} \rangle$ where the activities belong to the same process instance and where the timestamps associated with each e_{ik} is earlier than the start of T_j and create an entry in this modified sequence database of the form $\langle T_i, T_j, \tau(e_{i1} \wedge e_{i2} \wedge \dots \wedge e_{in} \oplus e_{j1} \wedge e_{j2} \wedge \dots \wedge e_{jm}) \rangle$ ¹.

The result of applying τ represents the result of performing state update on the post-

¹ τ is a function that takes a sentence in conjunctive normal form and outputs a sequence consisting of its conjuncts (recall that the relative sequencing between these is of no interest from this perspective). Thus $\tau(e_{i1} \wedge e_{i2} \wedge \dots \wedge e_{in}) = \langle e_{i1}, e_{i2}, \dots, e_{in} \rangle$

conditions of T_i with the post-conditions of T_j . If the state update operation leads to multiple non-deterministic outcomes, we create separate entries for each (sharing the same prefix $\langle T_i, T_j \rangle$). We use CMRules to obtain rules of the form $\langle T_i, T_j \rangle \rightarrow \langle e_1, \dots, e_p \rangle$ with the support and confidence being set as earlier to refer only to those process instances where T_i and T_j appear contiguously. We can now use the following *soundness* condition: There exists a modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ such that the corresponding suffix (viewed as the conjunction of its elements) $e_1 \wedge \dots \wedge e_n \cup \mathcal{KB} \models e$ for every $e \in e_{T_i} \oplus e_{T_j}$. We can similarly state a *completeness* condition: For every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \dots \wedge e_n$ (as before, we view the suffix as the conjunction of its elements), there must exist an $e \in e_{T_i} \oplus e_{T_j}$ such that $e \cup \mathcal{KB} \models e_1 \wedge \dots \wedge e_n$ (note that this will work only if we deal with contiguous sequences activities starting with the first activity). The approach generalizes to activity sequences of arbitrary length, but we omit details for ease of exposition. A general validation strategy is to consider all activity sequences of length $i = 1, \dots, n$ where n is the length of the longest activity sequence that conforms to the process design.

3.6 Abductive repair

We now consider the problem of what needs to be done when mined post-conditions are found to be unsound or incomplete according to the tests described above. An easy solution is to seek more data and mine again. More interestingly, we can offer guidance to analysts in manually modifying the first-cut post-conditions mined from available data by using a simple formulation as an abductive problem. The discussion focuses on settings with concurrent tasks, but the approach easily extends to the simpler class of settings satisfying the unique activity assumption.

We consider first the case where the mined set of activity post-conditions are found

to be incomplete. If we start the analysis with activity sequences of length 2, let $\langle T_i, T_j \rangle$ be the first pair of contiguous activities for which we violate the completeness condition. A finding of incompleteness entails that there are post-conditions (object state transitions) observed in the data which are not predicted by the mined immediate post-conditions. In other words, the mined post-conditions need to be *augmented* to redress this. We need to decide now what post-conditions to add and to which activity. Formally, the abductive problem is to identify the minimal (with respect to set inclusion) $a \subseteq A$ where A is the set of *abducibles* (in this case the vocabulary of post-conditions being used), given mined post-conditions e_{T_i} and e_{T_j} for tasks T_i and T_j such that at least one of the following hold:

- There exists an $e \in (e_{T_i} \wedge a) \oplus e_{T_j}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \dots \wedge e_n$ (from here on we will view effect sequences as the conjunction of their elements for simplicity) such that $e \cup \mathcal{KB} \models e_1 \wedge \dots \wedge e_n$
- There exists an $e \in e_{T_i} \oplus (e_{T_j} \wedge a)$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \dots \wedge e_n$ such that $e \cup \mathcal{KB} \models e_1 \wedge \dots \wedge e_n$

The first condition above corresponds to augmenting the post-conditions of T_i with a while the second corresponds to augmenting the post-conditions of T_j with a . If both conditions can be satisfied, we make a non-deterministic choice of any one task (and augment its post-conditions).

We consider next the case where the mined set of post-conditions are found to be unsound. If we start the analysis with activity sequences of length 2, let $\langle T_i, T_j \rangle$ be the first pair of contiguous activities for which we violate the soundness condition. A finding of unsoundness entails that there are mined post-conditions that are not

observed in the data. In other words, we need to *restrict* or *contract* one or more sets of post-conditions to redress this. We need to identify $e'_{T_i} \subseteq e_{T_i}$ and $e'_{T_j} \subseteq e_{T_j}$ such that both of the following hold:

- There exists an $e \in e'_{T_i} \oplus e'_{T_j}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ (with a corresponding suffix $e_1 \wedge \dots \wedge e_n$) such that $e_1 \wedge \dots \wedge e_n \wedge \mathcal{KB} \models e$
- There exists no e''_{T_i} where $e'_{T_i} \subset e''_{T_i} \subseteq e_{T_i}$ which satisfies the condition that there exists an $e \in e''_{T_i} \oplus e''_{T_i}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ (with a corresponding suffix $e_1 \wedge \dots \wedge e_n$) such that $e_1 \wedge \dots \wedge e_n \wedge \mathcal{KB} \models e$

The e'_{T_i} and/or e'_{T_j} identified via this analysis are set as the new post-conditions of T_i and T_j respectively.

We need to start this analysis with the first activity T_1 (since the modified sequence database includes the accumulated post-conditions of *all* tasks starting with the first), then incrementally expand the sequence of contiguous activities. Thus the first sequence of activities considered would be $\langle T_1, T_2 \rangle$, then $\langle T_1, T_2, T_3 \rangle$ and so on. Once we have ensured that a given sequence of mined post-conditions $\langle e_{T_i}, \dots, e_{T_i} \rangle$ is sound and complete, we expand the sequence by one activity, obtaining $\langle e_{T_i}, \dots, e_{T_i}, e_{T_{i+1}} \rangle$. Ensuring the new mined post-condition (i.e., $e_{T_{i+1}}$) is sound and complete is simpler, since only one candidate set of post-conditions needs to be either augmented or contracted. As above, repairing $e_{T_{i+1}}$ for incompleteness involves identifying the minimal (with respect to set inclusion) $a \subseteq A$ where A is the set of *abducibles* (in this case the vocabulary of post-conditions being used), given mined post-condition e_{T_i}, \dots, e_{T_i} which are known to be sound and complete such that at least one of the following hold: There exists an $e \in e_{T_1} \oplus \dots \oplus e_{T_i} \oplus (e_{T_{i+1}} \wedge a)$ such that for every modified sequence database entry with the prefix T_1, \dots, T_i, T_{i+1} and a corresponding suffix

$e_1 \wedge \dots \wedge e_n, e \cup \mathcal{KB} \models e_1 \wedge \dots \wedge e_n$. Similarly, repairing $e_{T_{i+1}}$ for unsoundness we need to identify $e'_{T_{i+1}} \subset e_{T_{i+1}}$ such that both of the following hold:

- There exists an $e \in e_{T_1} \oplus \dots \oplus e_{T_i} \oplus e'_{T_{i+1}}$ or every modified sequence database entry with the prefix $\langle T_1, \dots, T_i, T_{i+1} \rangle$ (with a corresponding suffix $e_1 \wedge \dots \wedge e_n$ such that $e_1 \wedge \dots \wedge e_n \wedge \mathcal{KB} \models e$
- There exists no $e''_{T_{i+1}}$ where $e'_{T_{i+1}} \subset e''_{T_{i+1}} \subseteq e_{T_{i+1}}$ which satisfies the condition that there exists an $e \in e_1 \oplus e_{T_i} \oplus e''_{T_{i+1}}$ for every modified sequence database entry with the prefix $\langle T_1, \dots, T_i, T_{i+1} \rangle$ (with a corresponding suffix $e_1 \wedge \dots \wedge e_n$) such that $e_1 \wedge \dots \wedge e_n \wedge \mathcal{KB} \models e$

The $e'_{T_{i+1}}$ identified via this analysis is set as the new post-condition of T_{i+1} .

A number of abductive reasoning techniques can be used to support automation of this analysis, but we leave this outside the scope of this work (see [197] for a good survey of available techniques).

3.7 Evaluation

Evaluation with synthetic process models: The aim is to establish that the approach generates reasonably reliable results. We ran the first set of experiments with a synthetic semantically annotated process model (i.e., a hand-crafted one with T_1, T_2, \dots etc, for task names and p, q, \dots for effects). The model had 8 activities, with an AND-split nested inside an XOR-split and with each task semantically annotated with 1 or 2 literals (in the 2 literal case, the states were conjunctions of the 2 literals), and one rule in the \mathcal{KB} . We simulated a large number of possible execution traces of this model, and obtained process and state transition events. These events involved the execution of multiple concurrent process instances. There were multiple possible states associated with some of the tasks in the process design, owing to the fact that

XOR gateway contributed to alternative flows that could have led to the same point (none of the states were generated by alternative means of resolving inconsistency in the state update operator). We then investigated the effect of scaling up the complexity of the process model, by generating a second synthetic process model with 12 activities with an XOR-split leading to two alternative flows, one of which included a nested AND-split and the other a nested XOR-split. The semantic annotations were 2 or 3 literals long and involved a mix of conjunctions and disjunctions. The background \mathcal{KB} had 4 rules. There were multiple effect scenarios associated with most of the tasks and these were generated both by alternative flows that could lead to a task (on account of XOR gateways) and by alternative resolutions of inconsistency by the state update operator.

Table 3.5 below describes the results of 4 experiments with each of these two process models. We used progressively larger numbers of overlapping instances of each process (i.e., T_i in instance 2 would start after the start of T_i in instance 1, but before the start of T_{i+1} in instance 1, and so on). We note that the problem would be no harder if the multiple concurrent process instances were of multiple distinct process models. We obtained progressively larger sizes of the sequence database. We recorded the precision (number of correct post-conditions mined over the total number of post-conditions mined) and recall (the number of correct post-conditions mined over the total number of actual post-conditions). Although not entirely monotonically improving, the results for process 2 confirm the intuition that better results are obtained with larger datasets. The results for process 2 also showed that the post-conditions mined tended to be incorrect for the last activity in a process instance (in those settings where precision and recall values were less than 1). This was due to the sequence of post-conditions for the final task not being bounded by the start of the next activity, but rather by the end of the log (artificially determined by length of the longest process).

Table 3.5: The recall and precision measures from the evaluation

	Process model 1				Process model 2			
Number of instances	5	10	100	500	5	10	100	500
Size of sequence DB	48	100	1082	5352	66	133	1297	6512
Recall	1.0	1.0	1.0	1.0	0.953	1.0	0.981	0.989
Precision	1.0	1.0	1.0	1.0	1.0	0.988	1.0	1.0

The synthetic process and state transition events used in these examples considered all possible flows. Real-life data might involve more imperfections (such as certain XOR flows never being executed, certain activities never being executed and so on). We have also considered cases where noise is artificially added to the entries - as expected, precision and recall suffer as noise increases. We performed experiments with 500 instances of the second model. The proportion of noise in the complete effect log ranges from 5 to 20%. We plotted the performance of the technique (in terms of recall and precision) against this parameter. As expected, recall and precision decreases as the amount of noise increases. The results in Figure 3.2 were consistently the same.

We took the mined post-conditions and used the validation technique from the previous section to repair the post-conditions (in the case of abductive repair we did not use any automated abductive framework, but used the abductive repair guidelines to perform manual repair). The result shown in Figure 3.3 suggests that the approach is effective in identifying inaccurate post-conditions (and repairing them) leading to an increase in precision measures.

User-mediated evaluation: User-mediated evaluation: To evaluate the approach in a more real-life setting, we took a real-life semantically annotated process model that illustrates a *Holiday Booking* process followed by a travel agent in Figure 3.4 and obtained a set of process and state transition events from an expert process modeler. We obtained a log of process events describing 10 execution instances (many of them with temporal overlaps) with a total of 110 entries, and a state transition events log

Noise percentage	Precision
5	1.00
10	0.83
15	0.78
20	0.50

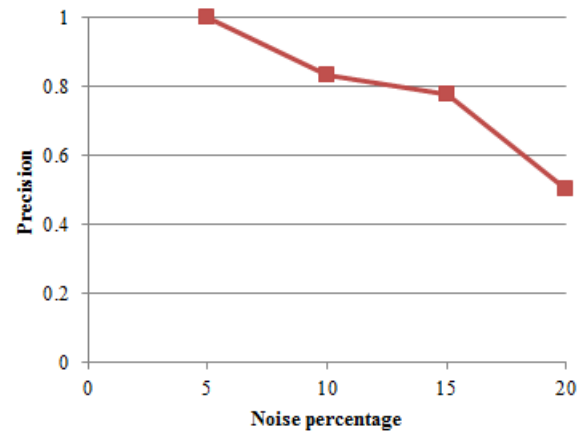


Figure 3.2: Precision measures with noise in the effect log

Noise percentage	Precision (before)	Precision (after)
5	1.00	1.00
10	0.83	0.83
15	0.78	0.79
20	0.50	0.67

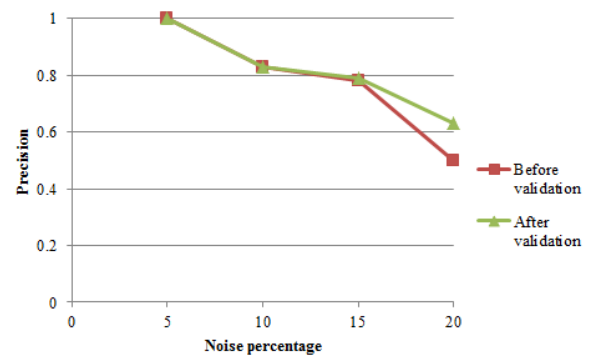


Figure 3.3: Precision measures after validation with length parameter 2 and 3

with 154 entries. Excerpts of both tables are presented in Table 3.6 and Table 3.7. In Table 3.7 we do not use real airline or hotel names.

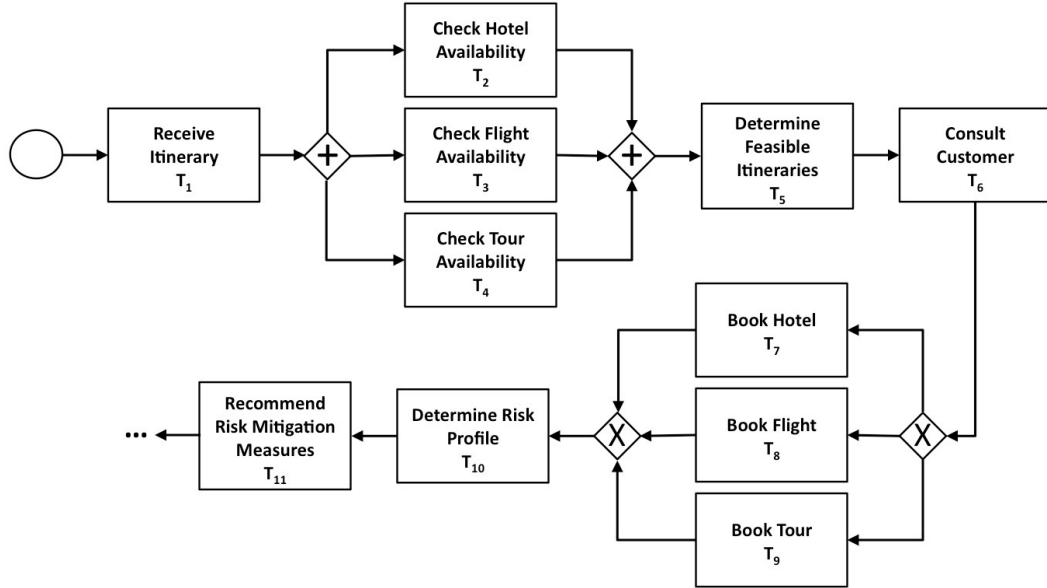


Figure 3.4: A semantic annotated BPMN process model for *Holiday Booking* process

We found that in about 1 in 9 activities, the post-conditions mined were incorrect, in the sense that the mined post-conditions did not correspond to the post-conditions provided by the expert process modeler. The best explanation of this appears to be the fact that in the user-generated process events, there were other activities that were exactly concurrent with the activity for which the wrong post-conditions were mined.

3.8 Related works

Artifact-centric business process modeling. An approach in the space of artifact-centric business process modeling is the GSM (Guard-Stage-Milestone) model by Hull et al. [51, 131]. In the GSM model, the state of an artifact at any given point during the execution of the model is described using three elements: (a) milestone, which represents a business objective with achieving and/or invalidating conditions;

Table 3.6: Excerpt from the process event log provided by the user

Time	Customerid	Activity
46	cust4	Receive Itinerary
47	cust9	Receive Itinerary
53	cust3	Receive Itinerary
53	cust3	Check Flight Availability
72	cust1	Receive Itinerary
72	cust4	Check Flight Availability
77	cust1	Check Hotel Availability
78	cust3	Check Hotel Availability
81	cust4	Check Tour Availability
87	cust7	Receive Itinerary
93	cust5	Receive Itinerary
99	cust6	Receive Itinerary
106	cust9	Check Hotel Availability
116	cust1	Check Flight Availability
116	cust6	Check Tour Availability
125	cust3	Check Tour Availability
130	cust6	Check Hotel Availability

Table 3.7: Excerpt from the state transition event log provided by the user

Time	Observed states
87	airline-preferences(cust4,Airline-23)
90	airline-classoftravel(cust4,ClassOfTravel-1)
114	departure-preferences(cust4,DepartTime-9)
117	arrival-prefs(cust4,ArriveTime-3)
120	meal-constraints(cust4,MealConstraints-47)
121	freq-flyer(cust4,FreqFlyer-53)
133	hotel-pref(cust4,Hotel-75)
137	room-prefs(cust4,RoomPref-95)
144	tour-prefs(cust4,TourPref-71)
155	hotel-available(Hotel-75,Dates-16)
173	flight-available(Flight-7,Airline-23,ClassOfTravel-1,DepartTime-9,ArriveTime-3)
191	tour-available(Tour-34,DepartTime-9,DepartLoc-35,Route-6 Stops-3)
203	feasible-itinerary(Flight-7,Hotel-75,Tour-71,CustPref-2)
220	customer-confirmation(cust4,Itinerary-30)
224	hotel-booking(cust4,Itinerary-30,Hotel-75,Dates-16)
233	flight-booking(cust4,Itinerary-30,Flight-7,Airline-23 ClassOfTravel-1,DepartTime-9,ArriveTime-3)
237	tour-booking(cust4,Itinerary-30,DepartTime-9,DepartLoc-35,Route-6,Stops-3)

(b) stage, which consists of a cluster of activities to achieve a milestone (in the atomic level, a stage consists of one activity); (c) guard, that controls whether a stage is active/open or not. Status change of a milestone and/or a stage is triggered by an incoming event in the form of a request or a task termination notification from the environment. Artifact-centric approaches such as GSM are of interest in the context mainly because of their focus on artifact lifecycles (in this vocabulary: object state transition events).

Semantically annotated process model. A number of proposals in the literature consider semantic annotations of processes in a manner similar to ours, and would stand to benefit from implementations of the framework. A few examples of the benefits that can be exploited from semantically annotated business process models including compliance checking, management level strategic alignment of business processes, and exception handling [85]. Di Francescomarino [63] leverage the semantically labelled business processes to automatically verify if business processes fulfill a set of given constraints, and to formulate queries that involve both knowledge about the domain and the process structure. Ghose and Koliadis [144] provide a semantic characterization of a minimal revision strategy that capable to detect and partially automate compliance resolution using a notion of compliance patterns and obtain compliant process models from models that might be initially non-compliant. Happel and Stojanovic [112] propose a semantic business process management, Ontoprocess, to provide means for automatically checking the compliance of business processes with business rules by combining semantically described business processes with SWRL rules by a set of shared ontologies. Hoffmann et al. [124] propose a framework where processes are annotated to capture the semantics of task execution, and compliance is checked against a set of constraints posing restrictions on the desirable process states. Morrison et al. [184] propose a framework for strategic alignment to understand the

relationship between a set of processes and the realization of a set of strategies and the optimal set of processes that can achieve these strategies using a semantically annotated process model.

A number of proposals add semantics to specify the dynamic behaviour of the business process, such as those by Weber et al. [265] and by Wong and Gibbons [273]. Semantic Business Process Validation (SBPV) [265] by Weber et al. is an approach that takes the annotations and the underlying ontology into account in order to determine whether the tasks are consistent with respect to each other, and with respect to the underlying workflow structure. Wong and Gibbons [273] propose a relative-timed semantic model for BPMN by introducing the notion of relative time in the form of delays to their model. The semantics is defined in the language of Communicating Sequential Processes (CSP). The annotated process model allows behavioural properties of BPMN diagrams to be mechanically verified. Koliadis et al. [144] propose an approach to analyse change against high-level models of the organization. Semantic EPC [243] by Thomas and Fellmann is a semantic extension of event-driven process chains.

A number of proposals seek to leverage semantics in assisting business analysts and process designers model business processes. Born and Dörr [37] extend the SAP Research modeling tool Maestro for BPMN. ProcessSEER [123] by Hinge et al. provides a user-friendly framework for analysts to explicitly annotate business process models and automatically computes the post-conditions associated with tasks selected by the user. Hornung et al. [129] propose a recommender system that suggests a list of correct and fitting process fragments for an edited business process model, which can be used to complete the process model being edited.

Mining process execution data: A large body of work on process mining algorithms—such as the alpha algorithm [249], heuristic miner [267], and fuzzy miner [105]—

offer the capability to extract the structure of the process model. Unlike this body of work, the focus is only on mining task post-conditions.

The research integrates the two approaches of: (1) mining historical data to discover useful process information and (2) adding semantics to business process models, to obtain richer descriptions of business process designs which in turn can be used to support a variety of process analysis tasks such as compliance checking and resolution [86], goal satisfaction analysis [205] and so on.

3.9 Summary

This chapter offers an approach to mining business process task post-conditions from process and state changes events in process execution histories. Specifying post-conditions is notoriously difficult for process analysts, yet these post-conditions are critical to a variety of process analysis tasks such as process compliance management [86], goal satisfaction analysis [205], change management [144], enterprise process architectures [145] and the management of the business process life cycle [146]. The proposal involves the innovative use of sequential pattern mining on event logs. The proposal also leverages event data and the state update notion implicit in process execution to achieve a sophisticated validation technique, which in turn supports an abductive approach to the repair of the mined post-conditions. The empirical evaluation suggests that the results are generally reliable, pointing to prospects for further development of techniques that leverage these post-conditions in semantic analysis.

Acknowledgment: This section has been published in *Data & Knowledge Engineering Journal* vol. 109. This was a collaboration work between the author (Metta Santiputri) with the author's supervisors (Prof. Aditya Ghose and Dr. Hoa Khanh Dam). The author's supervisors provided the guidance specifically in the direction of the work and the final reviews before the publication. All the other works were

performed by the author.

Chapter 4

Requirement model extraction

One of the knowledge driver in an enterprise is the requirement models. However, the connection between requirements and data has been largely under-explored. Yet the growing volume of data, the ability to access large-scale sensor instrumentation and the availability of "big data" tools has thrown up significant opportunities for developing a new generation of data-driven requirements engineering (RE) tools. This chapter provides a method to leverage process execution histories in the form of process logs and message logs to mine the requirements model represented in i* framework.

Section 4.1 provides the introduction to the approach. The first part of the method, the DE technique, is presented in Section 4.2, while the second part, the TDCE technique, is presented in Section 4.3. Section 4.4 provides an evaluation for our method. We summarize this chapter in Section 4.6.

4.1 Introduction

The connection between requirements and data has been largely under-explored. Yet the growing ubiquity of data, the ability to access large-scale sensor instrumentation and the availability of "big data" tools has thrown up significant opportunities for

developing a new generation of data-driven requirements engineering (RE) tools. These opportunities come in many forms.

First, data can alleviate the well-known challenges associated with requirements acquisition/elicitation [174]. Organizations are often unable to leverage the benefits of conceptual modeling and the principle use of enterprise architecture because of the (often steep) investment required. The phenomenon is an instance of the *knowledge acquisition bottleneck* - a problem with an even longer pedigree [29]. Conceptual modeling is a time consuming human task of considerable complexity. In this chapter, we argue that developing a capability to “mine” requirements from data can pay rich dividends. Earlier work [87] suggests that tools that extract “snippets” of models (or *proto-models*) by mining legacy text and model artefacts (these latter being in different notations) can significantly improve modeler productivity (with some empirical results pointing to about a two-thirds reduction in modeler effort).

Second, data-driven requirements monitoring provides the ability to improve the quality of requirements specifications, which in turn lead to improvements in the quality of the systems delivered. Execution data provides the basis for extracting requirements (which may be viewed as abstract descriptions of the data that these are mined from). Deviations between the mined requirements and those originally specified by stakeholders can flag problems. Similarly, we might cluster data associated with the particularly “desirable” (as determined by stakeholders) parts of execution histories, and extract requirements from these. The requirements thus obtained would represent more accurate encodings of stakeholder intent.

Third, clustering data associated with “undesirable” instances of execution histories (again, determined by stakeholders) can help us mine requirements anti-patterns. Within the context of a given RE exercise, these anti-patterns would identify “no-go” areas (i.e., requirements that lead to undesirable consequences).

Finally, the ability to establish an online, real-time correlation between requirements and data can help us use requirements models as dashboards.

What we have outlined above are effectively four distinct hypotheses about how data (specifically, behaviour histories) can deliver value in the requirements engineering exercise. In this chapter, we put the first two of these hypotheses to the test. We focus on a well-regarded early-phase requirements modeling language of long standing - the i^* notation [281]. The use of i^* makes the case for data-driven requirements engineering more compelling, for several reasons. i^* is particularly effective in modeling high-level strategic requirements, and also supports distributed goal modeling. Consequently i^* serves as a natural representation of complex organizational contexts. This chapter presents some initial steps toward an evaluation, by devising and evaluating techniques that permit us to (partially) "mine" i^* models from execution data. We restrict our attention to mining dependencies and the tasks within the depender and dependee actors that each dependency is associated with. We present two techniques: the *Dependency Extraction (DE)* technique which mines dependencies from message logs and the *Task-Dependency Correlation Extraction (TDCE)* technique which mines the tasks/ goals in an i^* SR model that are associated with each dependency from process logs.

Our focus on message logs and process logs is realistic. Message logs are routinely maintained within the enterprise context. Sometimes, these manifest as email repositories, but our current work does not address the deployment of sophisticated NLP techniques that would be required to mine these. Instead, we use an abstract, generalized messaging format that resembles a number of industry-standard electronic messaging formats such as RosettaNet [198], ebXML [89] and a host of EDI formats. These are clearly easier to mine than natural language message logs, but nonetheless provide a useful basis for a proof-of-concept tool. Process logs are also routinely main-

tained by firms. A variety of business process management tools as well as bespoke process logging tools can be leveraged to obtain these. Unlike process mining tools, however, we do not seek to extract process designs from process logs, but instead mine for patterns of task activations that point to the existence of a dependency.

We also simplify matters by assuming that the only i^* models of interest are those that involve only goal dependencies. This avoids defining separate extraction techniques for each distinct type of dependency. This is also not an unreasonable assumption. A task dependency may be viewed as a goal dependency where the goal is to execute a task. A resource dependency may be viewed as a goal dependency where the goal is to obtain the relevant resource. We keep softgoals entirely outside the purview of our current discussion (but these represent an important direction for future work).

These techniques only support the mining of partial i^* models, specifically inter-actor dependencies, and tasks/goals associated with each dependency. We present two different evaluations of these techniques. First, we evaluate their effectiveness (in terms of precision and recall) in mining partial i^* models from behaviour histories that simulate the execution of an initial complete i^* model. Second, we validate the hypothesis that it is possible to generate better quality (i.e., more accurate) models by mining behaviour histories of imperfect “as-is” contexts that have been filtered by stakeholders (to remove behaviour traces that are undesirable). Much more detailed evaluation is possible, and is the focus of future work, but our preliminary results are encouraging.

4.2 The Dependency Extraction (DE) technique

The Dependency Extraction (DE) technique is intended to mine message logs for i^* dependencies, and is based on the following intuitive observations. All dependencies manifest themselves in messages, such as a request from the depender to the dependee

at the creation of a dependency, and a message in the reverse direction when the dependency is fulfilled. Hence, a message log that maintains a record of all messages (over a certain period) between the actors of interest represents a rich repository of clues about these dependencies. Message logs are ubiquitous. A corporate email repository can be viewed as a message log, although the messages are entirely unstructured. In many cases, messages are structured such as in a variety of Electronic Data Exchange (EDI) languages, or in more recent standards such as RosettaNet and ebXML. Our current approach assumes a structured message log. We use a generalized message format in our evaluation, inspired by (and representing the common core of) the messaging standards discussed above. For our purposes, a message log is a sequence of messages consisting, at a minimum, the following components:

- An *interaction ID*, which is used to identify a conversation or *interaction*, but not an individual message.
- *Sender ID*
- *Receiver ID*
- A *timestamp* which describes the time when a message is sent or received (we assume message transmission to be instantaneous).
- A *message type*, which would involve types such as requests, responses etc.
- A *message payload*, consisting of the semantic content of the message (which might be imperative or descriptive or a variety of other speech acts).

In the spirit of RosettaNet, we assume that all messages that involve responses to an initial message (that starts a conversation, such as a service request) refer to a unique ID generated by the initial message. We shall refer to the set of all messages pertaining to such a unique ID as an *interaction*, the unique ID as the *interaction ID*.

Given the availability of unique interaction IDs, it is easy to extract complete interactions from a noisy message log where multiple interactions might be interleaved. Our next task is to extract the goal (e.g., the service request or product order) that is the object of the conversation. Goals are often represented in natural language using verb phrases. The information extraction techniques used for extracting verb phrases admit considerable complexity. For the purposes of our proof-of-concept evaluation, we assume an even simpler textual format, consisting of ⟨verb, noun⟩ pairs (such as buy book, supply product, assess claim etc.). Our technique for extracting these is as follows:

- We extract the set of all ⟨verb, noun⟩ pairs that appear in a given interaction.
- We annotate each element of this set with the number of messages that it appears in.
- We identify the element with the highest frequency and if it passes the threshold $k_{message}$, it referred to as the *goal designator* associated with the dependency.

We use the following procedure to identify dependencies from message logs:

- We partition the set of all interactions extracted from a message log into sets that share the same goal designator.
- We assume that a *significance threshold* $k_{interaction}$ is provided by the user. If a cluster of interactions (with the same goal designator) represents $k_{interaction}\%$ or higher of the set of all interactions, we treat that cluster as *significant* and indicative of a dependency.

4.3 The Task-Dependency Correlation Extraction (TDCE) technique

In this section, we will present the the Task-Dependency Correlation Extraction (TDCE) technique that identifies the task in the depender actor and the task in the dependee actor that are associated with a given dependency. This information will be mined from the process logs.

The mining of task dependency correlations starts with process logs from different actors where the execution of each actor generates a distinct log. It comprises of a list of tasks executed by an actor over time. Multiple process logs from different actors could be combined into one process log shown in the example Table 4.1. This process log lists all tasks executed by all actors (either as the depender or as the dependee). By examining this log, we can observe that when actor i activates task a at time t_x , within some n units of time in the future, at time $t_x + n$, actor j activates task b . When this particular pattern of task activation become frequent (or satisfy certain threshold), then there is an indication of a dependency between task a in actor i as the depender and task b in actor j as the dependee.

Each entry in the process log consists of:

- a *taskID*, which is used to identify certain task in an actor.
- a *timestamp* which describes the time when a task is activated by the actor.

The timestamp of the first entry is t_0 which indicated the initial time and the timestamp of subsequent entries is increased each by one unit time.

The list of these entries will comprise a process log.

In this example, from the first row, we can observe that at the initial time t_0 , there are three different actors, each of which activates a task, i.e. actor A activates task a_0 ,

actor B activates task b_1 , and actor C activates task c_0 . In the second row, the time is increased by one time unit to become $t_0 + 1$. At time $t_0 + 1$, actor A does nothing, actor B activates task b_4 , and actor c activates task c_1 , and so on.

In the process log shown in the example above, we can determine which of these task that has an indication of dependency by examining the task sequence pattern that occurs in the process log. We adapt the GSP (Generalised Sequential Patterns) algorithm in order to mine this sequence pattern. It generates pairs of tasks sequences where the first task is from the depender actor and the second task is from the dependee actor. For example a pair $\langle\langle a_0 \rangle\rangle \langle\langle c_2 \rangle\rangle$ means that there is a pattern between $\langle\langle a_0 \rangle\rangle$ and $\langle\langle c_2 \rangle\rangle$ which indicates that there is a dependency between those two tasks with task $\langle\langle a_0 \rangle\rangle$ as the depender and task $\langle\langle c_2 \rangle\rangle$ as the dependee. Then it will determine how frequent this pattern is in the log by counting the number of occurrence of each pair. This number of occurrence of each pair is called its support and the predetermined threshold is the minimum support.

The GSP (Generalised Sequential Patterns) algorithm was proposed by Srikant and Agrawal (1996). The algorithm takes as input a process log which consists of set of tasks ordered by time. It finds all sequences of tasks whose support is greater than the minimum support threshold specified by the user. In addition to the minimum support threshold, there are timing constraints that must satisfied, namely the maxi-

Table 4.1: Example of process log

Time	Task name		
	actor A	actor B	actor C
t_0	a_0	b_1	c_0
$t_0 + 1$	—	b_3	c_1
$t_0 + 2$	a_0	b_2	c_2
$t_0 + 3$	a_2	b_0	—
$t_0 + 4$	a_1	b_1	c_3
$t_0 + 5$	a_0	b_3	c_2

Table 4.2: Candidate generation

Frequent sequences of length 1	Candidates of length 2	
	after join	after pruning
$\langle\langle a_0 \rangle\rangle$	$\langle\langle a_0(b_1) \rangle\rangle, \langle\langle a_0(b_3) \rangle\rangle, \langle\langle a_0(c_2) \rangle\rangle$	$\langle\langle a_0(c_2) \rangle\rangle$
$\langle\langle b_1 \rangle\rangle$	$\langle\langle b_1(a_0) \rangle\rangle, \langle\langle b_1(b_3) \rangle\rangle, \langle\langle b_1(c_2) \rangle\rangle$	$\langle\langle b_1(a_0) \rangle\rangle, \langle\langle b_1(c_2) \rangle\rangle$
$\langle\langle b_3 \rangle\rangle$	$\langle\langle b_3(a_0) \rangle\rangle, \langle\langle b_3(b_1) \rangle\rangle, \langle\langle b_3(c_2) \rangle\rangle$	–
$\langle\langle c_2 \rangle\rangle$	$\langle\langle c_2(a_0) \rangle\rangle, \langle\langle c_2(b_1) \rangle\rangle, \langle\langle c_2(b_3) \rangle\rangle$	–

imum time difference between the earliest and latest task activation and the minimum and maximum gaps between adjacent task. We leverage this algorithm, providing as input a process log ordered by time and obtaining as output all sequential patterns in the log.

The algorithm makes multiple passes in the log. The initial constraint for this part is that we are only interested in the result that consists of two tasks because we only want to discover any dependency that occurs between two tasks. Therefore we limit the pass to $k = 2$. The first pass of the algorithm finds all the sequence with single task in it along with their occurrence count (support). The output is 1-task long sequences or L_1 . On the second pass, the algorithm generates 2-tasks-long candidate sequences C_2 with L_1 as its seed. This is motivated by the fact that for a sequence to be frequent, all of its subsequences must also be frequent. As the support counts is determined, the sequences with support greater than the determined threshold are included in L_2 .

There are two main phases that are explained in detail below, in terms of how candidates are generated and how their support are counted.

1. *Join phase.* In this phase, we generates all candidates starting from candidate of length 1. The process is straightforward as all the tasks that were in the process log are placed in this set of candidates L_1 . To generate the candidates of length 2, L_2 , a task from L_1 is joined with another that is also in the L_1 . Assume that i

and j are tasks belong to L_1 , then task j is added to i . But for all candidates of length 2, there is one more constraint i.e. any two tasks in a dependency must not happens in the same time, therefore any dependency between two tasks that activates in the same time must be excluded from the result. Initially task j should be added as an task-set ($\langle\langle i j \rangle\rangle$) and as a separate task ($\langle\langle i \rangle\rangle(j)$), but because of this constraint, we only add j as a separate task.

In our example from Table 4.1, we start with all candidates of length 1, L_1 . It would consists of $\langle\langle a_0 \rangle\rangle$, $\langle\langle a_1 \rangle\rangle$, $\langle\langle a_2 \rangle\rangle$, $\langle\langle b_0 \rangle\rangle$, $\langle\langle b_1 \rangle\rangle$, $\langle\langle b_2 \rangle\rangle$, $\langle\langle b_3 \rangle\rangle$, $\langle\langle c_0 \rangle\rangle$, $\langle\langle c_1 \rangle\rangle$, $\langle\langle c_2 \rangle\rangle$, and $\langle\langle c_3 \rangle\rangle$.

Next we need to eliminate these candidates according to the value of minimum support in the prune phase.

2. *Prune phase.* We must eliminate candidates according to our constraints:

- (a) Since any dependency must be occurs between two different tasks from two different actors, any pattern that contains tasks from the same actor must be excluded from the result.
- (b) All the candidates with less than minimum support is excluded from the result.

Note that constraint (a) only applied to 2-task-long candidates and does not applied to 1-task-long candidate. On the other hand, constraint (b) applied for both cases.

Back to our example, we have generated L_1 , and now all tasks in the L_1 must be examined against constraint (b), which is compared to the minimum support value. For this example, we set the minimum support as 2, which means that any task or set of task to be classified as frequent, it must occurs minimum two times. In the log in Table 4.1, for actor A there are three different tasks (a_0 , a_1 ,

and a_2). The support for each of these task are 3, 1, and 1 respectively. Because support for a_1 and a_2 are less than the minimum support, they do not included in the sequence of 1-task. We repeat this for all task and the result is shown in the first column of Table 4.2.

We continue to search for all candidates of length 2 by repeating the join and prune phases. This step is illustrated in Table 4.2. In the join phase, we start with the first candidate $\langle(a_0)\rangle$ in the first column. Thus, all sequences of form $\langle(a_0)(X)\rangle$, where X is any task, are searched. Remember that we do not search for $\langle(a_0, X)\rangle$ because it implies that the two tasks occurs at the same time. By combining $\langle(a_0)\rangle$ with the second candidate $\langle(b_1)\rangle$, we find 2-task-candidate $\langle(a_0)(b_1)\rangle$. Similar procedure is repeated for all sequences of the first column. All 2-task-long candidate sequences generated in the join phase are shown in the second column.

Next, according to constraint (a) in the pruning phase, we begin by determining whether the two events are executed by the same actor, and if they do, then the sequence is eliminated. For example in the second row, sequence $\langle(b_1)(b_3)\rangle$, both tasks are executed by the same actor, namely actor B , therefore the sequence is eliminated by constraint (a). Thus the remaining candidates for the second row are $\langle(b_1)(a_0)\rangle$ and $\langle(b_1)(c_2)\rangle$. The same also applies to sequence $\langle(b_3)(b_1)\rangle$ in the third row. We then determined the support count for each of the remaining candidates. The first candidate $\langle(a_0)(b_1)\rangle$ has support count 1, which is less than the minimum support - it is therefore eliminated. Next candidate, $\langle(a_0)(b_3)\rangle$, also has support count of 1, and is also eliminated. Candidate $\langle(a_0)(c_2)\rangle$ has support count of 2 - it is therefore included in the result. In the second row, we have two remaining candidates, $\langle(b_1)(a_0)\rangle$, and $\langle(b_1)(c_2)\rangle$. Both have a support count of 2 and are thus included in the result. We repeat this procedure for the rest of the candidates. The result is shown in the third column of Table 4.2.

The result patterns of this algorithm are all sequential pattern that occurs between two tasks in the process log. For our process log example in Table 4.1, these are $\langle(a_0)(c_2)\rangle$, $\langle(b_1)(a_0)\rangle$, and $\langle(b_1)(c_2)\rangle$.

4.4 Evaluation

The purpose of the evaluation exercise is to establish the following:

- The Dependency Extraction (DE) technique and the Task-Dependency Correlation Extraction (TDCE) technique generate reasonably reliable results.
- Both these techniques can be leveraged to improve the quality of i^* models (assessed in terms of how closely a model corresponds to the "ideal" model, and hence to the reality being modeled) by leveraging user tagging (or filtering) of the logs recording the behaviour of an "as-is" system or process that the target system is intended to replace. Since i^* is also particularly effective as a domain modeling tool, an improvement in model quality might also entail obtaining a better representation of the context in which the target system is to be situated (or even more generally, a better model of the organizational context).

Our evaluation involved the generation of simulated behaviour histories (message logs plus process logs) given an i^* model. To achieve this, we randomly executed these models, the sense described below. For each distinct dependency in an i^* model, we generated a large number of interactions (specific numbers in the following subsections), with configurable levels of noise (thus we had noisy messages within interactions, and we had entirely noisy interactions that would not point to any reasonable goal dependency). The non-noise components involves messages and interactions that were deliberately constructed to conform to the i^* model at hand. The sum total of these interactions provided the message log that we mined. We similarly generated

process logs by randomly selecting tasks/goals from the i^* model and allocating random timestamps to them. We, however, ensured that each dependency in the model was reflected at least once in the process log. In other words, if there was a dependency relating task a_i in actor A to task b_j in actor B in the model, we would ensure that the process log contained at least one entry for task b_j at a time point after that for task a_i .

4.4.1 Evaluation of DE technique

We started with the i^* model shown in Figure 4.1, originally used in [281]. The model consists of 3 actors and 6 dependencies.

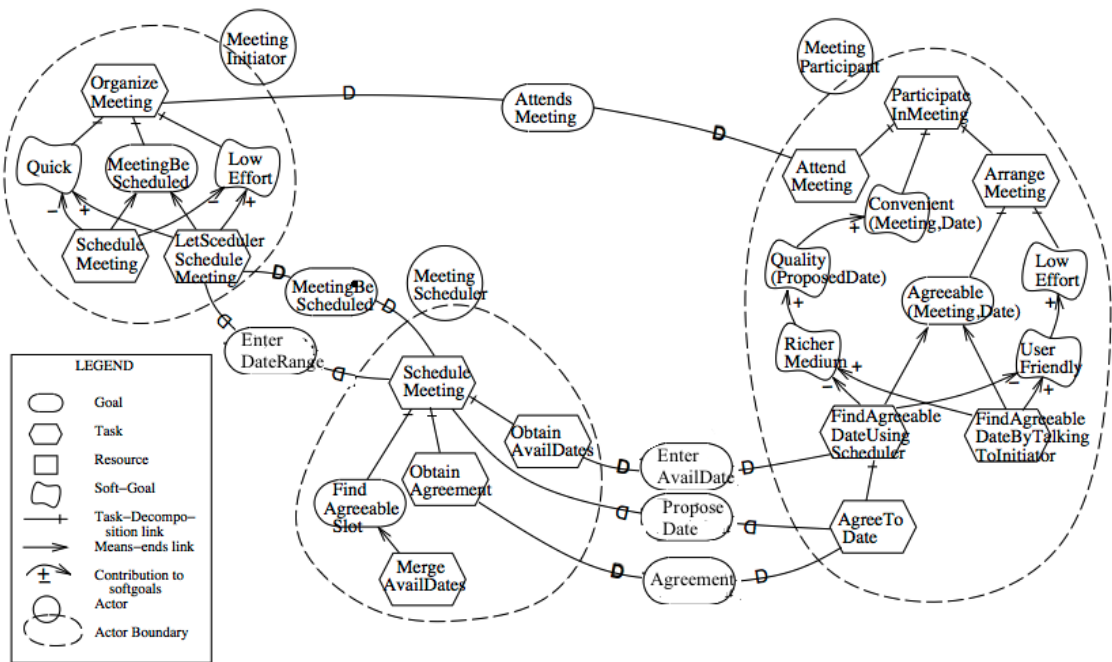


Figure 4.1: An i^* SR model for a meeting scheduler system (adapted from [281])

To simplify the process of obtaining *goal designator*, we assume that it consists of $\langle \text{verb}, \text{noun} \rangle$ pairs. When processing the payload, we extracted the pair by getting the first $\langle \text{verb} \rangle$ in the payload and the first $\langle \text{noun} \rangle$ following said verb. We used the Stanford Log-linear Part-Of-Speech Tagger v3.2.0 [103] for tagging message payloads. The

tagger takes a sentence such as *This is a sample sentence and assign parts of speech, e.g., noun verb, adjective, etc, like so* This_DT is_VBZ a_DT sample_NN sentence_NN. These tags conform to the Penn Treebank Tagset [165] where tag starting with *V* is verb and tag *N* is noun. So we can parse our payload with these tags to find the ⟨verb, noun⟩ pattern.

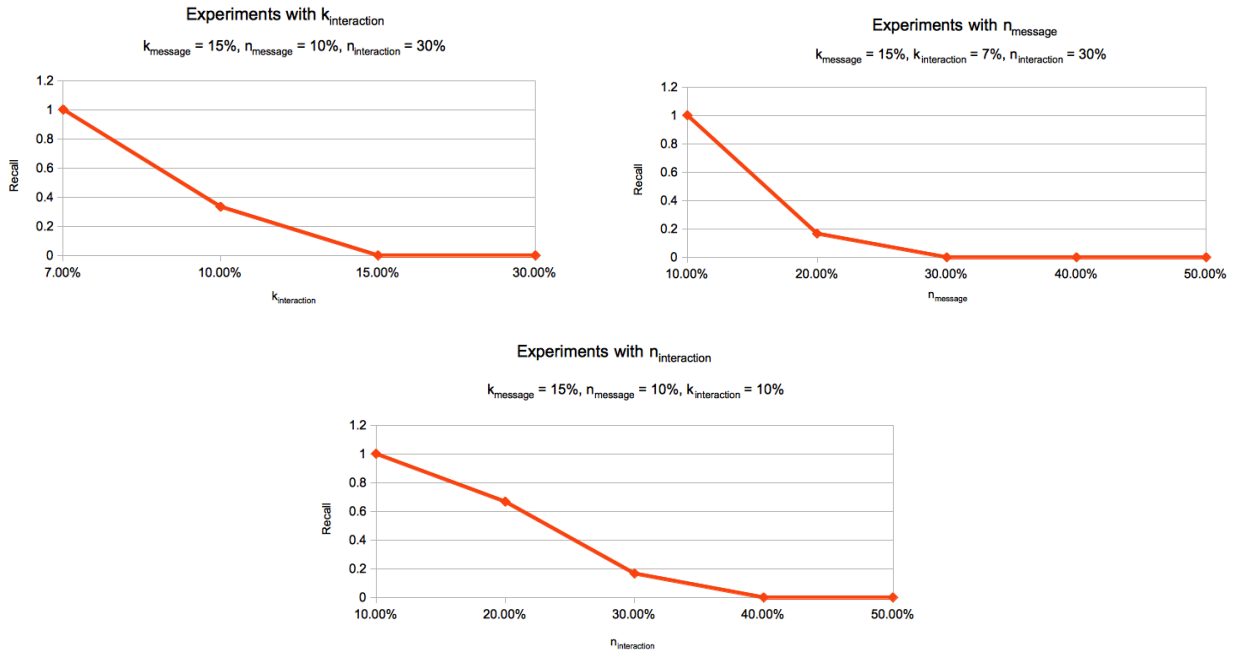
We performed experiments with 4 parameters. $n_{message}$ describes the proportion of noise messages in the complete message log (all non-noise messages permitted the extraction of the correct goal designator). $n_{interaction}$ describes the proportion of noisy interaction in the set of all interactions in the message log, where a noisy interaction is one which does not lead to any single identifiable goal designator. $k_{message}$ and $k_{interaction}$ are as defined in Section 4. We initially created a message log with no noise (i.e., neither noisy messages nor noisy interactions) consisting of 7381 interactions. Setting $k_{interaction} = 10\%$ and $k_{message} = 10\%$, we were able to extract all of the dependencies, as shown in Table 4.3.

Table 4.3: Controlled Environment $k_{interaction} = 10\%$ and $k_{message} = 10\%$

Depender	Dependee	Goal Designator	Interaction Percentage
Meeting Scheduler	Meeting ParticipantA	Enter AvailDates	15.13%
Meeting Scheduler	Meeting ParticipantA	Agreement	18.02%
Meeting Initiator	Meeting ParticipantA	Attends Meeting	17.18%
Meeting Scheduler	Meeting ParticipantA	Propose Date	15.76%
Meeting Initiator	Meeting Scheduler	Meeting BeScheduled	17.55%
Meeting Initiator	Meeting Scheduler	Enter DateRange	16.37%
Total interaction			7381

We plotted the performance of the DE technique (in terms of recall - there were no false positives and hence precision was always 1) against each of these parameters. We show 3 of the 4 results below (the final graph was omitted due to space considerations).

As expected, recall decreases as the amount of noise increases (i.e., as $n_{message}$ and $n_{interaction}$ increase). Similarly, recall decreases as we get more selective in identifying dependencies (i.e., as $k_{interaction}$ increases). These results were generated using message logs that were between 25000 to 30000 messages long, with about 2000 interactions. The results were consistently the same.



4.4.2 Evaluation of TDCE technique

Given a set of tasks as the input of our tool, we use two sets: the set of expected dependencies which were in the input model and the set of dependencies actually discovered in the result. Recall is defined by the number of correct dependencies discovered by our technique divided by the total number of expected dependencies in the result.

In this evaluation, we execute the input model including all of the dependencies. The task activation log will be created with the depender, dependee, the tasks and the dependencies from the input i^* model as the expected result along with other tasks

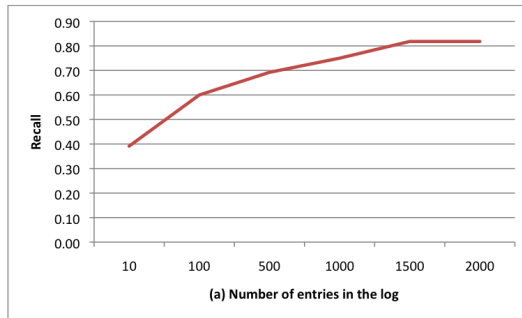


Figure 4.2: Precision relative to log size

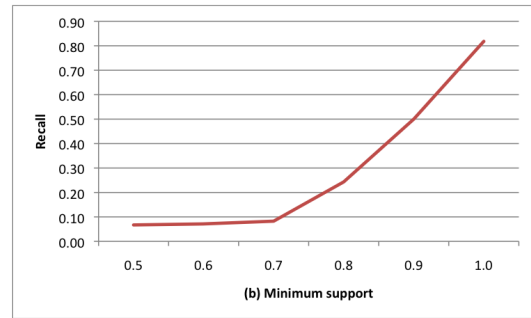


Figure 4.3: Precision relative to min support

from the input model as noise. The task activation log for each actor was created randomly but we deliberately input the execution of every dependencies. Then we ran this log against our tool.

There are two inputs that we will use as variable in this evaluation of the TDCE technique: (a) the number of entries in the process log; and (b) the minimum support. We performed separate experiments for each of those variable by varying the value of one variable and keeping the other fixed.

From the result, the precisions is 1.0 indicating that every dependency in the expected result set is discovered. This can be explained by the fact that if there is a dependency between two tasks, there are frequent patterns between those two tasks in the log and it will be detected.

On the other hand, in addition to the expected dependencies, there are other dependencies that were discovered in the actual result but were not in the input model. This might happen because one factor that might affects the result is the interleaved of the entries in the process log. For example lets assume that we have a log consisting of two actors (let say actor A and actor B) with one dependency between these two actors (between task a_0 of actor A and task b_0 of actor B). Since the algorithm will find all the sequence in the log, there can be two different dependency discovered depends on the order of the entry in the log.

The minimum support count for our evaluation in the first scenario is fix at 1.0

which means that support count = 1.0 * the number of execution. For example if we execute 10 times, then the support count is 10, so that any pattern that occurs 10 times or more is included in the result. For the second scenario, we use a process log with 2000 entries. The two graphs in Figure 4.2 and 4.3 show the recall of our technique in two different scenarios. As can be seen in Figure 4.2, with small log, the recall is not very accurate (given just ten entries, recall is only 0.39). But with larger log, the recall is increasing up to 0.82. Recall remains relatively constant at 0.82 when the log reach 1500 entries or more. While in Figure 4.3, the higher minimum support will give more accurate result. For example, given the support of 1.0, recall reach up to 0.82 but recall drop to 0.5 when support is set to 0.9 and keep decreasing until under 0.1 with support of 0.7 or less. Hence the result suggest that both the number of the entries in the process log and the minimum support are very influential to the accuracy of the result. Therefore if we want to get more accurate result, we can do two things, either increase the volume of the data or increase the minimum support count.

Since in this evaluation we artificially created the process log and deliberately executed all dependencies in the input model, we acknowledge that they would not be representative for all possibilities of process log in practice. For instance, there might be a case where a dependency in the input model was not executed at all or was executed but in a number of times which was lower that the minimal support. In such cases, our technique might not be able to detect it.

Another limitation of this technique involves settings where multiple dependencies exist between the same pair of actors. Our current approach works well if we have a guarantee that only one dependency would exist between a given pair of actors. Thus, when we determine that a pair of tasks are related via a dependency, we are able to leverage the DE technique to identify what the goal designator for that dependency is.

In the case of multiple dependencies between the same two actors, the DE technique would suggest multiple goal designators, while the TDCE technique would suggest multiple task pairs, but we would not have the wherewithal to associate the task pairs with the goal dependencies identified by the DE technique.

4.4.3 Improving requirements quality: Evaluation

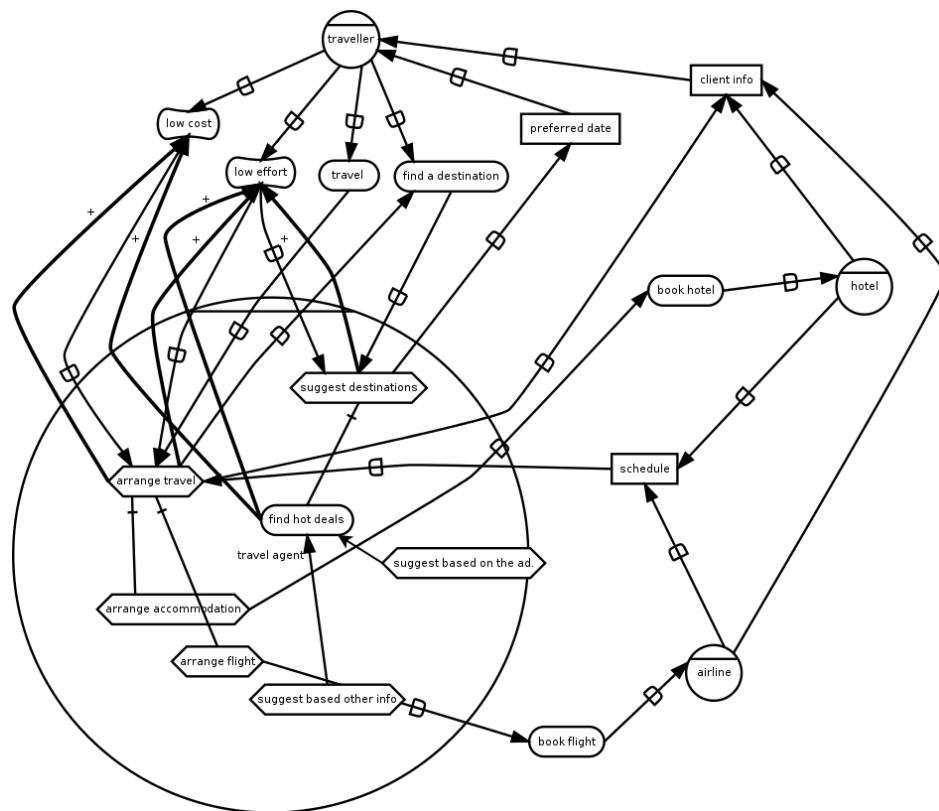


Figure 4.4: Model from user

A key contribution of this work is the ability to achieve data-driven improvements in the quality of requirements models. There are two approaches to this that we explore. In the first, we explore settings where the user is able to describe the ideal *behaviour* (for our purposes, a *behaviour* will be described via a combination of a *message log* and a *process log*) of the system in question. We extract models from these idealized behaviours, as opposed to the noisy behaviours that have been the focus of

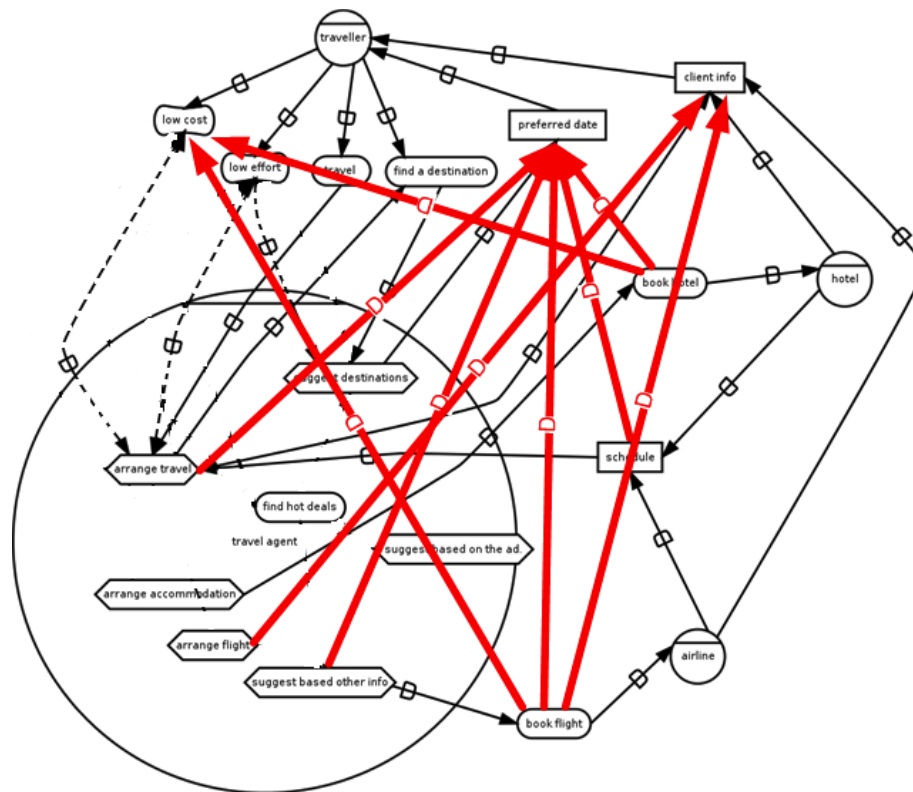


Figure 4.5: Extracted model-1

the previous parts of the evaluation. In the second approach, we explore settings where the requirements engineering exercise is conducted in the context of an existing, “as-is” system or process(es), the behaviour of which we are able to log. The user filters this (potentially imperfect) behaviour generated by the existing system/process (by removing entries from the message and process logs that (in the perception of the user) represent manifestations of imperfect behaviour, and the machinery extracts models from these filtered logs. The evaluation involved a trained i* modeler, who was asked to generate a model (Figure 4.4) which was not revealed to the research team. This model played the role of the “ideal” model against which the quality of the extracted models was evaluated. The quality of an extracted model was evaluated by either: (1) assessing how closely it conformed to the user’s “ideal” model (which was revealed to the research team after the model extraction phase was completed) or (2)

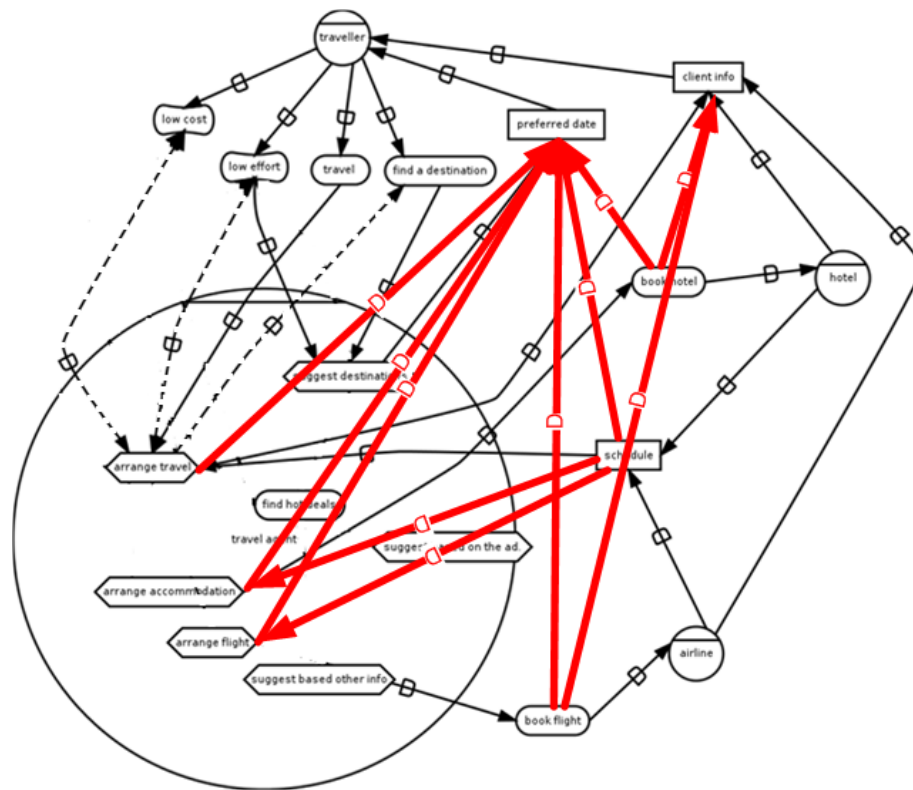


Figure 4.6: Extracted model-2

obtaining input from the user suggesting that some dependencies that existed in the user's intuitive understanding of the domain (and had been manifested in the idealized behaviours supplied by the user, but not in the "ideal" model) has been discovered by the machinery.

User-generated idealized behaviours: With the model in Figure 4.4 in mind, the i^* modeler gave us a message log with 12 entries (each of which was a request message) and a process log with 35 entries. The machinery then extracted a partial i^* model with the following characteristics. Of the 23 dependencies in the original user model, we discovered 19 dependencies (relating the correct pairs of actors and tasks). We also extracted 9 new dependencies that were not part of the user's original model. Figure 4.5 shows the model that was extracted. The bold lines denote false positives and the dashed lines denote false negatives.

User-filtered “as-is” behaviours: In this part of the evaluation, the i^* modeler revealed the idealized model to the research team. This was used to generate 5 behaviours (i.e., 5 sets of \langle message-log, process-log \rangle pairs). The message logs varied in length from 2 to 10 messages per log. The process logs varied in length from 9 to 13 entries. We additionally generated 5 incorrect behaviours, by randomly selecting 1 dependency in each case and randomly changing either the depender or dependee actor, or the source or target task. We then interleaved the 5 correct and 5 incorrect behaviours and presented these to the i^* modeler. Our intent was to simulate the execution of an imperfect system/process, whose behaviour would be represented by the interleaved logs. The i^* modeler was then asked to remove from the message and process logs entries that did not correspond to the intuitions that were represented in the idealized model. We then applied the machinery to extract a partial i^* model from the filtered logs. We discovered 19 of the original 23 dependencies in the idealized model (this was an identical result to the evaluation using user-generated idealized behaviours) and 10 new dependencies. Of these 10 new dependencies, 6 were distinct to the dependencies extracted in the evaluation using user-generated idealized behaviours. The i^* modeler also suggested that 9 of the new dependencies discovered were largely in accord with his intuitions about the domain being modeled, but had not been reflected in the idealized model that he had initially generated. This suggests that this approach can help surface implicit requirements via the filtering of noisy behaviours. The extracted model is shown in Figure 4.6 below. As before, the bold lines denote false positives and the dashed lines denote false negatives.

As discussed in the previous section, the existence of multiple dependencies between the same pair of actors in this model prevented us from correlating the task pairs generated by the TDCE technique with the dependencies generated by the DE technique. The net upshot was that we had several “unnamed” dependencies. Nonetheless, discov-

ering the existence of dependencies, even in the absence of goal designators, provides valuable insights.

Overall, this part of the evaluation suggests that there is merit in the general idea of using this machinery to improve the quality of requirements extracted, although the machinery missed some dependencies and generated some false positives.

4.5 Related works

The idea to utilize history data of process execution as a data source is not new [7, 45, 46, 249]. One research area where this idea grows rapidly is process mining. Process mining is a research discipline that discovers, monitors and improves real processes by extracting knowledge from event logs readily available from today's system [245]. It links the modeled behavior on one hand and the observed behavior on the other hand. There are three types of process mining techniques: discovery, conformance, and enhancement. Process discovery techniques take an event log as input and produces model that best described the behavior observed in the log, mostly to provide insights into what occurs in reality. Conformance checking techniques takes a process model and an event log of the same process as input and compares the observed behavior in the log with the behavior allowed by the model to identify where and when deviations occur, and measure the severity of such deviations. Enhancement techniques take a process model and an event log as input to extend and improve the model with information extracted from the log.

The research reported in this chapter is related in some ways to the process discovery techniques, in that we use process logs as one of several data sources. But unlike process mining, we use the process log to discover the correlation between tasks of different actors in a dependency.

Other area that also exploits this idea is the requirements elicitation process. Men-

dizabal et. al [176] proposed an approach to leverage web server logs as a representation of realistic user behaviors to support non-functional requirements elicitation and prioritization. It uses basic statistic analysis and application usage information to automatically identify the most relevant requirements. Although this approach shares the same intuitions with our work, but in this work, we use different techniques, such as sequential patterns mining, to extract the requirements model.

Requirements elicitation can also be performed by extracting the intentions or goal designator from natural language texts. A body of existing work, such as [38, 72], proposed and implemented this approach. Breaux and Anton [38] implemented the extraction of requirements in terms of rights and obligations in privacy and security setting. They developed a methodology called Semantic Parameterization that is used to map words that describe concepts from simple sentences into first-order predicate logic expressions. Fliedl et al. [72] presented heuristics that support the automatic generation of conceptual dynamic schemas based on textual scenarios. They introduce a conceptual schema model to collect functional requirements and support the association of a conceptual primitive to a given natural language phrase. Our work also uses logs in natural language as a source to extract goal designator, however the evaluation use simpler proof-of-concept implementation.

The Business Intelligence Model (BIM) [127] has the ability to serve as a data-driven dashboard for enterprise. It represents the internal and external business and environment by providing different views of the system and a range of analyses to support managers in making decisions at each level of management. BIM extends the notion of conceptual schema to include strategic objectives, business processes, risks and trends. This ability as enterprise dashboard bears some relation to our proposal in that our work also owns the ability to presents goal or intentions modeling as an internal abstract level view of a system.

Approaches to run-time adaptation also relevant to our work. Several approaches have been proposed including run-time adaptation through advance design tools [18] or dynamic modeling language [102]. Ardagna and Pernici [18] presented an approach to manage adaptation design-time and runtime execution by providing advanced design tools and separating the design from the run-time part in the composition process. Another approach to run-time adaptation by Grossmann et al. [102] proposed to re-configure the design of a service process in case of a run-time failure and to consider new alternative services in the composition. They defined a dynamic modelling language to support the dynamic structure of models and models changes without generating code.

4.6 Summary

In this chapter preliminary evidence was provided to support the hypotheses that data-driven extraction of requirements models can be effective, and that this approach, coupled with user involvement in identifying undesirable behaviour traces, can lead to more accurate models. This evaluation was performed in the context of i^* models, and have further simplified the problem by focusing not on extracting complete i^* models, but only the dependencies and the task dependency correlations.

Acknowledgment: This section has been published in the Tenth Asia-Pacic Conference on Conceptual Modelling (APCCM 2014) at Auckland, New Zealand on January 2014. This was a collaboration work between the author (Metta Santiputri) with the author's supervisors (Prof. Aditya Ghose and Dr. Hoa Khanh Dam) and another colleague (Ayu Saraswati). The author's supervisors provided the guidance specifically in the direction of the work and the final reviews before the publication. The development of the methods and the evaluation were mainly performed by the author, while the implementation of the methods into a working program was divided equally

between the author and the colleague.

Chapter 5

Towards data-driven enterprise architectures: Discovering correlations between the business and application layers in ArchiMate

It is generally acknowledged that enterprise architecture (EA) is a valuable asset to an enterprise [140, 153, 286]. However, organizations are often reluctant to devote their resources to these tasks, either because the investment is considered to be relatively high, or EA is difficult to maintain. Several studies have developed techniques for developing EA using automatic data acquisition, such as [40, 68, 125], but these techniques required significant effort to map the collected data into elements of an EA [68].

In this chapter, an approach to mine the relationships between two layers of enterprise architecture, i.e. business layer and application layer is presented. The method is introduced in Section 5.1. Two logs were leveraged in the method as explained in Section 5.3. The mapping of the data sources to the model is further explained in Section 5.4. Section 5.5 introduces the mining method that was used in the approach.

Four different settings used in the approach are presented in detail in Section 5.6 while the mining of the relationships is presented in Section 5.7. An evaluation to the method is provided in Section 5.8. Section 5.10 presents a summary of this chapter.

5.1 Introduction

An enterprise architecture gives an organization a broad and complete representation of all of its resources, both physical and conceptual, relationships between them, and how they help the enterprise achieve their objectives [153, 286]. With the aid of a well-defined EA, an enterprise may gain insights into opportunities and innovations, as well as determine any needs for change and assess any impact of the proposed change [140, 286].

However, even though the benefits of developing and maintaining an EA are self-evident, organizations often reluctant from devoting their resources in these tasks. Two main reasons according to [219] are as follows. First, the opinion that the investment in developing and maintaining EA is relatively high compares to the benefits, although this mostly has to do with the lack of recognition of the true nature of these benefits. It is a well known fact that a costly investment, both in effort and time, is required during data acquisition to describe the enterprise setting, especially in a complex and diverse environment, for instance when the environment covers both virtual and physical applications and infrastructures in several different locations [12, 68, 140]. Second, in the event of rapid change, EA is difficult to maintain. By the time an EA conclude its development, it is possible that it does not reflect the underlying reality due to constant transformation of the environment [12].

Therefore, mechanism to produce an EA from enterprise data, such as business process execution histories, would be beneficial for an enterprise (albeit a semi-automatic one) as it would be able to mitigate these drawbacks in EA development and mainte-

nance, either in resource, cost or time as discussed above. Ideally, with this mechanism, the resource and cost it took to develop and maintain an EA would be drastically reduced. Furthermore, with reduced time, an EA development would be able to adjust to a rapid changing environment.

Several studies have developed techniques for developing EA using automatic data acquisition, such as [40, 68, 125]. However these techniques required significant effort to map the collected data into elements of an EA [68]. Some techniques such as [40, 125] capture some elements of different layers in an EA, namely application and technology layers, using network scanner to discover the network topology. Next, the resulting elements are mapped into services in both layers. Process mining [249, 268] also provides some assistance to generate the higher abstraction layer, namely the business layer. However, these research only extract elements in each layer and there have been no studies into extracting the relationships between multiple layers.

One of the most prominent EA modeling framework is the ArchiMate framework [240]. It provides a graphical notation to represent entities in an enterprise and their relationships in a hierarchical fashion. An ArchiMate model thus consists of three layers, i.e., the business layer, the application layer, and the technology layer, which corresponds to the abstraction hierarchy in a model, with the business layer as the highest abstraction, the application is in the lower abstraction, and the technology layer being the lowest abstraction level in a model. One of the main advantage in ArchiMate is that it offers a formal notation, with which we are able to leverage in our work, as well as its tool support and widespread adoption. Nonetheless, the conceptual discovery that we identify are also suitable with different frameworks.

The contribution of this chapter is a data-driven method to determine the correlations between multiple layers in an ArchiMate model, namely the business and application layers. The input of the method is a timestamped record of events that

describe the start or end of a process task or function, usually referred as *event log*. We distinguish two types of event log based on their abstraction level, an event log that corresponds to events in the higher level of abstraction (i.e., business layer) and an event log that represents events from the lower abstraction level (i.e., application layer). The proposed method combines time correlation heuristics [69] and frequent closed sequence pattern mining [94, 262, 275] to discover the relationships between components in multiple levels of abstraction. The correlations discovered using our proposed method creates a preliminary (“first-cut”) version of partial EA. Nonetheless, this preliminary version are able to assist enterprise architects to build or develop EA as it will provide the basic model that can be edited accordingly.

5.2 Data-driven enterprise architectures: A general approach

We present a general framework for leveraging enterprise data in the development and management of enterprise architectures. The specific technical proposal (presented over the next several sections) can then be viewed as an instantiation of a specific component of this general framework. The following discussion is not specific to any particular enterprise architecture framework, but draws freely on elements of the Zachman framework [28], TOGAF [9] and Archimate for examples.

The motivations for a data-driven approach to enterprise architectures are addressed first. Given the growing body of sophisticated data analysis tools, a data-driven approach to enterprise architecture can deliver value in a number of ways.

- *The acquisition bottleneck:* It is generally acknowledged that building an enterprise architecture involves significant investments in time and effort. Organizations sometimes choose not to make the investment, possible because their

limited understanding of the long-term value of building and maintaining an enterprise architecture does not suggest a clear return on investment. The problem is a version of the well-known *knowledge acquisition bottleneck*. Enterprise data can be mined to obtain first-cut or draft enterprise architecture models, which can be further refined by enterprise architects to obtain a more accurate representation of the enterprise. The intent is therefore to generate adequate first-cut models that can be easily edited, rather than “perfect” ones. The emphasis is on improving the productivity of enterprise architects, and thus reducing the investment required to acquire enterprise architectures.

- *Characterizing the “to-be” enterprise:* Enterprise data can be curated in ways that can reveal a desired enterprise architecture - i.e., a picture of the enterprise as it was when it operated optimally, and thus a pointer to how the enterprise must look like in a normatively “ideal” mode. This approach has previously been deployed in the context of business processes [7] and enterprise models in the i* language [6]. The approach relies on the extraction from execution histories of desirable executions, i.e., data that reflects the operations of the enterprise when things went well. Extracting enterprise architectures from this subset of past execution histories ensures that we obtain desired enterprise architectures, as opposed to enterprise architectures extracted from data capturing periods of both optimal and sub-optimal enterprise performance. If past execution histories are annotated with performance indicators (these could be of a wide variety, including the full repertoire of key performance indicators that feature in the Balanced Scorecard framework [14]) then automating the extraction of this subset is relatively easy.
- *Anti-patterns:* In the same way that execution histories that reflect periods of optimal enterprise performance provide a good basis for mining desired enterprise

architecture models, execution histories from periods of sub-optimal enterprise performance can form the basis for mining enterprise architectures that describe what not to do (or enterprise configurations to be avoided). These play much the same role as *anti-patterns* [2], and can provide valuable management guidance.

- *Enterprise architecture governance*: Enterprise architecture frameworks such as TOGAF and Archimate provide detailed governance. A data-driven approach to enterprise architecture can support almost the full range of governance functions that these guidelines stipulate. In our discussion below, we will focus on two aspects of enterprise architecture governance where data analytics has, arguable, the most compelling impact: *monitoring/conformance analysis* and *strategic alignment*. We consider monitoring and conformance analysis first. The execution of any artefact that encodes a specification of behaviour can be *monitored* to ensure that actual, observed behaviour satisfies the properties specified. A related term, more commonly used in the context of process mining is *conformance*, where the intent is to ensure that there is no deviation from the normative behaviour specified. An enterprise architecture can also be viewed as a highly abstract specification of enterprise behaviour (but also structure, capabilities etc.). Viewed thus, it also makes sense to analyze *enterprise architecture conformance*, i.e., determine whether actual enterprise behaviour corresponds to the normative behaviour stipulated in the enterprise architecture. A data-driven approach can enable this analysis by extracting an *observed* enterprise architecture from available data, which can then be placed in juxtaposition to the *normative* enterprise architecture to determine if and where deviations or discrepancies have occurred. Similarly, temporal co-occurrence or co-variance patterns relating strategy fulfillment with the execution of elements of an enterprise architecture (in a manner similar to the technical proposal we present in

the next several sections) can provide clues to strategic alignment.

- *The enterprise architecture as a lever for change:* An enterprise architecture is often viewed as a dashboard that provides an abstract collection of *sensors* through which a complex underlying reality can be viewed. Data-driven extraction of enterprise architectures can enable a sensor dashboard view of the enterprise. More interestingly, it can support the use of an enterprise architecture as a collection of *effectors*, i.e., levers through which an enterprise can be managed. One approach is to use enterprise architecture abstractions to define the to-be enterprise, then leverage the mined enterprise architecture-data correlations to determine the specific execution histories/enterprise data one would require such that these, if mined using the same approach, would yield the specified (desired) enterprise architecture.

It is useful consider next the types of enterprise data that can be leveraged for mining enterprise architectures:

- *Process/event logs:* Process or event logs can be a valuable source of information about the *behaviour* of the enterprise. These are the most readily available form of *execution histories* that were referred to in the preceding discussion. In addition to recording *task execution events*, event logs can also record a wider repertoire of events such as object state transitions, the invocation of functions, the opening and closing of application programs, the receipt or despatch of messages and so on. These events provide a rich source of information about enterprise behaviour. Typically, each entry in such logs contains the following: (1) A *case identifier* that describes which process instance a given event is associated with (e.g., the particular claim that an instance of a insurance claim handling process addressed) (2) An *event descriptor* which could simply be a task/activity ID,

or a description of object state transition and (3) A *time-stamp*. That latter is particularly interesting, since it allows us to determine the sequence in which various behaviour milestones were achieved, and analyze temporal co-occurrence and co-variance of the events logged.

- *Resourcing logs*: Information on who did what (and when) is available in a variety of enterprise data sources. For instance, process/event logs, as discussed above, often contain information on the resources used for executing a given task. Task assignment sheets (or their electroning variants), staff rosters and the like also record similar information. Data of this kind can be valuable in determining role models, organizational structures, business architectures and so on.
- *Message logs*: Organizations routinely log phone calls, social media posts, emails and other forms of messages (including messages in various standards such as ebXML). The actual payload of messages, the time messages were sent or received and identities (and organizational unit affiliations) of the senders and recipients can serve as a rich source of information for mining a variety of enterprise architecture elements including strategies/goals, capabilities, roles as well relationships between a range of enterprise architecture elements.
- *Enterprise social media*: Posts on enterprise social media also provide a rich source of information on most of the aspects of enterprise architecture discussed above.
- *Enterprise document repositories*: Organizations typically maintain repositories (and for most modern organizations, in electronic form) of a range of documents such as requirements documents, policy and legal documents, manuals, standard operating procedures, investor- and stakeholder-oriented documents (such as a prospectus), strategy documents and so on. These contain clues to various as-

pects of an enterprise architecture and can be effectively text-mined (although a document itself can sometimes form part of an enterprise architecture).

As with all forms of modeling, defining an enterprise architecture involves specifying concepts/entities (such as a role, a service, a task or a goal) and their inter-relationships (such as the relationship between a role and a task or between elements of a TOGAF business architecture and a TOGAF information systems architecture). In abstract terms, the problem of extracting an enterprise architecture from data reduces to two distinct exercises: (1) Mining concepts/entities and (2) Mining relationships. A simple means of mining lower-level (infrastructure-level and application-level) entities is to use networks scanners or network crawlers (prior work has addressed the acquisition of enterprise architecture elements using such tools [21, 26]). Concepts or entities such as roles, goals/strategies, capabilities, business services, organizational units can also be identified by text mining some of the textual data sources (message logs, document repositories and enterprise social media, for instance) discussed above. Some concepts and entities, such as tasks or capabilities and roles can be directly extracted from data sources such as event logs and resourcing logs. A more complex challenge is to extract relationships between concepts/entities. Clues about such relationships can be mined from textual sources but can also be extracted from event logs by exploiting sequential or co-occurrence relationships between events of various types (our specific technical proposal in this research is an instance of this general approach).

5.3 Event logs

We distinguish two types of event log based on their abstraction level, an event log that corresponds to events in the higher level of abstraction (i.e., business layer) and an event log that represents events from the lower abstraction level (i.e., application layer). For convenience, we refer to the log that records the events in the business

layer as *process log* and the event log from the application layer simply as *event log*. Presumably both type of logs, *process log* and *event log*, are available.

A process log is a set of tuples $\langle timestamp, ID, task, actor \rangle$. The *timestamp* value indicates the time the *task* is executed while *ID* identified the process instance that the task belongs to. The *actor* records the user executing the task. At any given time, there possibly many process instances (of the same process design or distinct process designs) to be executed concurrently. We also define ϵ as the execution time of a task. Event log is identical to process log, however instead of tasks, an event log records events related to the functions. An event log consists of a set of tuples $\langle timestamp, function, source \rangle$ where *timestamp* signify the time when the *function* is started and *source* identify the application, service, or component that executed or logged the function. The process log can be generated using a variety of business process management tools or process logging tools. On the other hand, the event log can be obtained from any operating systems where function calls are recorded. One such example is the Application log which can be accessed from the Microsoft Windows Event Viewer.

5.4 Mapping between logs and layer components

Before we can mine the correlations between the business and application layer, first we need to map the components in the process and event log to the components in the ArchiMate layers.

The mapping between process log and the business layer is clear-cut. A task represents the activity or work (atomic or non-atomic) that needs to be performed by an actor in an organization. In ArchiMate, the actual work that must be performed (by actors and their associated roles) is defined in the business behavior concept. Therefore, the mapping between process log and ArchiMate are as follows: the combination

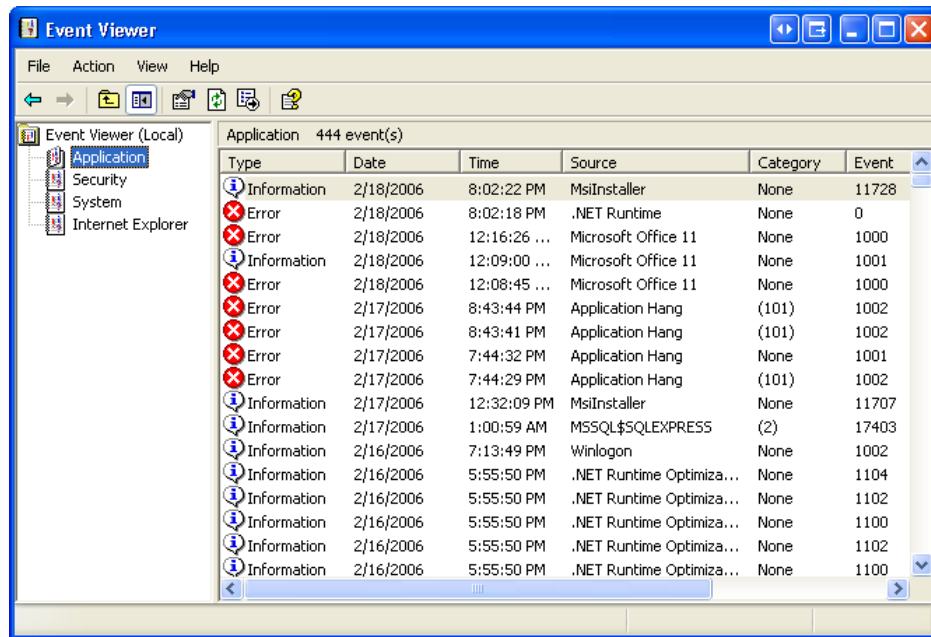


Figure 5.1: Microsoft Windows Event Viewer, an instance of event log in the application layer

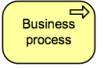
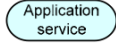
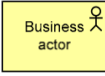

of ID and task is mapped to the business process and the actor who perform the task is mapped to business actor.

The mapping between event log and application layer is not so straightforward. The external view of application layer is represented by the application service where application service serves the business process and exhibit the external behavioral concept of application layer. In the event log, a function is started when there is a request from external user thus it represents the behavior that is accessible to external user which is similar to the external behavioral concept in ArchiMate. Therefore we mapped the function in the event log to the application service in the ArchiMate application layer. Application component in ArchiMate is defined as a modular, deployable, and replaceable part of a software system that encapsulates its behavior and data. In the event log, we recognize this concept as similar to the Source field of the function invocation where the source represents the application, service, or component that the function belongs to. Therefore the source is mapped to the application component in

ArchiMate application layer.

The mapping between the logs and the ArchiMate layers are presented in Table 5.1.

Table 5.1: Mapping Between Logs and Layer Components

Process log	Business layer	Event log	Application layer
ID task		function	
actor		source	

Therefore we can infer the correlation between ArchiMate layers from the relationship between the process log and the event log, more specifically between (ID,task)-combination in the process log and function in the event log.

5.5 Frequent closed sequential pattern

We particularly interested in the sequential pattern mining techniques because the logs consists of sequential events. There is a large volume of research that propose techniques to discover sequential pattern (see Section 2.5.2 for a number of example of established techniques), such as SPADE [290]. They generate all permutation of sequences and count the number of their occurrences in the data set. Only the sequence whose frequency exceed a specified threshold (refer as pre-defined minimum support ($min_{support}$)) is returned.

Similar to sequential pattern algorithms, closed sequential pattern algorithms also generate all permutation of sequence patterns and then examine their frequency against the minimum support value, however only the sequence that do not contain any sub-sequence with the same support will be returned. Few examples of such algorithms are CloSpan [275], BIDE+ [262], ClaSP[94] and CM-ClaSP [74],

Given two pattern, S and S' , both has the same frequency (assumed that this frequency exceeds the minimum support) and S' is a subsequence of S , then both S and S' are considered as frequent sequence, however only S is considered as frequent closed sequence.

Table 5.2: Frequent sequence vs. frequent closed sequence

```
Sequences = {CAABC, ABCB,CABC,BBCA}
Frequent sequences = {A:4, AA:2, AB:4, ABB:2,
ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2,
CABC:2, CAC:2, CB:3, CBC:2, CC:2}
Frequent closed sequences = {AA:2, ABB:2, ABC:4,
CA:3, CABC:2, CB:3}
```

Table 5.2 shows a data set consists of four sequences. Assume that the minimum support is set to 50% i.e., 2. As a result, frequent sequential pattern mining algorithms and frequent closed sequential pattern mining return different set. For instance, both CBC and CABC are considered as frequent sequence, however only sequence CABC is considered as frequent closed sequence, however CBC is not a frequent closed sequence because both has the same support, i.e., 2, and the pattern CBC is a subsequence of CABC.

5.6 Generating joined log

We examine the problem of mining the relationship between the process log and the event log in two different scenarios: (1) a **complete timestamp scenario** where the start and end times of all tasks are available in the process log; and (2) a **partial timestamp scenario** where for each task only the start time is known but not the end time.

We assume that all possible behaviours of task and function fall under these two scenarios. During its execution, a task may required that a set of functions to accom-

plish its operation. Depending on the underlying system, some protocol, such as TCP, may require the function to return an acknowledgment to the task. In this case, the completion of the function can be recorded. However, in the setting where the communication model is connectionless, such as in UDP protocol, these acknowledgment is never exist, thus the completion of a function can not be recorded.

In relation to task execution, we consider two different settings: (1) a **unique task setting** which stipulates that only one task may be executed at any given time; and (2) a **concurrent task setting** admits the possibility of multiple tasks executing at the same time. The latter setting may be of interest if the process being executed admits parallel flows, or if multiple instances (of one or more process designs) are executed at the same time.

In all four possible setting combinations, to determine the correlation between the process log and the event log, we first create a joined log of the form: $\langle\langle p_1, \langle\langle f_{11} \rangle, \dots, \langle f_{1n} \rangle \rangle \rangle, \dots, \langle p_i, \langle\langle f_{i1} \rangle, \dots, \langle f_{im} \rangle \rangle \rangle, \dots, \langle p_j, \langle\langle f_{j1} \rangle, \dots, \langle f_{jk} \rangle \rangle \rangle$ where each $\langle p_i, p_{i+1} \rangle$ pair represents contiguous tasks and each f_{ij} represents the j -th function invoked during task execution time, ϵ_{p_i} . This joined log represents the sequence database provided as input to the frequent closed sequential pattern miner.

The difference between these four settings lies in the generation of the joined log, more specifically on how we define ϵ in each setting. This point is important because any function serves any particular task must be invoked during the execution duration of the corresponding task. To illustrate this point, an example for each setting is presented in Table 5.3 and 5.4.

Let us consider the first example in the complete timestamp-unique task setting presented in Table 5.3(a). There are two process instances with ID ID_1 and ID_2 , each process instance consists of three tasks $P = \{p_1, p_2, p_3\}$. There is also a set of functions $F = \{f_1, f_2, f_3, f_4\}$. The corresponding process log and event log are shown

in Table 5.3(a)(1) and Table 5.3(a)(2). Since in this setting, the process log records the start and end time for task p_i , thus ϵ for any task p_i is the duration time between start and end time. Because all functions that serves task p_i is invoked during ϵ , therefore in this setting, for both process instances, the first sequence is $\langle p_1, \langle f_1, f_2 \rangle \rangle$ because functions $\langle f_1, f_2 \rangle$ are started during the execution period of task p_1 . The second sequence is generated for task p_2 with functions $\langle f_4, f_2 \rangle$ as well as functions $\langle f_2, f_3 \rangle$ with task p_3 . The joined log is shown in Table 5.3(a)(3).

In the case where the complete time assumption is relaxed, we obtain the partial timestamp setting. In this setting, the start time of a task is known, but the end time is not recorded. However, with the unique task setting, it is guaranteed that in any given time, there is only one single task being executed. Therefore, although the end time of task is not recorded, we use the start time of the subsequent task as the surrogate end time, as illustrated in the timeline in Table 5.3(b). With the start and end time of task established, we can determine ϵ of each task and use it to generate the joined log. In the example, the process log and event log are shown in Table Table 5.3(b)(1) and Table 5.3(b)(2), respectively. The joined log from both these logs is presented in Table 5.3(b)(3). We observe that the result joined log is identical with the complete timestamp-unique task setting in the previous example. However, we also observe that since the last task does not have any subsequent task, we must make an assumption. In that case, we use the end of the log as the surrogate end time.

In the concurrent task setting, we admit the possibility of multiple tasks may be performed at the same time. First we consider the complete timestamp scenario. With this scenario, the ϵ of any given task can be established and the functions invoked during ϵ can be determined. The main difference with the unique time setting being that not all function invocations during that duration may be related to that particular task. However, the joined log is generated in identical manner. Looking at the timeline

in Table 5.4(c) and the process log and event log in Table 5.4(c)(1) and Table 5.4(c)(2), we generate the joined log in Table 5.4(c)(3).

In the last setting, i.e., partial timestamp-concurrent task setting, we can not use the same assumption as in the unique task setting where we use the subsequent task start time as the surrogate end time. The end of the process log also acts as the end of all task. The joined log generated using this provision has a cascading pattern as shown in Table 5.4(d)(3). Furthermore, depends on the size of the log, the sequence would potentially infinitely long, however this is unavoidable.

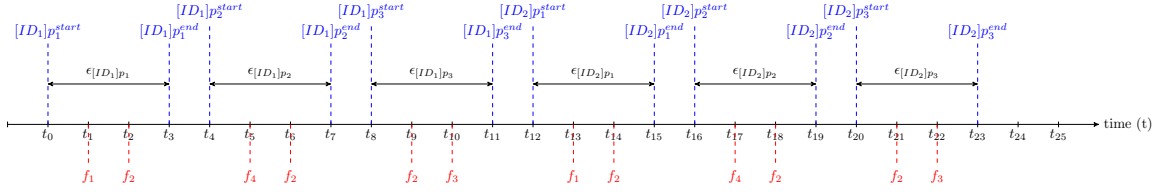
5.7 Mining the sequence patterns

We then use the joined log from the previous section as the input to the frequent closed sequence pattern miner. In our instance, we will apply BIDE+ algorithm, but other similar algorithms could be used instead. The miner then returns the sequence $\langle p_i, \langle f_{i1}, \dots, f_{in} \rangle \rangle$ whose frequency is over the threshold to represent the correlation of task p_i and function sequence $\langle f_{i1}, \dots, f_{in} \rangle$. The threshold ($min_{support}$) is bounded from below by the number of distinct cases in which a specific task prefix p_i occurs.

Back to the settings explained in the previous section, in the first three settings (i.e., complete timestamp-unique task setting, partial timestamp-unique task setting, and complete timestamp-concurrent task setting) we can apply the pattern miner to the joined log directly. As previously mentioned, to guarantee that we pick up the correct function sequence correlated to a particular task, we use the number of distinct cases with the task as prefix as threshold, or in other word, for each task, we set the threshold for all sequences with the that particular task prefix to 100%.

Let us consider our example. In the first joined log in Table 5.3(a)(3), for p_1 , we get sequences $\{\langle p_1, \langle f_1, f_2 \rangle \rangle, \langle p_1, \langle f_1, f_2 \rangle \rangle\}$. Using the pattern miner with 100% threshold, we get the result $\langle p_1, \langle f_1, f_2 \rangle \rangle$, which is interpreted that task p_1 is related to functions

Table 5.3: Unique task setting



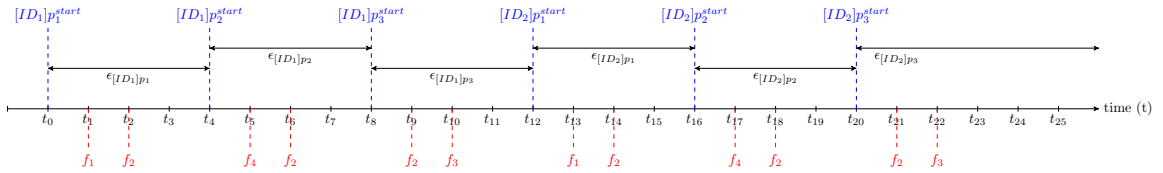
(1) Process log

(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2 \rangle \rangle$
t_3	ID_1	p_1^{end}	t_2	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_4	ID_1	p_2^{start}	t_5	f_4	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
t_7	ID_1	p_2^{end}	t_6	f_2	$\langle p_1, \langle f_1, f_2 \rangle \rangle$
t_8	ID_1	p_3^{start}	t_9	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_{11}	ID_1	p_3^{end}	t_{10}	f_3	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
t_{12}	ID_2	p_1^{start}	t_{13}	f_1	
t_{15}	ID_2	p_1^{end}	t_{14}	f_2	
t_{16}	ID_2	p_2^{start}	t_{17}	f_4	
t_{19}	ID_2	p_2^{end}	t_{18}	f_2	
t_{20}	ID_2	p_3^{start}	t_{21}	f_2	
t_{23}	ID_2	p_3^{end}	t_{12}	f_3	

(a) Complete timestamp-unique task setting



(1) Process log

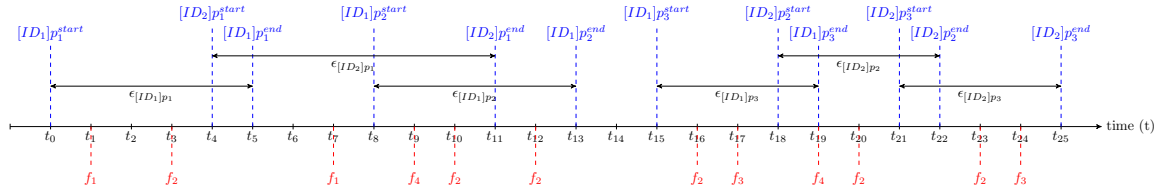
(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2 \rangle \rangle$
t_4	ID_1	p_2^{start}	t_2	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_8	ID_1	p_3^{start}	t_5	f_4	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
t_{12}	ID_2	p_1^{start}	t_6	f_2	$\langle p_1, \langle f_1, f_2 \rangle \rangle$
t_{16}	ID_2	p_2^{start}	t_9	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_{20}	ID_2	p_3^{start}	t_{10}	f_3	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
			t_{13}	f_1	
			t_{14}	f_2	
			t_{17}	f_4	
			t_{18}	f_2	
			t_{21}	f_2	
			t_{22}	f_3	

(b) Partial timestamp-unique task setting

Table 5.4: Concurrent task setting



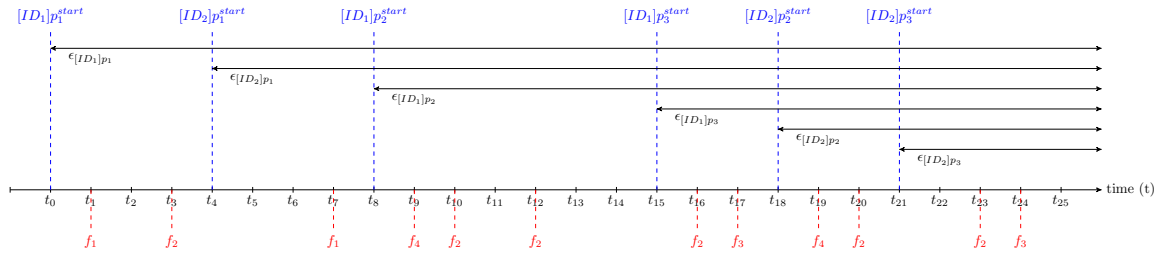
(1) Process log

(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2 \rangle \rangle$
t_4	ID_2	p_1^{start}	t_2	f_2	$\langle p_1, \langle f_1, f_4, f_2 \rangle \rangle$
t_5	ID_1	p_1^{end}	t_7	f_1	$\langle p_2, \langle f_4, f_2, f_2 \rangle \rangle$
t_8	ID_1	p_2^{start}	t_9	f_4	$\langle p_3, \langle f_2, f_3, f_4 \rangle \rangle$
t_{11}	ID_2	p_1^{end}	t_{10}	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_{13}	ID_1	p_2^{end}	t_{12}	f_2	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
t_{15}	ID_1	p_3^{start}	t_{16}	f_2	
t_{18}	ID_2	p_2^{start}	t_{17}	f_3	
t_{19}	ID_1	p_3^{end}	t_{19}	f_4	
t_{21}	ID_2	p_3^{start}	t_{20}	f_2	
t_{22}	ID_2	p_2^{end}	t_{23}	f_2	
t_{25}	ID_2	p_3^{end}	t_{24}	f_3	

(c) Complete timestamp-concurrent task setting



(1) Process log

(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2, f_1, f_4, f_2, f_2, f_2, f_3, f_4, f_2, f_2, f_3 \rangle \rangle$
t_4	ID_2	p_1^{start}	t_2	f_2	$\langle p_1, \langle f_1, f_4, f_2, f_2, f_2, f_3, f_4, f_2, f_2, f_3 \rangle \rangle$
t_8	ID_1	p_2^{start}	t_7	f_1	$\langle p_2, \langle f_4, f_2, f_2, f_2, f_3, f_4, f_2, f_2, f_3 \rangle \rangle$
t_{15}	ID_1	p_3^{start}	t_9	f_4	$\langle p_3, \langle f_2, f_3, f_4, f_2, f_2, f_3 \rangle \rangle$
t_{18}	ID_2	p_2^{start}	t_{10}	f_2	$\langle p_2, \langle f_4, f_2, f_2, f_3 \rangle \rangle$
t_{21}	ID_2	p_3^{start}	t_{12}	f_2	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
			t_{16}	f_2	
			t_{17}	f_3	
			t_{19}	f_4	
			t_{20}	f_2	
			t_{23}	f_2	
			t_{24}	f_3	

(d) Partial timestamp-concurrent task setting

f_1 and f_2 , or in ArchiMate terms, that f_1 and f_2 serve task p_1 . Next, we move to the next task, which is task p_2 , with sequences $\{\langle p_2, \langle f_4, f_2 \rangle \rangle, \langle p_2, \langle f_4, f_2 \rangle \rangle\}$, and we will get the result $\langle p_2, \langle f_4, f_2 \rangle \rangle$. The same method can be applied to the joined table in Table 5.3(b)(3) for the next setting, partial timestamp-unique task setting, and we will get the same result. The joined table in Table 5.4(c)(3) for complete timestamp-concurrent task setting is slightly different, but applying the same method generates the same result.

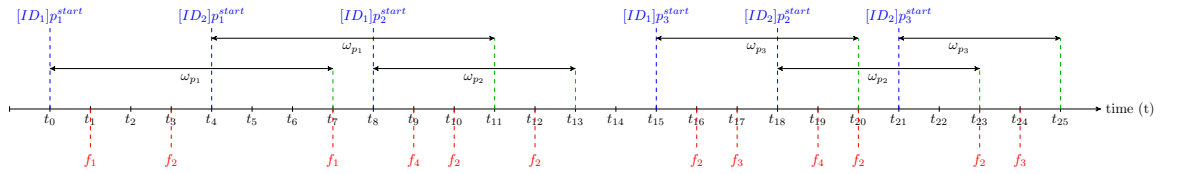
In the last setting, partial timestamp-concurrent task setting, we need to adjust the application of the miner to serve the cascading pattern that emerge as the consequence of the missing end time of the task. In [219], the author proposed an approach to mine the patterns from this type of log. The method suggested to start from the last task (refer as p_{last}) in the log and gradually working backwards. The last task is chosen as the starting point because the last task always has the shortest sequence. All sequences that contains p_{last} as its prefix then used as input to the miner and get a function sequence (refer as $F_{p_{last}}$) as the result. Moving to the previous task in the log, first we have to remove all subsequence $F_{p_{last}}$ from the sequence, times the number of subsequent p_{last} in the process log. The result sequences is then put through the miner to get the function sequence. This method is repeated until we reach the first task in the log. However, we acknowledge that this method rely heavily on the fact that all functions related to tasks other than p_{last} are invoked before the start time of p_{last} and all functions invoked after the start time of p_{last} are only functions related to p_{last} . In this chapter, we propose an improvement of the method by employing two heuristics for mining correlations in the partial timestamp-concurrent task setting.

Window Related Heuristic: We resolve the cascading patterns in the joined table by introducing the sliding window, as introduced by Srikant and Agrawal in [232]. Using sliding window, the sequence contributes to the support only when the difference

in transaction-times is less than the user-specified window-size. In our setting, we define the window that represents all possible function invocations for a particular task (ω_{p_i}) as the maximum of the execution times of all task p_i in the log (i.e., $Max(\epsilon_{p_i})$). So, with respect to p_i^{start} , all functions related to p_i must be invoked during p_i^{start} until $p_i^{start} + \omega_{p_i}$. In turn, any function invoked during during p_i^{start} until $p_i^{start} + \omega_{p_i}$ might be related to p_i (but not necessarily). With the introduction of the sliding window, we can reduce this problem to be in the same domain as the complete timestamp-concurrent task setting. Thus with our example in Table 5.4(d), adding the sliding window, we get the timeline as illustrated in Table 5.5 and the joined table becomes Table 5.5(3). We then can apply the pattern miner to generate the function sequence related to each task.

Instance Related Heuristic: We consider that the full concurrency occurs on process level, but not in the instance level. Therefore, multiple instances (from the same process design or not) may executed at any given time, but inside an instance, there is no concurrent task execution. Hence at any given time, in a particular instance, only one task is executed. Hence, we can use the same method as in the partial timestamp-unique task setting to determine the surrogate end time, where the end time of a task is equal to the start time of the subsequent task ($[ID_j]p_i^{end} = [ID_j]p_{i+1}^{start}$, where p_i and p_{i+1} are subsequent tasks in an instance with $ID = ID_j$). However, in the case of the last task in an instance, since it does not have any subsequent task, we still have to use the end of the log as its end time. Thus with our example in Table 5.4(d), with this consideration, we get the timeline as illustrated in Table 5.6 and the joined table becomes Table 5.6(3). We then can apply the pattern miner to generate the function sequence related to each task.

Table 5.5: Concurrent task setting



(1) Process log

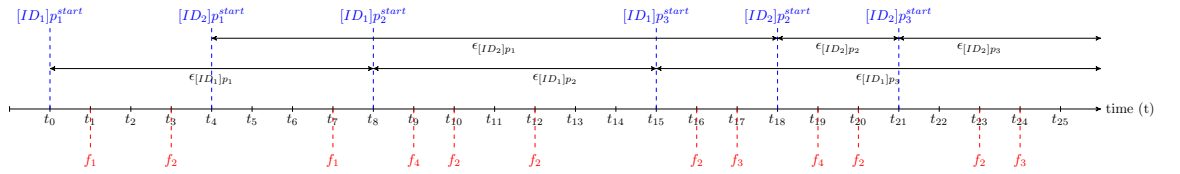
(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2, f_1 \rangle \rangle$
t_4	ID_2	p_1^{start}	t_2	f_2	$\langle p_1, \langle f_1, f_4, f_2 \rangle \rangle$
t_8	ID_1	p_2^{start}	t_7	f_1	$\langle p_2, \langle f_4, f_2, f_2 \rangle \rangle$
t_{15}	ID_1	p_3^{start}	t_9	f_4	$\langle p_3, \langle f_2, f_3, f_4, f_2 \rangle \rangle$
t_{18}	ID_2	p_2^{start}	t_{10}	f_2	$\langle p_2, \langle f_4, f_2, f_2 \rangle \rangle$
t_{21}	ID_2	p_3^{start}	t_{12}	f_2	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
			t_{16}	f_2	
			t_{17}	f_3	
			t_{19}	f_4	
			t_{20}	f_2	
			t_{23}	f_2	
			t_{24}	f_3	

(d) Partial timestamp-concurrent task setting

Table 5.6: Concurrent task setting



(1) Process log

(2) Function log

(3) Joined log

timestamp	ID	task	timestamp	function	sequence
t_0	ID_1	p_1^{start}	t_1	f_1	$\langle p_1, \langle f_1, f_2, f_1 \rangle \rangle$
t_4	ID_2	p_1^{start}	t_2	f_2	$\langle p_1, \langle f_1, f_4, f_2, f_2, f_2, f_3 \rangle \rangle$
t_8	ID_1	p_2^{start}	t_7	f_1	$\langle p_2, \langle f_4, f_2, f_2 \rangle \rangle$
t_{15}	ID_1	p_3^{start}	t_9	f_4	$\langle p_3, \langle f_2, f_3, f_4, f_2, f_2, f_3 \rangle \rangle$
t_{18}	ID_2	p_2^{start}	t_{10}	f_2	$\langle p_2, \langle f_4, f_2 \rangle \rangle$
t_{21}	ID_2	p_3^{start}	t_{12}	f_2	$\langle p_3, \langle f_2, f_3 \rangle \rangle$
			t_{16}	f_2	
			t_{17}	f_3	
			t_{19}	f_4	
			t_{20}	f_2	
			t_{23}	f_2	
			t_{24}	f_3	

(d) Partial timestamp-concurrent task setting

5.8 Evaluation

We perform the evaluation with an event log of a telephone repair process¹. This example describes a business process in a telephone repair company. When a customer sends their telephone to be repaired, it is registered as a new case. The telephone is then sent to the Problem Detection (PD) department. In this department, the telephone is analyzed and its defect is categorized. After the problem is identified, the telephone is sent to the Repair (R) department. Meanwhile, a letter is sent to the customer to inform them about the problem. The Repair department has two teams: one team to fix simple defects and another team to resolve more complex defects. Once the telephone is fixed, it is sent to the Quality Assurance (QA) department to be checked if the defect was indeed repaired or not. If the defect is not fixed yet, the telephone is sent back to the Repair department. If the telephone is repaired, the telephone is sent to the customer and the case is archived.

The event log consists of eight activities, which for simplicity we called them p_1 to p_8 . The log consists of 5460 events that represent 500 cases. Table 5.7 shows a small section of the event log.

We consider this event log as the process log for our evaluation. However, this log does not have the accompanied event log. Therefore, for our purpose we need to simulate the event log based on the settings we need to evaluate. We generate application service and their corresponding application components with notation App_n and f_i respectively. We then couple them with the business process that they serve. The complete list is shown in Table 5.8.

We use the start event and complete event timestamp as the start time and the end time of the activity, respectively. However upon closer inspection, we notice that not all activity has a start event, although all activities have a complete event. We

¹http://www.processmining.org/_media/tutorial/repairexample.zip

Table 5.7: A small section of the telephone repair event log

Case id	Time	Activity	Event type	Resource
0	2012-01-02T21:23:00+10:00	p_1	complete	System
0	2012-01-02T21:23:00+10:00	p_2	start	Tester3
0	2012-01-02T21:30:00+10:00	p_2	complete	Tester3
0	2012-01-02T21:31:00+10:00	p_3	start	SolverC1
0	2012-01-02T21:49:00+10:00	p_3	complete	SolverC1
0	2012-01-02T21:49:00+10:00	p_5	start	Tester3
0	2012-01-02T21:55:00+10:00	p_5	complete	Tester3
0	2012-01-02T22:10:00+10:00	p_7	complete	System
0	2012-01-02T22:10:00+10:00	p_8	complete	System
1	2012-01-01T20:09:00+10:00	p_1	complete	System
1	2012-01-01T20:09:00+10:00	p_2	start	Tester2
1	2012-01-01T20:15:00+10:00	p_2	complete	Tester2
1	2012-01-01T20:35:00+10:00	p_4	start	SolverS1
1	2012-01-01T20:42:00+10:00	p_4	complete	SolverS1
1	2012-01-01T20:42:00+10:00	p_5	start	Tester6
1	2012-01-01T20:48:00+10:00	p_5	complete	Tester6
1	2012-01-01T20:54:00+10:00	p_6	complete	System
1	2012-01-01T20:54:00+10:00	p_4	start	SolverS2
1	2012-01-01T20:55:00+10:00	p_7	complete	System
1	2012-01-01T21:03:00+10:00	p_4	complete	SolverS2
1	2012-01-01T21:03:00+10:00	p_5	start	Tester4
1	2012-01-01T21:09:00+10:00	p_5	complete	Tester4
1	2012-01-01T21:14:00+10:00	p_8	complete	System
...

Table 5.8: The business processes, application functions, and application components and their correlations

Business process	Application function	Application component	Application function
p_1	$\langle f_1, f_2 \rangle$	App_1	f_1, f_2
p_2	$\langle f_3, f_4 \rangle$	App_2	f_3, f_4, f_5
p_3	$\langle f_5, f_6, f_7 \rangle$	App_3	f_6, f_7, f_8
p_4	$\langle f_5, f_6 \rangle$	App_4	$f_9, f_{10}, f_{11}, f_{12}$
p_5	$\langle f_8, f_9 \rangle$		
p_6	$\langle f_{10} \rangle$		
p_7	$\langle f_{11}, f_{12} \rangle$		
p_8	$\langle f_{12} \rangle$		

particularly need the start time because all scenarios or settings require that all tasks have start time. There are three activities in particular that does not have a start time, i.e., p_1 , p_6 , p_7 , and p_8 . Therefore we have to generate the start time for each of these activities so that we can determine the task execution time, ϵ . We use the following provision to calculate *epsilon* for tasks without start time:

$$\epsilon_{p_i} = \begin{cases} \text{if } i = 0 & \sum_{i=1}^n \epsilon_{p_i} / n \text{ and } p_i^{start} = p_i^{end} - \epsilon_{p_i} \\ \text{if } i > 0 & p_i^{end} - p_i^{start}, \text{ where } p_i^{start} = p_{i-1}^{end} \end{cases}$$

After we add in the start time for all activities, the log now contains 7152 entries. Next we use this log in four different settings as defined in Section 5.6. For sequence pattern mining, BIDE+ [262], a frequent sequential pattern mining algorithm from SPMF pattern mining library [259] is used.

The process log now represents the complete timestamp scenario. The corresponding event log was generated by assigning a random timestamp to related functions in the duration of task execution, i.e., after task start time and before the end time. This simulates functions that must be started for a task completion. We then apply the method in Section 5.6 to generated the joined log on the complete timestamp - unique task setting (we refer to this log as *Setting₁* log) and then use the sequence pattern miner to mine the task-function correlation.

The log was then modified to reflect the second setting, partial timestamp - unique task setting. In this setting, we remove the end time of all tasks. Following the method explained in Section 5.6, we generate the joined log (we refer to this log as *Setting₂* log) and use it as the input to the pattern miner.

In the next setting, complete timestamp - concurrent task setting, to reflect the concurrent task setting, we sorted the log by the timestamp in ascending order. Again, we simulated the function call where its timestamp is between the start of the task

and task completion. Using the method described in the Section 5.6, the joined log (refer as *Setting₃* log) is generated and use as input to the sequence pattern miner.

For the last setting, partial timestamp - concurrent task setting, we generated the joined log (referred as *Setting₄* log) using the process log from the previous setting and removed the end time of each task. Similar to the previous setting, we also simulated the corresponding event log with random timestamp later than the start time of the task. We use the assumption where the end of sequences is the end of the log (the log was relatively small thus the sequences were not considerably long.)

Recall and precision value of every discovered correlation were used to measured the performance of the technique. Perfect recall (where recall value is 100%) for a correlation means that all functions that were invoked or started to accomplish task p_i were discovered in the generated sequence $\langle p_i, \langle f_l, \dots, f_k \rangle \rangle$. For instance, the recall for discovered sequence $\langle \text{Archive User}, \langle \langle f_{11}, f_{12} \rangle \rangle \rangle$ is 100% for correlation $\langle \text{Archive User}, \langle \langle f_{11} \rangle \rangle \rangle$. Perfect precision (precision=100%) for task-function correlation means that only the functions that were invoked or started to accomplish task p_i were discovered in the generated sequence $\langle p_i, \langle f_l, \dots, f_k \rangle \rangle$. For example, the precision for discovered sequence $\langle \text{Archive User}, \langle \langle f_{11}, f_{12} \rangle \rangle \rangle$ is 50% for correlation $\langle \text{Archive User}, \langle \langle f_{11} \rangle \rangle \rangle$.

In our settings, *Setting₁* log measured in 100% both for recall and precision. *Setting₁* log represents the setting where the complete timestamp was recorded in the log thus all correlations are expected to be discovered correctly. Furthermore, *Setting₁* log guaranteed that at any given time, only a single task is being executed, thus all the functions correlated to any particular task would be started before the task completion time. Thus, all function sequence always occurred during task execution period and no other task may interfere with the execution of any particular task. For these reason, we considered *Setting₁* log as our baseline in this evaluation. We get the same results for *Setting₂* and *Setting₃* logs where both settings also produce 100%

value for both recall and precision.

Table 5.9 shows the result for *Setting₄* log. The average recall for this setting is less than 100%. We conclude that this is due to the limitations of the pattern mining library. In our case study, tasks p_3 and p_4 shared functions $\{f_5, f_6\}$. This caused some repetitions in the sequence which seems to not be able to be picked up properly by the pattern mining library.

Table 5.9: *Log₄* Result

Task	Function Sequence	Discovered Sequence	Recall	Precision
p_1	$\langle f_1, f_2 \rangle$	$\langle f_1, f_2 \rangle$	1	1
p_2	$\langle f_3, f_4 \rangle$	$\langle f_3, f_4 \rangle$	1	1
p_3	$\langle f_5, f_6, f_7 \rangle$	$\langle f_5, f_6 \rangle$	0.67	1
		$\langle f_5, f_7 \rangle$	0.67	1
p_4	$\langle f_5, f_6 \rangle$	$\langle f_5, f_6 \rangle$	1	1
p_5	$\langle f_8, f_9 \rangle$	$\langle f_8, f_9 \rangle$	1	1
p_6	$\langle f_{10} \rangle$	$\langle f_{10} \rangle$	1	1
p_7	$\langle f_{11}, f_{12} \rangle$	$\langle f_{11}, f_{12} \rangle$	1	1
p_8	$\langle f_{12} \rangle$	$\langle f_{12} \rangle$	1	1
Average			0.98	1

We utilized the assistance of the ProM framework [241], a process mining tool, to construct the business layer. As we mentioned earlier, we considered *Setting₁* log as our baseline, thus the generated model based on *Setting₁* log is also used as our baseline. *Setting₂*, *Setting₃* and *Setting₄* logs also resulted in identical models as *Setting₁* log. Figure 5.2 shows the resulting EA model extracted from *Setting₁*, *Setting₂* and *Setting₃* logs (which returned perfect recall and precision) that represents business and application layers and their relationships.

5.9 Related works

A number of Enterprise Architecture (EA) tools has been marketed for EA modeling. Most of them are equipped with data collection mechanism to build EA. TOGAF

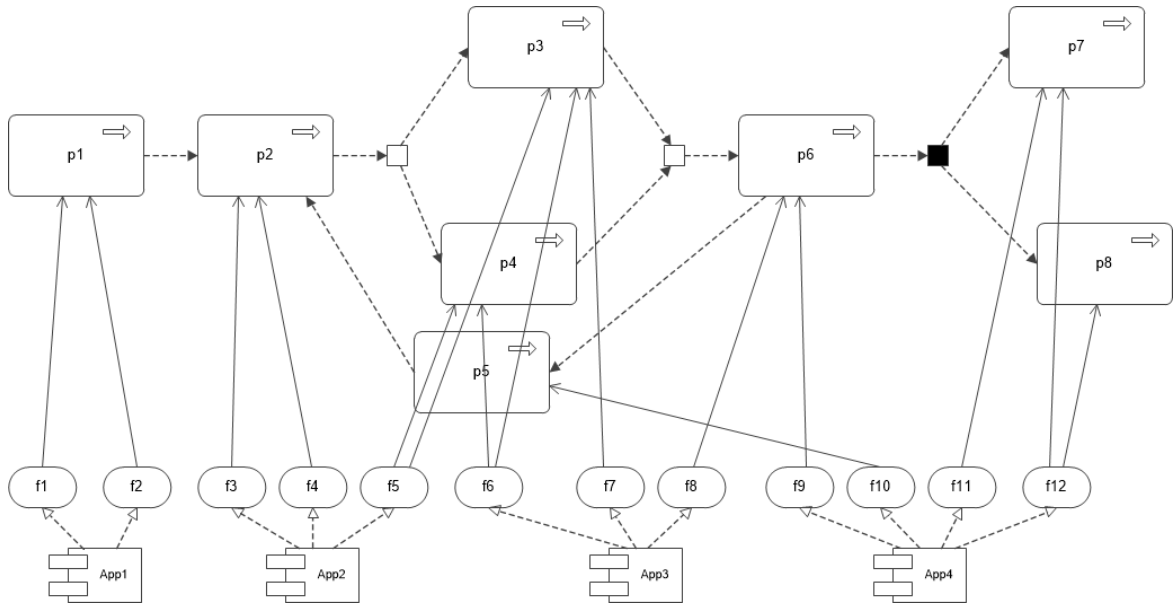


Figure 5.2: ArchiMate Business and Application Layer

ADM Tools [260] provide visual assistance through every development phases of enterprise architecture. Features of the tool includes actionable activities for developing all deliverables, auto deliverable composer, and auto versioning. BiZZdesign Enterprise Studio [33] is a modelling, visualisation, analysis and documentation of enterprise architecture tool that supports ArchiMate 3.0. It also supports automatic data collection from office applications to build the architecture model. Another another modeling tool for enterprise architecture is Enterprise Architect [231]. It offers supports in developing enterprise architecture through simulation and traceability. Corso Agile Enterprise Architecture [47] provides analysts with a collaborative platform to build an enterprise architecture. It also provides tools to build multiple diagrams and roadmaps necessary to develop an enterprise architecture. Planview Enterprise [203] is another modeling tool for enterprise architecture for visualisation across multiple views. It also offers a library of component and relationship types to model enterprise data and a data collection feature to import information from databases using SQL queries. ARIS Architect and Designer [228] is a visualisation tool that also supports model analysis and provides data analysis and process monitoring using KPIs . How-

ever, these modeling tools generate partial EA model where the resulting models of these modeling tools do not completely include all elements of EA, for instance infrastructure services, and a significant effort of its abstraction is dependent on the EA modeller.

As we mention previously, our work closely related to [219], where both works focusing on discovering the correlations between multiple layers in EA. We extend this study by proposing different mechanisms in generating joined log in different settings and heuristics in more specific setting. By applying these heuristics, we can avoid the problem of long sequences due to the length of the log.

In this section, we focus on two layers of EA, business layer and application layer. However, we do not discuss the mining of the elements in those layer itself, instead we concentrate on the correlation between the two. The mining of the elements in both layers themselves has been presented in several studies. Process mining represents a vast amount of research to mine business processes from event log. With a variety of mining methods, such as α -algorithm [249], heuristic miner [267], and fuzzy miner [105], one can employ these algorithms to mine business processes (and business actors) in the business layer. For the application layer, Holm et al. [125] and Buschle et al. [40] propose a technique to map automatically collected data (acquired using network scanners) to an ArchiMate model. In this technique, the collected data only represented elements in the application layer. Thus, the information provided is not enough to infer about elements in multiple layers or the inter-relationships between different layers.

5.10 Summary

In this chapter, preliminary evidence was provided to support the hypotheses that the relationship between business and application layer of ArchiMate can be inferred

from process and event logs within the enterprise. The proposed technique combines the time correlation heuristic and frequent closed sequential pattern mining. The evaluation result, as a proof-of-concept, shows that this technique is able to extract the correlations between multiple layers in an ArchiMate model.

Acknowledgment: This section was a collaboration work between the author (Metta Santiputri) with the author's supervisors (Prof. Aditya Ghose and Dr. Hoa Khanh Dam). The author's supervisors provided the guidance specifically in the direction of the work and the final reviews before the publication. This work also developed with a collaboration with other colleagues (Ayu Saraswati and Chee Fon Chang) who were provided the initial ideas. However the elaboration of this initial ideas and the development of this work has been performed by the author.

Chapter 6

Goal orchestrations: Modelling and mining flexible business processes

Goals in an enterprise and its achievement are considered as knowledge driver. An insight into an enterprise can be gain from readily available data is the goals that needs to be achieved. Therefore, the process in an enterprise are described as the coordination of goals instead of the coordination of tasks. The objective in this chapter is to formalize this alternative approach to process modeling via goal orchestrations. It provides a more natural means of modeling behaviour (or processes) and to ease the human understanding.

This chapter presented the approach in method in goal-oriented business process modeling. The chapter starts with an introduction in Section 6.1. The goal orchestration modeling and semantics is presented in Section 6.2. Section 6.3 gives more details about goal orchestration execution. A method to mine goal orchestration is provided in Section 6.4. An evaluation using both synthetic process models and a real-life dataset is presented in Section 6.5. This chapter is summarized in Section 6.6.

6.1 Introduction

Typically, in an application domain, a business process is described as a set of tasks to be performed. However, in many domains, it is more natural to think of a process as a coordination model of goals to be achieved. Take for example in the clinical domain. For a physician, they act based on the outcome they want to achieve, such as to lower a patient's blood pressure or to raise a patient's blood sugar, rather than just following a sequence of given actions, such as give medicine A or perform procedure B. This example illustrates a goal-driven system, where the goals dictate what the actions to be taken, and a knowledge-intensive system, where the background knowledge of an actor in the system influence the choice of actions that they perform. Therefore, in these domain settings, it is preferable to define goals and alternative ways to realize a goal are given.

In any system, goals encompass the various objectives that the system should achieve. A goal model constructs these goals into a hierarchy that describes the relations between goals and its subgoals. With goal decompositions or refinements, a goal model also defines the alternative realizations of each goal.

These goals are then translated into a series of tasks or activities to be performed. Given the post-conditions after a task or an activity is executed, we can determine whether a goal has been achieved. However, the outcome of an activity is context sensitive, which means that the result of the current activity is also depends on the outcome or result from the previous activity. Therefore, this notion of non-deterministic caused by context sensitivity of the task post-conditions also influence the realization of a goal.

Combining both the flexibility of goal realizations described in the goal model and the context sensitivity of a task outcome, our idea is to represent the business process model as a coordination of goals. This representation allows us to enact processes both in flexible and context sensitive ways. Moreover, by representing a business process in

this fashion and thus giving the actors enough independent to realize a goal in multiple different ways, we enable a flexible process management.

6.2 Goal orchestration models and semantics

A goal can be represented in any truth-functional language that comes equipped with machinery for checking satisfiability (and hence entailment). In the remainder of the chapter, we will implicitly refer to *achievement goals* when we refer to a *goal*. A *goal orchestration* is best viewed as a process graph, as commonly used in the literature, with the tasks/activities replaced with goals. Formally, a **goal orchestrations** is a pair $\mathbf{GC} = (N, F)$ where $N = G \cup \Gamma \cup E$ (G is a set of goal assertions, Γ is a set of gateways, and $E = E_s \cup E_f$ is a set of special events, with E_s representing start events and E_f denoting end events); $F \subseteq (N \setminus E \times N \setminus E) \cup (E_s \times N \setminus E) \cup (N \setminus E \times E_f)$ corresponds to sequence flows connecting goal assertions with goal assertions, goal assertions with gateways, gateways with goal assertions, start events with goal assertions and goal assertions with end events.

We will now describe the semantics by specifying under what circumstances an *event log* will be deemed to *satisfy* a goal orchestration. Recall that an event log is a set of pairs of the form $\langle event, timestamp \rangle$. We order an event log from the earliest timestamp to the latest, obtaining a sequence of the form $\langle e_1, e_2, \dots, e_n \rangle$, where each element of the sequence is of the form $e_i = \langle \epsilon_i, \tau_i \rangle$ (ϵ_i is the i -th event and τ_i is the corresponding timestamp) and for every adjacent pair of elements in the sequence $\langle e_i, e_{i+1} \rangle$, $\tau_i \leq \tau_{i+1}$.

At a fundamental level, every event involves one or more state transitions (a task object transitions from an *incomplete* state to a *completed* state, or a business object such as an insurance claim transitions from a *not-determined* state to an *accepted* state etc.). The effects of some events *persist* (an insurance claim once accepted remains in

the accepted state) while others do not (a light that is initially switched on is eventually switched off). An event log describes the changes but not the non-changes. In other words, such a log describes new events as they occur but does not describe which prior events have persistent effects. In the following, we will not distinguish between an event and its effects - thus the description of an event is also the description of its effects. To obtain a sequence of states (or partial states) from an event log, we *accumulate* effects using a *state update operator* in a manner similar to the approach adopted in [266, 123]. Recall that a state update operator takes a state description and the effects of an action to generate one or more descriptions of the state that would accrue from executing this action in the input state. Some well-known state update operators are the Possible Worlds Approach (PWA) [90] and the Possible Models Approach (PMA) [271]. Given a set of accumulated effects (representing a possibly partial description of the state of the operating environment), and a new effect, we use the state update operator to determine what new set of accumulated effects should be (in our evaluation, we use the PWA operator, but others could be used without loss of generality). The application of the state update operator (we shall refer to it with the symbol \oplus) leads to non-deterministic outcomes. Thus, if s_1 and s_2 are even effect assertions, then $e_1 \oplus e_2$ is a set of event/effect assertions (the intuition being that any one of these could be the result of making e_2 true in state e_1).

The idea, now, is to generate from an event log a sequence of *sets of states* or partial states (the non-deterministic nature of the state update operator making it necessary to consider sequences of sets of states as opposed to sequences of states). Given a set of prior states and a set of posterior states (i.e., those obtained from the prior set via state update), it is important to note that a state in the posterior set can be arrived at only from some (but possibly not all) of the states in the prior set. In other words, there is a predecessor-successor relationship between temporally adjacent sets of states

that is important to record. We will therefore first extract from an event log a *state set sequence* consisting of pairs of states, where the first element is the predecessor and the second element is the successor. Given an event log $\langle e_1, e_2 \dots, e_n \rangle$, we compute a *state set sequence* $\langle StateSet_1, StateSet_2, \dots, StateSet_n \rangle$, where each $StateSet_i$ is of the form $\{StatePair_1, StatePair_2, \dots, StatePair_k\}$ and each $StatePair_i$ is of the form $\langle state_{pred}, state_{succ} \rangle$ (i.e., these are predecessor-successor pairs) as follows:

- We set $StateSet_1 = \{\langle \emptyset, \epsilon_1 \rangle\}$ (where $\langle \epsilon_1, \tau_1 \rangle$ is the first entry in the temporally ordered event log).
- We set $StateSet_2 = \{\langle \epsilon_1, s \rangle \mid s \in \epsilon_1 \oplus e_2\}$
- For $i = 3 \dots n$, $StateSet_i = \{\langle s_{i-1}, s_i \rangle \mid s_{i-1} \in StateSet_{i-1} \text{ and } s_i \in s_{i-1} \oplus e_i\}$

A *state sequence* $\langle s_1, s_2, \dots, s_n \rangle$ is supported by a state set sequence $\langle StateSet_1, StateSet_2, \dots, StateSet_n \rangle$, if and only if:

- $StateSet_1 = \{\langle \emptyset, s_1 \rangle\}$
- Every adjacent pair $\langle s_{i-1}, s_i \rangle$ in the state sequence must be an element of $StateSet_i$ in the corresponding state set sequence.

Given a state sequence $\langle s_1, s_2, \dots, s_n \rangle$ and a goal model with a goal set $\{g_1, g_2, \dots, g_k\}$, we compute a *goal sequence* $\langle G_1, G_2, \dots, G_n \rangle$ by setting each $G_i = \{g_i \mid s_i \models g_i\}$. Note that a goal sequence is a sequence of sets of goals. We define a *goal orchestration trace* as a sequence of goals $\langle g_1, \dots, g_m \rangle$ satisfying the constraints of the corresponding goal orchestration model (much like a trace through a process model). Given a goal orchestration model and a trace $\langle g_1, \dots, g_m \rangle$, we will say that the trace is *supported* by a goal sequence $\langle G_1, G_2, \dots, G_n \rangle$ if it is the case that $n \geq m$ and every $g_i \in G_i$.

Given a goal model (and thence, the set of goals contained in it), an event log *satisfies* a goal orchestration model if and only if a goal sequence can be obtained from

the event log and the goal model in the manner described above such that the goal sequence supports a trace for the goal orchestration model.

6.3 Executing goal orchestrations

For a goal orchestration approach to enable flexible process execution, we require tasks/activities or enterprise capabilities to be annotated with post-conditions (specified in the same ontology as the goals). More generally, one can view this as an instance of a generic scheme that permits us to relate task execution to the functional outcomes that are used to specify goals. A number of recent proposals suggest that leveraging task post-condition annotations can be effective and practical [70, 71, 117, 123, 64, 227, 266, 64, 86]. Still more recent results [218] suggest that task post-conditions can be relatively reliably mined from readily available enterprise data.

The first question we need to address is whether a goal orchestration is *feasible* with respect to an *enterprise capability library*. We shall view the latter as a repertoire of tasks or capabilities annotated with post-conditions. A goal orchestration is *strongly feasible* if and only if for every trace admitted by the orchestration, there exists a task/capability sequence $\langle t_1, \dots, t_n \rangle$ with a corresponding sequence of post-conditions $\langle p_1, \dots, p_n \rangle$ such that this latter, if viewed as an event log (this can be easily done by inserting time-stamps with each each post-condition that respects the relative ordering), generates (given a goal model) a goal sequence that support that trace. In the case of *weak feasibility*, we only require that there exist a task sequence that generates a goal sequence that supports at least one trace. The subsequent analyses will only be performed for goal orchestrations that are (strongly or weakly) feasible with respect to the available enterprise capability library.

Practical deployment of goal orchestrations must ideally be done with a goal model at hand. A goal model, typically an AND-OR goal tree, is critical in offering alternative

means of arriving at the same outcome. We will refer to any goal related to a parent goal g in the goal model via an OR-link as an OR-refined child goal, and the OR-refined children of these and so on as the OR-refined descendants of g . We shall refer to the set of all OR-refined descendants of a goal g as the *OR-alternatives of g* . Given a goal orchestration model GOM , the set $OR-Alt(GOM)$ of OR-alternatives of GOM consists of all goal orchestration models obtained by replacing at least one goal in GOM with an OR-alternative.

Executing a goal orchestration model consists of computing an *optimal suffix* for a partially executed task sequence (empty at the start of execution). By introducing a *current state* into the problem, one can deal with the problem of *semantic compensation* [97], where a process deviates from the functionality it is expected to deliver (manifested via events/effects) and where the challenge is to compute a new sequence of activities that will restore the process to *semantic conformance* (where it delivers the expected effects) and achieve the final goals. Formally, given: (1) The current state S of the process and its environment, (2) a goal orchestration model and (3) the current sequence of goals achieved $\langle g_1, \dots, g_i \rangle$, compute: a sequence of tasks $\langle t_j, \dots, t_m \rangle$ drawn from the enterprise capability library such that the corresponding sequence of task post-conditions $\langle p_j, \dots, p_m \rangle$, when concatenated with the achieved goal sequence $\langle g_1, \dots, g_i \rangle$ generates a sequence of events $\langle g_1, \dots, g_i, p_j, \dots, p_m \rangle$ which can be viewed as an event log (with the appropriate insertion of sequence-maintaining time-stamps, as before) generates a goal sequence that supports a goal trace through the input goal orchestration model.

Figure 6.2 shows the goal orchestrations of the treatment for a child with a head injury as described in Figure 6.1. This model contains the goal assertions that represents the goals to be achieved by performing any task in the procedure. For example, the goal of administering IV bolus of dextrose is to maintain the blood glucose level

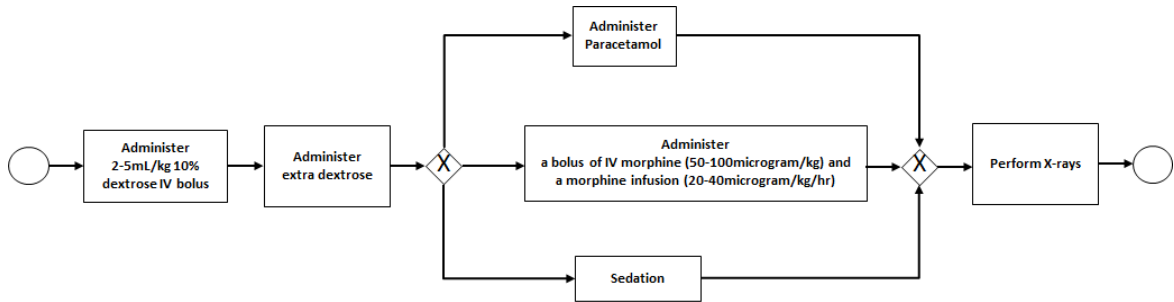


Figure 6.1: Treatment for children sustaining head injury with low blood sugar level on normal range. By administering extra dextrose, the body fluid balance is achieved. The administration of analgesia is aim to reduce pain and stress. To observe the chest, pelvis, and spine for precaution on trauma cases is perform using x-rays. Therefore the goal assertions are *Maintain patient's blood glucose level*, *Maintain patient's body fluids*, *Reduce patient's pain and stress* and *Observe patient's chest, pelvis and spine* as illustrated in the goal orchestration.

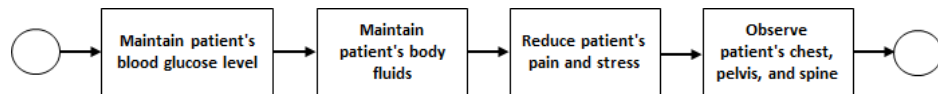


Figure 6.2: Goal orchestrations for business process model in Figure 6.1

Observe that ‘Administer Paracetamol’, ‘Administer a bolus of IV morphine (50-100 microgram/kg) and a morphine infusion (20-40 microgram/kg/hr)’, and ‘Sedation’ tasks are represented by one goal, i.e. *Reduce patient's pain and stress*, seeing as these tasks has identical purpose and performing any of these tasks aims for that purpose.

In this example, we progress through each possible trace and compare the post conditions of each task with every goals. By identifying the links between tasks in the process model to goals by way of the post conditions, the goal orchestration obtained in previous example represent the normative satisfaction links.

6.3.1 Goal Consistency

Another monitoring aspect in the goal satisfaction is in the sequence of which the goals are satisfied. An execution trace is a sequence of tasks or activities, where each task or activity execution is intended to achieve goal(s), therefore each goal in the sequence must be achieved before its successor goal. Based on the refinement of goals in the goal model and the sequence on which the goal is realized, there is a temporal constraint between goals.

By definition, any sub-goals has precedence before its parent goal, which means that in any execution the sub-goals must be satisfied before the parent goal is satisfied. This is true both in AND and OR-refinements. In AND-refinement, to satisfied the parent goal, all the sub-goals must be satisfied beforehand. While in OR-refinement, one of the sub-goal must be satisfied to satisfy the parent goal, thus in any trace, one of the sub-goals must be satisfied before the parent goal.

One might argue that this is not always the case, specifically in the occasion where the parent goal is an abstract goal in which case there is no task or activity to be executed that bring about the resulted state. However, since in our representation we focus on observations of the state of the objects, we can maintain that it is always the case that when all sub goals are achieved (as indicated by the observation of the state of some objects), then we can observe that these states also indicated that the parent goal has been achieved, even without any execution of any task or activity.

In any execution, the trace must conform to the goal model (any path in the process is executed to realize or satisfy any one of the goal in the goal model) and the precedence of goals (the trace must reflect the hierarchy in the goal model). Otherwise the trace is considered as an exception.

6.4 Mining Goal Orchestrations

Obtaining the data:

- *Pre-defined instance-object associations:* In settings where we know a priori the set of objects associated with and impacted by a given instance, we can partition an effect log based on the identifier of the instance (i.e., the case ID), predicated on the assumption that every effect is associated with an object or a set of objects.
- In some settings, the complete set of objects impacted by a process might not be known a priori (we might in fact be interested in *discovering* what these objects might be)

In this section, we show how goal orchestrations can be mined from event logs. A formal statement of the problem is as follows. Given: (1) An event log and (2) a goal model, compute: a goal orchestration that best explains the behaviour encoded in the event log. Recall that an event log records two kinds of events: events that flag the execution of a task and events that describe state transitions in objects impacted by a process. Our interest is in the latter kind of event (we shall refer to these as *effects*). It is useful to note that we do not need case IDs associated with effects. Given a set of effects, we are only interested in their temporal ordering, but not which process instance, or actor/agent, might have generated. Our intent is to identify the sequence of goals achieved (and thence a goal orchestration model) from the sequence of effects manifested. The vocabulary of available goals (as provided in the input goal model) provides the lens through which we view the effects. If the goal model is specific to an actor or a process instance, then the goals we will recognize and mine will be specific to the process or actor in question.

Mining goal orchestrations from event logs involves a sequence of pre-processing

steps, followed by the application of an off-the-shelf process mining tool (in the empirical evaluation presented in the next section, we use AlphaMiner from the ProM toolkit [241]). The steps involved are as follows:

- Processing event logs (specifically *effect logs*) to obtain a *non-deterministic cumulative effect sequence*.
- Extracting a set of *cumulative effect sequences* from the non-deterministic cumulative effect sequence.
- Extracting *goal sequences* from cumulative effect sequences.
- Extracting a set of ordering assertions from each goal sequence identified in the previous step.
- Running an off-the-shelf process mining tool with the goals playing the role of tasks.

Let the event log be the form $\langle e_1, e_2, \dots, e_n \rangle$. We assume that there is also a background knowledge base \mathcal{KB} defined in the same language as that in which the effects are described. The non-deterministic cumulative effect sequence can be obtained by:

- We set the first element of the non-deterministic cumulative effect sequence to be $\{e_1\}$
- We obtain each subsequent element in the non-deterministic cumulative effect sequence (of the form $\{E_i\}$) from the prior element using the following rule:

$$E_{i+1} = E_i \oplus e_{i+1}.$$

The following example illustrates how this is done.

The non-deterministic notion of the cumulative effect sequence is the result of the application of the state update operator which can be non-deterministic in general.

Table 6.1: An example of event log with corresponding non-deterministic cumulative effect sequence

$\mathcal{KB} : t \rightarrow \neg(p \wedge r)$	
e_i	non-deterministic cumulative effect sequence
$\{p, q\}$	$\{\{p, q\}\}$
$\{r, s\}$	$\{\{p, q, r, s\}\}$
$\{t\}$	$\{\{p, q, s, t\}, \{r, q, s, t\}\}$

Thus an element of the non-deterministic cumulative effect sequence is a set of sets of effects.

We extract a set of cumulative effect sequences that is *supported* by the non-deterministic cumulative effect sequence by following the condition that every adjacent pair $\langle \mathbb{E}_{i-1}, \mathbb{E}_i \rangle$ in the cumulative effect sequence must be an element of the corresponding non-deterministic cumulative effect sequence. Thus from our example in Table 6.1, the cumulative effect sequences are $\{\{p, q\}, \{p, q, r, s\}, \{p, q, s, t\}\}$ and $\{\{p, q\}, \{p, q, r, s\}, \{r, q, s, t\}\}$.

Given a goal model with a goal set $\{g_1, g_2, \dots, g_k\}$ and a cumulative effect sequence $\{\mathbb{E}_1, \mathbb{E}_2, \dots, \mathbb{E}_m\}$, we compute a goal sequence $\langle G_1, G_2, \dots, G_n \rangle$ where $G_i = \{g_i \mid \mathbb{E}_i \models g_i\}$.

The extraction of ordering assertions from a goal sequence proceeds as follows:

- $o_1 > o_2$ if and only if there is a traceability links tr such that $tr = \langle G_1, G_2, \dots, G_m \rangle$ and $G_i = o_1$ and $G_{i+1} = o_2$
- $o_1 \rightarrow o_2$ if and only if $o_1 > o_2$ and $o_1 \not\neq o_2$
- $o_1 \# o_2$ if and only if $o_1 \not\neq o_2$ and $o_2 \not\neq o_1$
- $o_1 || o_2$ if and only if $o_1 > o_2$ and $o_2 > o_1$

Using these relations, we use the adaptation of alpha algorithm [249] to discover the goal orchestrations. The main difference in our approach to discover goal orches-

trations is that the input of the algorithm is not a set of event trace, but instead the input is a set of traceability links.

6.5 Evaluation

The purpose of the evaluation is to establish that our approach is capable of the following:

- mining the goal orchestration from the readily available data
- identifying different alternatives to achieve a goal based on the execution history

We present two cases to perform our evaluation. The first case involve a synthetic dataset and the second evaluation using a real-life dataset from a ticket handling process.

6.5.1 Evaluation with synthetic process models

Our aim is to establish that our approach discovers the goal orchestration from the data. We ran the first experiment with a synthetic semantically annotated process model using T_1, T_2, \dots etc, for task names and p, q, \dots for effects. The model consists of 12 tasks with an XOR-split leading to two alternative flows, one of which included a nested AND-split and the other a nested XOR-split. The semantic annotations were 2 or 3 literals long and involved a mix of conjunctions and disjunctions. For this exercise, we omitted the knowledge base, i.e. we did not have any rule in the knowledge base. We generated a large number of possible execution traces of this model, and obtained the synthetic log using BIMP (The Business Process Simulator)¹ (with a small process model, performing the execution by hand also produced similar logs). We also investigated the effect of scaling up the complexity of the process model, by

¹<http://bimp.cs.ut.ee/>

generating a second synthetic process model with 20 tasks with and XOR-split leading to four alternative flows, one flow included a nested AND-split, two included XOR-split (one leading to two alternative flows and the other leading to three alternative flows), and the other was a sequence.

We randomly assign effects to tasks, then perform the pre-processing steps to obtain goal sequences, and from there, the goal orchestration. In this exercise, we have access to the ground truth (by maintaining the original process models together with its assignments of effects associates with each task and the goal sequence of each trace in the process model) thus we can determine the fitness and precision values for the mined goal orchestration.

Table 6.2 below describes the results of the experiments with each of these two process models. We measure the fitness and precision of the goal orchestrations generated from the log. Fitness evaluates whether the observed process complies with the control flow specified by the process, while precision indicates how precisely the model describes the observed process. In both process model, the results shows that the goal orchestrations generated from the mining conform to the data. The results also shows that there is an insignificant number of possible incorrect in the link between task and goal which cause an incorrect goal sequence in the traceability link.

The synthetic effect logs used in these examples considered all possible flows. Real-life data might involve more imperfections (such as certain XOR flows never being executed, certain tasks never being executed and so on).

6.5.2 Evaluation with real-life dataset

An important part of the evaluation of the feasibility of the overall approach to goal orchestration was to gain experience in using it in with a real-life dataset in a large complex practical setting. Our intent was to test several key elements of our pro-

Table 6.2: Evaluation result with synthetic data

Process model 1

# of instances	# of events	Fitness	Precision	Time (ms)
100	1520	1.0	1.0	52
500	7540	1.0	1.0	160
1000	15094	1.0	1.0	257
5000	75640	1.0	1.0	548
10000	151080	0.99	1.0	1,149

Process model 2

# of instances	# of events	Fitness	Precision	Time (ms)
100	1810	1.0	1.0	95
500	9008	1.0	1.0	287
1000	18026	1.0	1.0	377
5000	90040	0.99	1.0	1,170
10000	180540	0.99	1.0	3,147

posal, including the processing (and pre-processing) of event logs, the identification of goals and goal sequences and eventual use of process mining to obtain explicit goal orchestration models. Specifically, we looked at data from a team in one of the world’s largest IT companies that supports IT infrastructure management as an outsourced service. Much of its activities involves the handling and resolution of *problem tickets* generated by customers. These can span the spectrum of complexity from a simple password reset to dealing with a complete ATM network that might have gone down. The dataset we analyzed described how 65000 distinct problem tickets were handled.

The ticket handling process is illustrated in Figure 6.3. In this process, when a member of a client firm faces IT-related problems or has queries about the IT systems whose management has been outsourced, they raise a ticket. The ticket handling system maintains records of ticket status from the opening of a ticket until the closing of it, responds with an acknowledgment to the user along with a notification to a system engineer who is assigned to handle the ticket. Also further input from the user may be requested. At this stage, if the problem can be resolved, the ticket is closed.

In case the problem can not be resolved, the system checks to see whether there is any update from the user. If no update is provided and the ticket is not re-opened within a stipulated time, then the problem is considered as resolved and it is automatically closed. If the ticket is updated with new information then the system checks the nature of the ticket, whether it is incident or request, depending on which the ticket is serviced or resolved respectively.

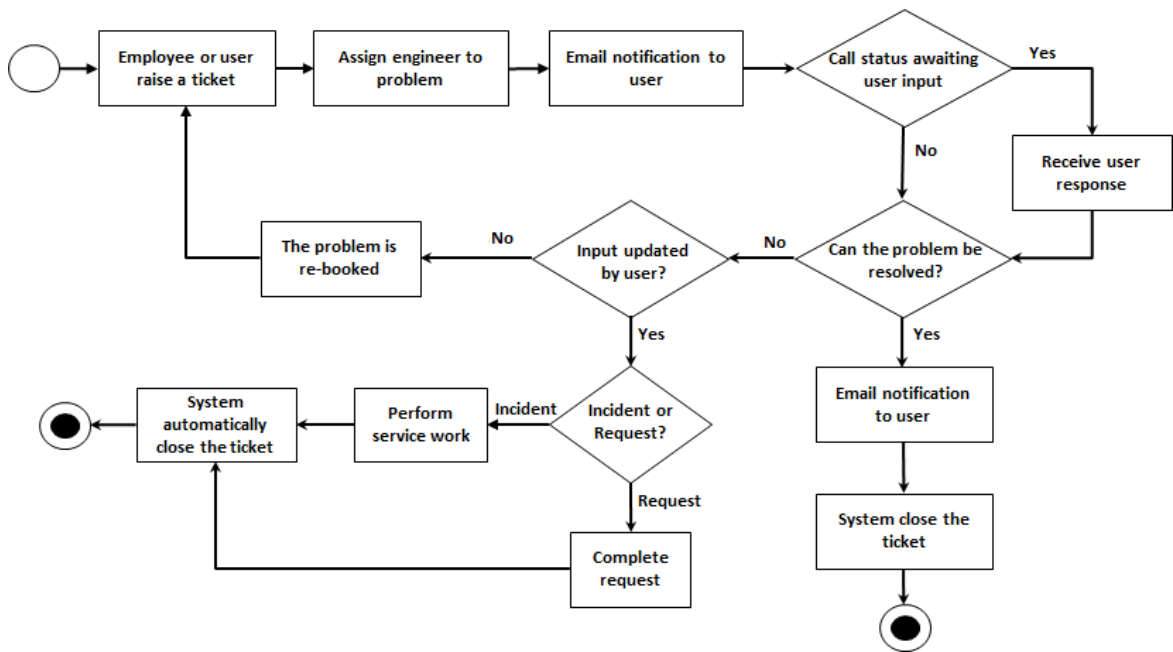


Figure 6.3: Ticket handling process

The ticket handling system recorded all events related to a ticket in the process. Each record represents all attributes of a ticket, such as incident number or ticket number to identify any particular ticket, the identity of the user or employee that raised the ticket, the timestamp of when the ticket is raised (`open date` attribute), when the problem is resolved (`resolve date` attribute), when the system sends a response to the user and the engineer (`respond date` attribute), when the ticket is closed (`close date` attribute), an attribute to signify if the ticket is reopened, etc. These attributes will be used to identify the current state of the ticket. For example, a ticket in the `Open` state signifies that the ticket has been received and currently at

the start of the ticket handling process. Similarly, a ticket in `Close` or `Auto-close` state signifies that it is at the end of the process, etc.

The 65000 tickets in our dataset were created or submitted during December 2013. Three ticket instances in Table 6.3 illustrate how time-stamped milestones (such as “Open Date” or “Receive Date”) together with ticket attributes can be used to generate an event log for each ticket. From the 65000 tickets, we identify 16 distinct effect sequences, shown in Table 6.4.

Table 6.3: Ticket examples

P		Q		R		S		T		W		X		AE	
Incident No	Open Date	Resolve Date	Close Date	Respond Date	Open	Closed	Reopened_or_not								
RQ14279521	12/4/13 15:39	12/6/13 13:18	12/9/13 13:21	12/4/13 16:39	0	1	0								

In this ticket, the `open`, `resolve`, `respond` and `close` events have already been executed and their timestamps have been recorded. The `close` attribute signifies that the ticket is already closed (the value of the `close` attribute is ‘1’). Therefore, we conclude that the current state of the ticket is `close` and based on the timestamp of each event, the event log/sequence for this ticket is {`open,respond,resolve,close`}.

P		Q		R		S		T		W		X		AE	
Incident No	Open Date	Resolve Date	Close Date	Respond Date	Open	Closed	Reopened_or_not								
RQ14279757	12/4/13 15:45	12/25/13 15:49			0	1	0								

In this second ticket, the `open` and `resolve` events have already been executed and their timestamps have been recorded but the `close` event has not been executed yet. However the `close` attribute the value of the `close` attribute is ‘1’ which signifies that the ticket is closed. Therefore, we conclude that the current state of the ticket is `auto-close` (close automatically by system by the system) and based on the timestamp of each event, the event log/sequence for this ticket is {`open,resolve,auto-close`}.

P		Q		R		S		T		W		X		AE	
Incident No	Open Date	Resolve Date	Close Date	Respond Date	Open	Closed	Reopened_or_not								
IN14308409	12/5/13 18:49			12/5/13 19:31	0	1	1								

In this last example, the `open` and `respond` events have already been executed. The `close` attribute value is ‘1’ which means that the ticket is closed but in the `reopened_or_not` attribute, the value is ‘1’ which signifies that the ticket has been reopened. Therefore, we conclude that the ticket is reopened and the event log/sequence for this ticket is {`open,respond,reopen,auto-close`}.

We use the goal assertions in Table 6.5 to recognize goal sequences from event sequences (these goal assertions were provided by domain experts from the organization - the authors might have articulated these goals somewhat differently).

We extract a goal sequence for each event sequence in Table 6.4. Recall that a goal is recognized in an event if the formal representation of the event entails the formal assertion of the goal. The complete list of goal sequences thus obtained is presented

Table 6.4: Event sequences identified in the log

Event sequence name	Event sequence	# of sequences
TR1	{open}, {open,respond}	1299
TR2	{open}, {open,respond, \neg receive}	4546
TR3	{open}, {open,respond}, {open,respond,close}	2
TR4	{open}, {open,respond}, {open,respond,resolve}, {open,respond,resolve,close}	53296
TR5	{open}, {open,resolve}, {open,resolve,auto-close}	128
TR6	{open}, {open,approved}	70
TR7	{open}, {open,respond}, {open,respond,receive}, {open,respond,receive, \neg reopen,auto-close}	25
TR8	{open}	296
TR9	{open}, {open, \neg approved}	383
TR10	{open}, {open,respond}, {open,respond,rejected}, {open,respond,rejected,close}	1
TR11	{open}, {open,respond}, {open,respond,reopen,auto-close}	37
TR12	{open}, {open,respond}, {open,respond,receive}, {open,respond,receive,incident,resolve,auto-close}	1195
TR13	{open}, {open,respond}, {open,respond,receive}, {open,respond,receive,resolve,auto-close}	3169
TR14	{open}, {open,respond}, {open,respond, \neg reopen,auto-close}	12
TR15	{open}, {open,respond}, {open,respond,receive}	531
TR16	{open}, {open,respond}, {open,respond, \neg instock}	10

Table 6.5: Goal assertions for the goal model

Goal	Goal assertion
Ticket Handled (G0)	close \vee auto-close
Ticket Initiated (G1)	open
Ticket Acknowledged and Problem Assigned (G2)	respond
Requirements provided (G3)	approved \vee receive \vee instock
DM Approval Acquired (G5)	approved
User Input Acquired (G6)	receive
Stock Acquired (G7)	instock
Unresolved Problem Handled (G9)	auto-close
Problem Resolved (G10)	resolve
Request Fulfilled (G11)	request \wedge resolve
IncidentResolved (G12)	incident \wedge resolve
New Ticket Created (G13)	reopen
Problem Closed (G14)	\neg reopen

in Table 6.6.

Table 6.6: Goal sequence for effect trace

Event sequence name	Goal sequence
TR1	(G1), (G1, G2)
TR2	(G1), (G1, N/A)
TR3	(G1), (G1, G2), (G1, G2, G0)
TR4	(G1), (G1, G2), (G1, G2, G10), (G1, G2, G10, G0)
TR5	(G1), (G1, G10), (G1, G10, G9), (G1, G10, G9, G0)
TR6	(G1), (G1, G5)
TR7	(G1), (G1, G2), (G1, G2, G6, G3), (G1, G2, G6, G3, G14, G9, G0)
TR8	(G1)
TR9	(G1), (G1, N/A)
TR10	(G1), (G1, G2), (G1, G2, N/A), (G1, G2, N/A, G0)
TR11	(G1), (G1, G2), (G1, G2, G13, G9, G0)
TR12	(G1), (G1, G2), (G1, G2, G6, G3), (G1, G2, G6, G3, G12, G0)
TR13	(G1), (G1, G2), (G1, G2, G6, G3), (G1, G2, G6, G3, G11, G0)
TR14	(G1), (G1, G2), (G1, G2, G14, G9, G0)
TR15	(G1), (G1, G2), (G1, G2, G6, G3)
TR16	(G1), (G1, G2), (G1, G2, N/A)

For this exercise, the first check is towards the end effect scenario of each trace where in all traces, the end effect must satisfy any one of the goal in the goal model. We can determine from Table 6.6 that among the 16 distinct traces, the end effect of TR2, TR8, TR9 and TR16 do not conform to any goal. Therefore these four traces are considered as **exception**. Upon closer inspection, it reveals that some of these traces are not fault or error, but the process is not finished yet and the effects are simply some kind of intermediate state. For example in TR2 where the end effect is `¬receive`, the state is to identify that the process is still waiting for user input and has not received any at the observed time.

The next check would be whether any one of the effect in the trace conform to a goal. By annotating each effect, we discover that the effect `rejected` of TR10 does not conform to any goal, therefore we annotate this trace as **exception**, while the 12 other traces are annotated as **normal**.

The last check is to examine whether in the normal trace, the goal precedence constraints in each trace is preserved. We perform the checking between any two consecutive goals (pair-wise) in the trace. From eight normal traces, we found that all

of them are preserving the goal precedence constraints.

Based on our examination, we omit the TR2, TR8, TR9 and TR16 as exceptions and only TR10 is considered as an exception that need to be handled further. We need to find any other normal trace with identical prefix and suffix of the effect `rejected`. We found this in trace TR4. The prefix `open` and `respond` and the suffix `close` are identical. Therefore we conclude that TR10 might be an alternative to trace TR4. We also check the number of ticket with this state and we discover that among 65000 tickets, there is only one ticket with this distinct trace and it will need further input from the IT team or the company to decide whether the state will be considered as another goal and incorporated in the goal model, or categorized as error or fault and should not be encounter again in the future. In the case that the state is regarded as a goal, it should be considered as an alternative of the `Resolve Problem` goal (probably as `Reject Problem`) and inserted with OR-refinement under the `Handle Problem` goal.

Mining Exercise. After we obtains goal sequences from the normal traces in the log, next we build the goal orchestrations based on them using the algorithm in Section 6.4. For this exercise, we only use the complete trace, that is all traces that end in the highest goal (G0), therefore we have eight traces to build the orchestrations. Note that we do not take into consideration the frequency of each trace, only the ordering of goals in a trace. We utilize ProM [241] to mine the workflow net and convert the result goal orchestrations using the conversion plugin.

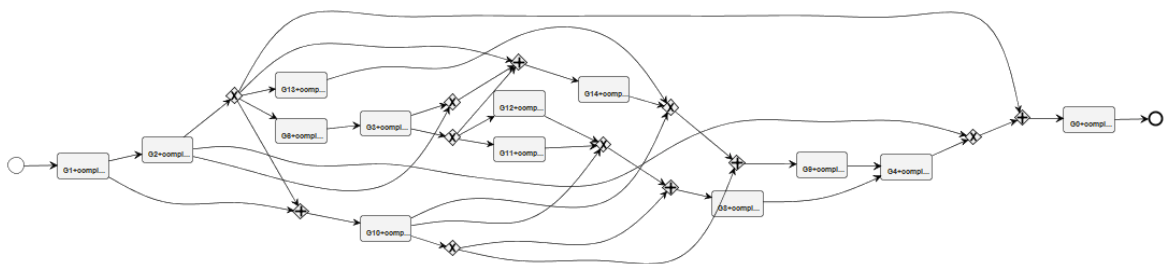


Figure 6.4: Goal orchestrations for ticket handling process

To determine the consistency between discovered the goal orchestrations with the goal model, we need to establish that all goals in the goal orchestrations presents in the goal model and all transitions preserves the goal precedence constraints in the goal model.

Looking at the goal orchestrations in Figure 6.4, there are 13 goals in the goal orchestrations. We confirm that they are also goals in the goal model. The next checking compares the precedence constraints in our library with the transitions in the discovered model. There are 23 transitions between goals in the goal orchestration. Eight of the transitions have a precedence constraint related to them. The checking reveals that these transitions conform to the precedence constraints. The rest of the transitions do not have any constraints related to them. Take for instance, the transitions between **G1** and **G2**. In the goal model, both are sub-goals of **G0**, thus both have precedence over their parent goal, but there is no constraint defined between **G1** and **G2**. Since there is no violation against the goal precedence constraints, we conclude that the discovered goal orchestrations is consistent with the goal model.

Generalization and the threats to validity. Firstly, in our evaluation, we assume that the semantic annotations or the post-conditions of the tasks or activities in the business process model and the formulation of goals are applicable for any given domain. We demonstrate this in our evaluation by adopting the states and state transitions directly from the available records. The records describe the execution history and contain standard property associated with events or activities. Secondly, the consistency checking is performed in two parts, the first is between the available execution data, in the form of effect traces, with the goal model, and the second is between the discovered model with the goal model. Our approach in this aspect is generalized in terms of goal models or event logs of any scale or any domain. This is demonstrated

with the scale of our case study involving 65000 records. Therefore, we argue that the threat to its validity (both construct and internal) in terms of systematic errors or data measurement is minimal. The main limitation we recognize in our approach is incompletely annotated process models, which will produce uncomplete or incorrect logs, or incompletely annotated goal models, which will contribute in incomparable effects between logs and goals. However, ensuring the completeness of annotations can help address this limitation and will contribute in correctly correlate goal models and available event logs.

6.6 Summary

In this chapter, a representation of business process as a coordination of goals called goal orchestrations, is proposed. This representation gives us a flexible and context sensitive enactment of processes and convenient for a goal-driven and knowledge-intensive process. A simple method of mining goal orchestrations from event logs is also presented. This method is illustrated using a real world setting of a ticket handling system. In the future work, we would like to further explore the mining of goal orchestrations and implement the concept in other application domains, more specifically in clinical setting.

Acknowledgment: This section has been published in the 36th International Conference on Conceptual Modelling (ER 2017) at Valencia Spain on November 6th-9th, 2017. This was a collaboration work between the author (Metta Santiputri) with the author's supervisors (Prof. Aditya Ghose and Dr. Hoa Khanh Dam). The author's supervisors provided the guidance specifically in the direction of the work and the final reviews before the publication. All the other works were performed by the author.

Chapter 7

Conclusion and future work

This thesis is summarized with a number of conclusions in Section 7.1. Limitations of the current approach are discussed in Section 7.2. These limitations give further opportunities for future works, as outlined in Section 7.3.

7.1 Conclusion

In this thesis, the aim is to generate models or knowledge drivers from enterprise data to enable some type of dashboard view of enterprise and to provide supports for analysts. The questions in Chapter 1 are addressed by leveraging process execution histories recorded in an enterprise repositories and performing the methods to produce reasonable results. The details of each remark are presented below.

1. Semantic annotation of business process model in the business process designs has been addressed in a large and growing body of work, but these annotations can be difficult and expensive to acquire. We presented a data-driven approach to mining and validating these annotations (and specifically context-independent semantic annotations). We leverage events in process execution histories which describe both activity execution events (recorded in *process logs*) and state up-

date events (recorded in *event logs*). A *sequential rule mining* algorithm and *abductive approach* were applied in the approach. An empirical evaluation were also presented, which suggests that the approach provides generally reliable results.

2. With regards to identifying process semantics, a formalization of annotation of process models were also provided by way of accumulating effects of individual tasks specified by analysts using belief bases and computing the accumulated effect up to the point of execution of the process model in an automated manner. This technique permits the analyst to specify immediate effect annotations in a practitioner-accessible simple propositional logic formulas and generates a sequence of tasks along with cumulative effects. Further a method were proposed in which given an effect log, it discovered the process model with effect annotations of individual tasks which is close to the original annotated process model using *algorithm which exploits the temporal sequence of the effects*.
3. Requirements acquisition is widely recognized as a hard problem, requiring significant investments in time and effort. Given the availability of large volumes of data and of relatively cheap instrumentation for data acquisition, the prospect of data-driven model extraction in the context of i^* models, an early-phase requirements modeling framework, were explored. The techniques were presented for extracting dependencies from *message logs*, and for extracting task-dependency correlations from *process logs* by performing a *sequential pattern mining* algorithm to the logs. A domain model, or a model of the “as-is”, were mined, but not requirements or goals in the minds of stakeholders that have no manifestation in data.
4. Key challenges in defining an enterprise architecture are specifying concepts/enti-

ties (such as a role, a service, a task or a goal) and their inter-relationships (such as the relationship between a role and a task or between elements of different layers in an architecture). Of the two, extracting relationships between concepts/entities is more challenging. A method to mine the relationships between business layer and application layer of an ArchiMate model from events recorded in *process logs* and the corresponding function calls recorded in *invocation logs* was presented. A *closed sequential pattern mining* algorithm was leveraged in the method.

5. In many application domains, it is more natural to think of a process as a coordination model of goals to be achieved rather than tasks or activities to be performed. Replacing tasks or activities with goals in process models allows us to enact processes in flexible, context-sensitive ways. It was showed how a goal orchestrations and a goal model constraint each other, and how these enable flexible process management. A simple means of mining goal orchestrations from readily available *event logs* was also provided.

The results of the points above conclude that it is possible to derive the knowledge drivers from the enterprise data. The data referred here is the historical data in the running operation of the enterprise, in the form of event log, process log, effect log, message log and invocation log. The models that were mined from these data represent the desired enterprise operational or “knowledge drivers”. By mining these knowledge drivers through data-driven extraction, the abstraction of the underlying reality can be acquired.

7.2 Limitation

The main drawback in the approach and methodology currently is caused by the lack of open large-scale comprehensive case study to test the framework in entirety. This difficulty was accomplished by evaluating each section of the framework independently. Nevertheless, it was maintained that the evaluation is adequate as a proof-of-concept tool. However, the availability of a single case study for the whole framework will help the for better evaluations.

With respect to the models that extracted or mined through the approach, some of the elements or aspects were not addressed. For example, in the i^* model, out of four types of dependencies, only one type of dependency, i.e., goal dependency, was defined. It was argued that the current approach is enough for the objective, however this leaves room for further improvement.

7.3 Future work

With regard to the limitations, we outline several lines of research towards improvement of the approach. First, performing experiments or evaluations using a comprehensive case-studies, preferably taken from a real-life industry, should be considered. With the coverage of the case-study, it should be able to specifically improve the evaluation of the approach. Second, if the case study also provides data other than the ones we have explored in this thesis, it will allow future research to improve the framework towards mining either more elements of the current models, such as in the i^* model previously mentioned, or additional models, such as goal models. Third, in respect to mining more elements of the current models, there are also a possible research towards refining the current method with more advance algorithms or methods, such as more sophisticated NLP techniques in goal mining.

References

- [1] Camille Ben Achour, Mustapha Tawbi, and Carine Souveyet. Bridging the Gap Between Users and Requirements Engineering: the Scenario-Based Approach. Technical Report CREWS Report Series 99-07, CRI Université de Paris 1 - Sorbonne, Paris, France, 1999.
- [2] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. Depth First Generation of Long Patterns. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 108–118, Boston, Massachusetts, USA, 2000. ACM.
- [3] Ramesh C. Agarwal, Charu C. Aggarwal, and V.V.V. Prasad. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.
- [4] Ritu Agarwal and Mohan R Tanniru. Knowledge Acquisition Using Structured Interviewing: An Empirical Investigation. *Journal of Management Information Systems*, 7(1):123–140, 1990.
- [5] Charu C Aggarwal and Jiawei Han. *Frequent Pattern Mining*. Springer, 2014.
- [6] R. Agrawal, T. Imielinski, and A. Swami. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.
- [7] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, pages 469–483. Springer-Verlag, 1998.
- [8] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

-
- [9] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A Inkeri Verkamo, et al. Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, 12(1):307–328, 1996.
- [10] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, ICDE'95*, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.
- [11] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
- [12] Stephan Aier, Bettina Gleichauf, and Robert Winter. Understanding Enterprise Architecture Management Design An Empirical Analysis. *Wirtschaftsinformatik Proceedings 2011*, January 2011.
- [13] Budoor Allehyani and Stephan Reiff-Marganiec. Maintaining Goals of Business Processes during Runtime Reconfigurations. In *Proceedings of the 8th ZEUS Workshop, 2016.*, pages 21–28, Vienna, Austria, January 2016.
- [14] Annie I. Antón. Goal-Based Requirements Analysis. In *Proceedings of the 2nd International Conference on Requirements Engineering, RE'96*, pages 136–144, Colorado Springs, Colorado, USA, 1996. IEEE Computer Society.
- [15] Annie I. Antón, Ryan A. Carter, Aldo Dagnino, John H. Dempster, and Devon F. Siegel. Deriving Goals from a Use-Case Based Requirements Specification. *Requirements Engineering*, 6(1):63–73, 2001.
- [16] Annie I. Antón and Julia Brande Earp. A Requirements Taxonomy for Reducing Website Privacy Vulnerabilities. *Requirements Engineering*, 9(3):169–185, 2004.
- [17] Annie I. Antón and Colin Potts. The Use of Goals to Surface Requirements for Evolving Systems. In *Proceedings of the 20th International Conference on Software Engineering, ICSE 1998*, pages 157–166, Kyoto, Japan, April 1998. ACM.
- [18] Danilo Ardagna and Barbara Pernici. Adaptive Service Composition in Flexible Processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007.
- [19] Paul Arkley and Sean Riddle. Tailoring Traceability Information to Business Needs. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE'06*, pages 239–244, Minneapolis-St. Paul, Minnesota, USA, September 2006. IEEE Computer Society.

-
- [20] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential PAttern Mining Using a Bitmap Representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 429–435, Edmonton, Alberta, Canada, 2002. ACM.
- [21] Linden J. Ball and Thomas C. Ormerod. Putting Ethnography To Work: The Case For A Cognitive Ethnography Of Design. *International Journal of Human-Computer Studies*, 53(1):147–168, 2000.
- [22] Thaís Vasconcelos Batista, M. Cecilia Bastarrica, Sérgio Soares, and Lyrene Fernandes da Silva. A Marriage of MDD and Early Aspects in Software Product Line Development. In *Proceedings of the 2008 Workshop on Early Aspects: Aspect-Oriented Requirements and Architecture for Product Lines, Second Volume*, EA@SPLC.08, in conjunction with SPLC 2008, pages 97–103, Limerick, Ireland, September 2008.
- [23] Roberto J. Bayardo, Jr. Efficiently Mining Long Patterns from Databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 85–93, Seattle, Washington, USA, 1998. ACM.
- [24] Kent Beck and Ward Cunningham. A Laboratory for Teaching Object Oriented Thinking. *SIGPLAN Notes*, 24(10):1–6, 1989.
- [25] Kent Beck and Ward Cunningham. A laboratory for teaching object oriented thinking. In *Proceedings of the Conference on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '89, pages 1–6, New Orleans, Louisiana, USA, 1989. ACM.
- [26] Jörg Becker, Michael Rosemann, and Christoph Von Uthmann. Guidelines of business process modeling. In *Business Process Management*, pages 30–49. Springer, 2000.
- [27] Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, Sherif Sakr, Daniela Grigori, Hamid Reza Motahari-Nezhad, Moshe Chai Barukh, Ahmed Gater, and Seung Hwan Ryu. *Model-Based Business Process Query Techniques and Languages*, pages 91–106. Springer International Publishing, 2016.
- [28] Bernard Marr. How Big Data And Analytics Are Transforming The Construction Industry. <http://www.gereports.com/post/108012387578/2015-a-big-year-for-big-data/>, April 2016.

-
- [29] Dianne C. Berry and Donald E. Broadbent. Expert systems and the man-machine interface. Part Two: The user interface. *Expert Systems*, 4(1):18–27, February 1987.
- [30] P. Bertrand, Robert Darimont, Emmanuelle Delor, Philippe Massonet, and Axel van Lamswerde. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. In *Proceedings of the 20th International Conference on Software Engineering, ICSE'98*, Kyoto, Japan, April 1998. IEEE-ACM.
- [31] Hugh Beyer and Karen Holtzblatt. *Contextual design: defining customer-centered systems*. Elsevier, 1997.
- [32] Hugh R. Beyer and Karen Holtzblatt. Apprenticing with the Customer. *Communication of the ACM*, 38(5):45–52, May 1995.
- [33] BiZZdesign BV. Enterprise Studio — BiZZdesign. <http://www.bizzdesign.com/tools/bizzdesign-architect/>, 2016.
- [34] Steven J. Bleistein, Karl Cox, and June Verner. Strategic Alignment in Requirements Analysis for Organizational IT: An Integrated Approach. In *Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05*, pages 1300–1307, Santa Fe, New Mexico, 2005. ACM.
- [35] Andreas Bögl, Michael Schrefl, Gustav Pomberger, and Norbert Weber. Automated Construction of Process Goal Trees from EPC-Models to Facilitate Extraction of Process Patterns. In *Proceedings of the 11th International Conference on Enterprise Information Systems, ICEIS 2009*, pages 427–442, Milan, Italy, May 2009. Springer Berlin Heidelberg.
- [36] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In *Proceedings of the 15th International Conference on Computational Statistics, Compstat 2002*, pages 395–400, Berlin, Germany, August 2002. Springer.
- [37] Matthias Born, Florian Dörr, and Ingo Weber. User-Friendly Semantic Annotation in Business Process Modeling. In *Proceedings of the 8th International Conference on Web Information Systems Engineering Workshops, WISE 2007*, Nancy, France, December 2007.
- [38] Travis D Breaux and Annie I Antón. Analyzing Regulatory Rules for Privacy and Security Requirements. *IEEE Transactions on Software Engineering*, 34(1):5–20, 2008.
- [39] Doug Burdick, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu. MAFIA: A Maximal Frequent Itemset Algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1490–1504, November 2005.

-
- [40] Markus Buschle, Hannes Holm, Teodor Sommestad, Mathias Ekstedt, and Khurram Shahzad. A Tool for automatic Enterprise Architecture modeling. In *IS Olympics: Information Systems in a Diverse World: CAiSE Forum 2011 Selected Extended Papers*, pages 1–15. Springer, London, UK, June 2011.
- [41] Business Process Management Initiative (BPMI.org). Business Process Modeling Language, 2003.
- [42] Evellin C. S. Cardoso, João Paulo A. Almeida, Giancarlo Guizzardi, and Renata S. S. Guizzardi. Eliciting goals for business process models with non-functional requirements catalogues. In *Proceedings of the 10th Workshop on Business Process Modeling, Development, and Support, BPMDS'09*, in conjunction with CAiSE'09, pages 33–45, Amsterdam, The Netherlands, June 2009. Springer Berlin Heidelberg.
- [43] Aaron Ceglar and John F. Roddick. Association mining. *ACM Computing Surveys*, 38(2), July 2006.
- [44] Jane Cleland-Huang, Raffaella Settini, Chuan Duan, and Xuchang Zou. Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability. In *Proceedings of the 13th IEEE International Requirements Engineering Conference, RE'05*, pages 135–144, Paris, France, August 2005. IEEE Computer Society.
- [45] Jonathan E Cook and Alexander L Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.
- [46] Jonathan E Cook and Alexander L Wolf. *Event-based detection of concurrency*, volume 23. ACM, 1998.
- [47] Corso. Enterprise Architecture Software Tools — Corso. <http://www.corso3.com/enterprise-architecture>, 2016.
- [48] Bill Curtis, Marc I. Kellner, and Jim Over. Process Modeling. *Communication of the ACM*, 35(9):75–90, September 1992.
- [49] Fabiano Dalpiaz, Evellin Cardoso, Giulia Canobbio, Paolo Giorgini, and John Mylopoulos. Social Specifications of Business Processes with Azzurra. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 7–18, May 2015.

-
- [50] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. iStar 2.0 Language Guide. *CoRR*, abs/1605.07767, 2016.
- [51] Elio Damaggio, Richard Hull, and Roman Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard–stage–milestone lifecycles. *Information Systems*, 38(4):561–584, 2013.
- [52] Christophe Damas, Bernard Lambeau, Pierre Dupont, and Axel Van Lamsweerde. Generating Annotated Behavior Models from End-User Scenarios. *IEEE Transactions on Software Engineering*, 31(12):1056–1073, December 2005.
- [53] Christophe Damas, Bernard Lambeau, and Axel van Lamsweerde. Scenarios, Goals, and State Machines: A Win-Win Partnership for Model Synthesis. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT 2006, pages 197–207, Portland, Oregon, USA, 2006. ACM.
- [54] Anne Dardenne, Stephen Fickas, and Axel van Lamsweerde. Goal-directed Concept Acquisition in Requirements Elicitation. In *Proceedings of the 6th International Workshop on Software Specification and Design*, IWSSD '91, pages 14–21, Como, Italy, 1991. IEEE Computer Society Press.
- [55] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1):3–50, 1993.
- [56] Robert Darimont. *Process Support for Requirements Elaboration*. PhD thesis, Université catholique de Louvain, Louvain-La-Neuve, Belgium, 1995.
- [57] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-driven Requirements Elaboration. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '96, pages 179–190, San Francisco, California, USA, 1996. ACM.
- [58] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns for Goal-driven Requirements Elaboration. *SIGSOFT Software Engineering Notes*, 21(6):179–190, November 1996.
- [59] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule Discovery from Time Series. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 16–22, New York, New York, USA, 1998.

-
- [60] Jose Luis de la Vara, Juan Sánchez, and Oscar Pastor. On the Use of Goal Models and Business Process Models for Elicitation of System Requirements. In *Proceedings of the 14th International Conference on Business Process Modeling, Development, and Support, BPMDS 2013*, in conjunction with CAiSE 2013, pages 168–183, Valencia, Spain, June 2013. Springer Berlin Heidelberg.
- [61] Jose Luis De la Vara González and Juan Sánchez Díaz. Business process-driven requirements engineering: a goal-based approach. In *Proceedings of the 8th Workshop on Business Process Modeling, Development, and Support, BPMDS'07*, in conjunction with CAiSE07, Trondheim, Norway, June 2007.
- [62] Jitender Deogun and Liying Jiang. Prediction Mining – An Approach to Mining Association Rules for Prediction. In *Proceedings of the 10th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, RSFDGrC 2005*, pages 98–108, Regina, Canada, August 2005. Springer Berlin Heidelberg.
- [63] Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Paolo Tonella. Reasoning on Semantically Annotated Processes. In *Proceedings of the 6th International Conference on Service Oriented Computing, ICSOC 2008*, pages 132–146, Sydney, Australia, December 2008. Springer Berlin Heidelberg.
- [64] Ivan Di Pietro, Francesco Pagliarecci, and Luca Spalazzi. Model Checking Semantically Annotated Services. *IEEE Transactions on software engineering*, 38(3):592–608, 2012.
- [65] Merriam-Webster Dictionary. Process — definition of process. <http://www.merriam-webster.com/dictionary/process>.
- [66] Amal Elgammal, Oktay Turetken, Willem-Jan van den Heuvel, and Mike Papazoglou. Formalizing and Applying Compliance Patterns for Business Process Compliance. *Software & Systems Modeling*, 15(1):119–146, 2016.
- [67] Neil A Ernst, John Mylopoulos, Yijun Yu, and Tien Nguyen. Supporting Requirements Model Evolution Throughout the System Life-cycle. In *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08*, pages 321–322, Barcelona, Spain, September 2008. IEEE Computer Society.
- [68] Matthias Farwick, Ruth Breu, Matheus Hauder, Sascha Roth, and Florian Matthes. Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation. In

- Proceedings of the 46th Hawaii International Conference on System Sciences*, HICSS '13, pages 3868–3877, January 2013.
- [69] Walid Fdhila, Stefanie Rinderle-Ma, and Conrad Indiono. Memetic Algorithms for Mining Change Logs in Process Choreographies. In *Proceeding of the 12th International Conference on Service-Oriented Computing*, ICSOC 2014, pages 47–62, Paris, France, November 2014. Springer Berlin Heidelberg.
- [70] Dieter Fensel, Federico Michele Facca, Elena Simperl, and Ioan Toma. *Semantic Web Services*. Springer, 2011.
- [71] Dieter Fensel, Holger Lausen, Axel Polleres, Jos De Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling semantic web services: the web service modeling ontology*. Springer Science & Business Media, 2006.
- [72] Günther Fliedl, Christian Kop, and Heinrich C Mayr. From Textual Scenarios to A Conceptual Schema. *Data & Knowledge Engineering*, 55(1):20–37, 2005.
- [73] Philippe Fournier-Viger, Usef Faghihi, Roger Nkambou, and Engelbert Mephu Nguifo. CM-Rules: Mining sequential rules common to several sequences. *Knowledge-Based Systems: Special Issue on New Trends in Data Mining*, 25(1):63–76, 2012.
- [74] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In *Proceedings of The 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, PAKDD 2014, pages 40–52, Tainan, Taiwan, May 2014. Springer International Publishing.
- [75] Philippe Fournier-Viger, Ted Gueniche, Souleymane Zida, and Vincent S. Tseng. ERMiner: Sequential Rule Mining Using Equivalence Classes. In *Proceedings of the 13th International Symposium on Advances in Intelligent Data Analysis*, IDA 2014, pages 108–119, Leuven, Belgium, October 2014. Springer International Publishing.
- [76] Philippe Fournier-Viger, Roger Nkambou, and Vincent Shin-Mu Tseng. RuleGrowth: Mining Sequential Rules Common to Several Sequences by Pattern-growth. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 956–961. ACM, 2011.
- [77] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S. Tseng. VMSP: Efficient Vertical Mining of Maximal Sequential Patterns. In *Proceedings of the 27th Canadian*

- Conference on Artificial Intelligence*, Canadian AI 2014, pages 83–94, Montréal, QC, Canada, May 2014. Springer International Publishing.
- [78] Philippe Fournier-Viger, Cheng-Wei Wu, and Vincent S. Tseng. Mining Maximal Sequential Patterns without Candidate Maintenance. In *Proceedings of the 9th International Conference on Advanced Data Mining and Applications*, ADMA 2013, pages 169–180, Hangzhou, China, December 2013. Springer Berlin Heidelberg.
- [79] Xavier Franch, Lidia López, Carlos Cares, and Daniel Colomer. The i* Framework for Goal-Oriented Modeling. In *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, pages 485–506. Springer International Publishing, 2016.
- [80] Shang Gao. Relating Goal Modeling with BPCM Models in a Combined Framework. In *Proceedings of the 13th International Conference on Computational Science and Its Applications, Part III*, ICCSA 2013, pages 33–42, Ho Chi Minh City, Vietnam, June 2013. Springer Berlin Heidelberg.
- [81] Shang Gao and John Krogstie. A Combined Framework for Development of Business Process Support Systems. In *Proceedings of The Practice of Enterprise Modeling: Second IFIP WG 8.1 Working Conference*, PoEM 2009, pages 115–129, Stockholm, Sweden, November 2009. Springer Berlin Heidelberg.
- [82] René Arnulfo García-Hernández, José Francisco Martínez-Trinidad, and Jesús Ariel Carrasco-Ochoa. A New Algorithm for Fast Discovery of Maximal Sequential Patterns in a Document Collection. In *Proceedings of the 7th International Conference Computational Linguistics and Intelligent Text Processing*, CICLing 2006, pages 514–523, Mexico City, Mexico, February 2006. Springer Berlin Heidelberg.
- [83] GE Reports Staff. 2015 A Big Year for Big Data. <http://www.gereports.com/post/108012387578/2015-a-big-year-for-big-data/>, January 2015.
- [84] Sepideh Ghanavati, Daniel Amyot, and Liam Peyton. Compliance analysis based on a goal-oriented requirement language evaluation methodology. In *Proceedings of the 17th IEEE International Requirements Engineering Conference*, RE'09, pages 133–142, Atlanta, Georgia, USA, August 2009. IEEE Computer Society.
- [85] Chiara Ghidini, Marco Rospocher, and Luciano Serafini. A formalisation of BPMN in description logics. *FBK-irst, Tech. Rep. TR*, pages 06–004, 2008.

-
- [86] Aditya Ghose and George Koliadis. Auditing Business Process Compliance. In *Proceedings of the Fifth International Conference on Service-Oriented Computing*, ICSOC 2007, pages 169–180, Vienna, Austria, September 2007. Springer Berlin Heidelberg.
- [87] Aditya Ghose, George Koliadis, and Arthur Chueng. Process Discovery from Model and Text Artefacts. In *Proceedings of the 2007 IEEE Congress on Services*, pages 167–174, Salt Lake City, Utah, USA, July 2007. IEEE.
- [88] Aditya K Ghose, Evan Morrison, and Yingzhi Gou. A Novel Use of Big Data Analytics for Service Innovation Harvesting. In *Proceedings of the 5th International Conference on Service Science and Innovation*, ICSSI 2013, pages 208–214, Kaohsiung, Taiwan, May 2013. IEEE Computer Society.
- [89] Brian K Gibb and Suresh Damodaran. *ebXML: Concepts and application*. John Wiley & Sons, Inc., 2002.
- [90] Matthew L. Ginsberg and David E. Smith. Reasoning about Action I: A Possible Worlds Approach. *Artificial Intelligence*, 35(2):165–195, 1988.
- [91] Octavio Glorio, Jesus Pardillo, Jose-Norberto Mazon, and Juan Trujillo. Dawara: An eclipse plugin for using i* on data warehouse requirement analysis. In *Proceedings of the 16th IEEE International Requirements Engineering Conference*, RE'08, pages 317–318, Barcelona, Spain, September 2008. IEEE Computer Society.
- [92] Bart Goethals. Survey on Frequent Pattern Mining. Technical report, University of Helsinki, Helsinki, Finland, 2003.
- [93] J. A. Goguen and C. Linde. Techniques for Requirements Elicitation. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, RE'93, pages 152–164, San Diego, CA, USA, January 1993.
- [94] Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences. In *Proceedings of The 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, PAKDD 2013, pages 50–61, Gold Coast, Australia, April 2013. Springer Berlin Heidelberg.
- [95] Raj P Gopalan and Yudho Giri Suchahyo. High performance frequent patterns extraction using compressed FP-Tree. In *Proceedings of 2004 SIAM International Workshop on High Performance and Distributed Mining*, HPDM'04, Orlando, USA, 2004.

- [96] Ellen Gottesdiener. *Requirements by Collaboration: Workshops for Defining Needs*. Addison-Wesley Professional, 2002.
- [97] Yingzhi Gou, Aditya Ghose, Chee-Fon Chang, Hoa Khanh Dam, and Andrew Miller. Semantic Monitoring and Compensation in Socio-technical Processes. In *Proceedings of the 1st International Workshop on Conceptual Modeling in Requirements and Business Analysis*, MReBA14, in conjunction with ER 2014, Atlanta, GA, USA, October 2014. Springer International Publishing.
- [98] Gösta Grahne and Jianfei Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, FIMI'03, Melbourne, Florida, USA, November 2003. IEEE Computer Society.
- [99] Gösta Grahne and Jianfei Zhu. Fast Algorithms for Frequent Itemset Mining Using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, October 2005.
- [100] Gösta Grahne and Jianfei Zhu. Fast Algorithms for Frequent Itemset Mining Using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, October 2005.
- [101] Gemma Grau, Xavier Franch, and Neil A.M. Maiden. PRiM: An i-based process reengineering method for information systems specification. *Information and Software Technology*, 50(12):76–100, 2008.
- [102] Georg Grossmann, Michael Schrefl, and Markus Stumptner. A model-driven Framework for Runtime Adaptation of Web Service Compositions. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 184–189. ACM, 2011.
- [103] The Stanford NLP Group. StanfordNLP (Stanford Natural Language Processing) Software. <http://nlp.stanford.edu/software/tagger.shtml>.
- [104] Susan Gunelius. The Data Explosion in 2014 Minute by Minute Infographic. <https://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/>, July 2014.
- [105] Christian W Günther and Wil M.P. Van Der Aalst. Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In *Proceedings of the 5th International Conference on Business Process Management*, BPM 2007, pages 328–343, Brisbane, Australia, September 2007. Springer.

- [106] Curtis Hall and Paul Harmon. The 2005 Enterprise Architecture, Process Modeling & Simulation Tools Report. Technical report, Business Process Trends, 2005.
- [107] Michael Hammer and James Champy. *A Reengineering the Corporation: Manifesto for Business Revolution*. Collins Business Essentials. HarperCollins, 2009.
- [108] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, July 2007.
- [109] Jiawei Han and Jian Pei. Mining frequent patterns by pattern-growth: Methodology and implications. *ACM SIGKDD Explorations Newsletter - Special issue on “Scalable data mining algorithms”*, 2(2):14–20, December 2000.
- [110] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. FreeSpan: Frequent Pattern-projected Sequential Pattern Mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 355–359, Boston, Massachusetts, USA, 2000. ACM.
- [111] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.
- [112] Hans-Jörg Happel and Ljiljana Stojanovic. Ontoprocess—a Prototype for Semantic Business Process Verification Using SWRL rules. In *Proceedings of the 3rd European Semantic Web Conference*, ESWC 2006, Budva, Montenegro, June 2006.
- [113] Paul Harmon and Celia Wolf. The State of Business Process Management 2016. <http://www.bptrends.com/bpt/wp-content/uploads/2015-BPT-Survey-Report.pdf>, 2015.
- [114] Sherri K. Harms, Jitender Deogun, and Tsegaye Tadesse. Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences. In *Proceedings of the 13th International Symposium on Foundations of Intelligent Systems*, ISMIS 2002, pages 432–441, Lyon, France, June 2002. Springer Berlin Heidelberg.
- [115] Peter Haumer, Klaus Pohl, and Klaus Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering*, 24(12):1036–1054, December 1998.
- [116] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, and Sarah Howard. Helping Analysts Trace Requirements: An objective look. In *Proceedings of 12th IEEE International*

- Requirements Engineering Conference*, RE'04, pages 249–259, Kyoto, Japan, September 2004. IEEE Computer Society.
- [117] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic Business Process Management: A Vision Towards Using for Business Process Management. In *Proceedings of the IEEE International Conference on e-Business Engineering*, ICEBE 2005, pages 535–540, Beijing, China, October 2005. IEEE Computing Society.
- [118] Nico Herzberg, Matthias Kunze, and Andreas Rogge-Solti. Towards process evaluation in non-automated process execution environments. In *ZEUS*, pages 97–103. Citeseer, 2012.
- [119] Nico Herzberg and Andreas Meyer. Improving process monitoring and progress prediction with data state transition events. *Data & Knowledge Engineering*, 98:144–164, 2015.
- [120] Nico Herzberg, Andreas Meyer, and Mathias Weske. An event processing platform for business process management. In *Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 107–116. IEEE, 2013.
- [121] Nico Herzberg, Andreas Meyer, and Mathias Weske. Improving business process intelligence by observing object state transitions. *Proceedings of the 32th International Conference on Conceptual Modeling (ER 2013)*, pages 146–160, 2013.
- [122] Nico Herzberg and Mathias Weske. Enriching raw events to enable process intelligence: research challenges. Technical Report 73, Universitätsverlag Potsdam, 2013.
- [123] Kerry Hinge, Aditya Ghose, and George Koliadis. Process SEER: A Tool for Semantic Effect Annotation of Business Process Models. In *Proceedings of the Thirteenth IEEE International Enterprise Distributed Object Computing Conference*, EDOC '09, pages 54–63, Auckland, New Zealand, September 2009. IEEE Computing Society.
- [124] Jörg Hoffmann, Ingo Weber, and Guido Governatori. On compliance checking for clausal constraints in annotated process models. *Information Systems Frontiers*, 14(2):155–177, 2012.
- [125] Hannes Holm, Markus Buschle, Robert Lagerström, and Mathias Ekstedt. Automatic data collection for enterprise architecture models. *Software & Systems Modeling*, 13(2), 2014.
- [126] Karen Holtzblatt and Hugh R. Beyer. Requirements Gathering: The Human Factor. *Commun. ACM*, 38(5):31–32, May 1995.

- [127] Jennifer Horkoff, Alex Borgida, John Mylopoulos, Daniele Barone, Lei Jiang, Eric Yu, and Daniel Amyot. Making Data Meaningful: The business intelligence model and its formal semantics in description logics. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7566 of *Lecture Notes in Computer Science*, pages 700–717. Springer, 2012.
- [128] Jennifer Horkoff, Tong Li, Feng-Lin Li, Mattia Salnitri, Evellin Cardoso, Paolo Giorgini, and John Mylopoulos. Using Goal Models Downstream: A Systematic Roadmap and Literature Review. *International Journal of Information System Modeling and Design (IJISMD)*, 6(2):1–42, 2015.
- [129] Thomas Hornung, Agnes Koschmider, and Andreas Oberweis. A recommender system for business process models. In *Proceedings of the 17th Annual Workshop on Information Technologies & Systems*, WITS 2009, 2009.
- [130] Bowen Hui and Eric Yu. Extracting conceptual relationships from specialized documents. *Data & Knowledge Engineering*, 54(1):29–55, 2005.
- [131] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *International Workshop on Web Services and Formal Methods*, pages 1–24. Springer Berlin Heidelberg, 2010.
- [132] IBM. Bringing big data to the enterprise. <http://wikibon.org/blog/big-data-statistics/>.
- [133] Silvia Ingolfo, Alberto Siena, and John Mylopoulos. Establishing Regulatory Compliance for Software Requirements. In *Proceedings of the 30th International Conference on Conceptual Modeling*, ER 2011, pages 47–61, Brussels, Belgium, October 2011. Springer Berlin Heidelberg.
- [134] Shareeful Islam, Haralambos Mouratidis, and Stefan Wagner. Towards a Framework to Elicit and Manage Security and Privacy Requirements from Laws and Regulations. In *Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ 2010, pages 255–261, Essen, Germany, June 2010. Springer Berlin Heidelberg.
- [135] Henk Jonkers, Iver Band, and Dick Quartel. The ArchiSurance Case Study. *White paper, The Open Group*, Spring, 2012.

- [136] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, et al. Business Process Execution Language for Web Services(BPEL4WS) 1.0. <https://docs.oasis-open.org/wsbpel/2.0/PR02/wsbpel-specification-draft-diff.pdf>, November 2006.
- [137] Yuchul Jung, Yoonsung Cho, Yoo-Mi Park, and TaeDong Lee. Automatic Tagging of Functional-Goals for Goal-Driven Semantic Service Discovery. In *Proceedings of the IEEE Seventh International Conference on Semantic Computing, ICSC 2013*, pages 212–219, Irvine, California, USA, September 2013. IEEE Computer Society.
- [138] Stefan Junginger. The Workflow Management Coalition Standard WPDL: First Steps Towards Formalization. In *Proceedings of the 7th European Concurrent Engineering Conference, ECEC'2000*, pages 163–168, Leicester, United Kingdom, April 2000.
- [139] Mohammed J.Zaki and Ching-Jui Hsiao. CHARM: An Efficient Algorithm for Closed Itemset Mining. In *Proceedings of 2002 SIAM International Conference on Data Mining, SDM'02*, Arlington, VA, USA, April 2002.
- [140] Stephen H. Kaisler, Frank Armour, and Michael Valivullah. Enterprise Architecting: Critical Problems. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS '05*, pages 224b–224b. IEEE, January 2005.
- [141] Lowell D Kaufman. *System Modeling For Scenario-Based Requirements Engineering*. University of Florida, 1989.
- [142] Evangelia Kavakli. Goal-Oriented Requirements Engineering: A Unifying Framework. *Requirements Engineering*, 6(4):237–251, 2002.
- [143] Nadzeya Kiyavitskaya and Nicola Zannone. Requirements model generation to support requirements elicitation: the Secure Tropos experience. *Automated Software Engineering*, 15(2):149–173, 2008.
- [144] George Koliadis and Aditya Ghose. Correlating Business Process and Organizational Models to Manage Change. In *Proceeding of the Australasian Conference on Information Systems, ACIS 2006*, Adelaide, SA, Australia, December 2006.
- [145] George Koliadis, Aditya Ghose, and Srinivas Padmanabhuni. Towards an enterprise business process architecture standard. In *Proceedings of the 2008 IEEE Congress on Services - Part I*, pages 239–246. IEEE, July 2008.

- [146] George Koliadis, Aleksander Vranesevic, Moshiur Bhuiyan, Aneesh Krishna, and Aditya Ghose. A Combined Approach for Supporting the Business Process Model Lifecycle. In *Proceeding of the 10th Pacific Asia Conference on Information Systems, PACIS'06*, Kuala Lumpur, Malaysia, July 2006.
- [147] George Koliadis, Aleksander Vranesevic, Moshiur Bhuiyan, Aneesh Krishna, and Aditya Ghose. Combining i* and bpmn for business process model lifecycle management. In *Proceedings of the 2006 International Workshop on Grid and Peer-to-Peer Based Workflows, GPWW 2006*, in conjunction with BPM 2006, pages 416–427, Vienna, Austria, September 2006. Springer Berlin Heidelberg.
- [148] Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.
- [149] Manolis Koubarakis and Dimitris Plexousakis. A Formal Framework for Business Process Modelling and Design. *Information Systems*, 27(5):299–319, 2002.
- [150] John Krogstie, Guttorm Sindre, and Håvard Jørgensen. Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems*, 15(1):91–102, 2006.
- [151] Richard A Krueger and Mary Anne Casey. *Focus groups: A practical guide for applied research*. Sage publications, 2014.
- [152] Qihua Lan, Defu Zhang, and Bo Wu. A New Algorithm for Frequent Itemsets Mining Based on Apriori and FP-Tree. In *Proceedings of the 2009 WRI Global Congress on Intelligent Systems*, volume 2, pages 360–364, May 2009.
- [153] Marc Lankhorst, editor. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [154] Marc Lankhorst, Henderik Alex Proper, and Henk Jonkers. The Anatomy of the ArchiMate Language. *International Journal of Information System Modeling and Design (IJISMD)*, 1(1):1–32, 2010.
- [155] Alexei Lapouchnian. Goal-Oriented Requirements Engineering: An Overview of the Current Research. Technical report, Department of Computer Science, University Of Toronto, Toronto, Canada, June 2005.

- [156] Chiung-Hon Leon Lee and Alan Liu. Toward Intention Aware Semantic Web Service Systems. In *Proceedings of the 2005 IEEE International Conference on Services Computing Volume 1*, volume 1 of *SCC'05*, pages 69–76, Orlando, Florida, USA, July 2005. IEEE Computer Society.
- [157] Emmanuel Letier. *Reasoning About Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université catholique de Louvain, Louvain-La-Neuve, Belgium, 2001.
- [158] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wang Wei, and Xiangye Xiao. AFOPT: An Efficient Implementation of Pattern Growth Approach. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, FIMI'03, Melbourne, Florida, USA, November 2003. IEEE Computer Society.
- [159] David Lo, Siau-Cheng Khoo, and Limsoon Wong. Non-redundant sequential rules Theory and algorithm. *Information Systems*, 34(45):438–453, 2009.
- [160] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Fast and Memory Efficient Mining of Frequent Closed Itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):21–36, January 2006.
- [161] Congnan Luo and Soon M. Chung. Efficient Mining of Maximal Sequential Patterns Using Multiple Samples. In *Proceedings of 2005 SIAM International Conference on Data Mining*, SDM'05, Newport Beach, CA, USA, April 2005.
- [162] Nizar R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys*, 43(1):3:1–3:41, December 2010.
- [163] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [164] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Efficient algorithms for discovering association rules. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, KDD-94, pages 181–192, Seattle, Washington, USA, July 1994.
- [165] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June 1994.
- [166] Sabrina Marczak and Daniela Damian. How Interaction between Roles Shapes the Communication Structure in Requirements-Driven Collaboration. In *Proceedings of the 19th IEEE International Requirements Engineering Conference*, pages 47–56, Trento, Italy, August 2011.

- [167] Andrea Marrella, Massimo Mecella, Alessandro Russo, Sebastian Steinau, Kevin Andrews, and Manfred Reichert. A Survey on Handling Data in Business Process Models (Discussion Paper). In *Proceedings of the 23rd Italian Symposium on Advanced Database Systems, SEBD 2015*, Gaeta, Italy, June 2015.
- [168] Andrea Marrella, Massimo Mecella, Alessandro Russo, Sebastian Steinau, Kevin Andrews, and Manfred Reichert. Data in Business Process Models, A Preliminary Empirical Study. In *Proceedings of the IEEE 8th International Conference on Service-Oriented Computing and Applications, SOCA 2015*, pages 116–122, Rome, Italy, October 2015.
- [169] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, et al. Bringing semantics to web services: The OWL-S approach. In *Semantic Web Services and Web Process Composition*, pages 26–42. Springer, 2005.
- [170] Florent Masseglia, F Cathala, and Pascal Poncelet. *The PSP Approach for Mining Sequential Patterns*, pages 176–184. PKDD '98. Springer Berlin Heidelberg, Nantes, France, September 1998.
- [171] Aaron K. Massey, Jacob Eisenstein, Annie I. Antón, and Peter P. Swire. Automated Text Mining for Requirements Analysis of Policy Documents. In *Proceedings of the 21st IEEE International Requirements Engineering Conference, RE'13*, pages 4–13, Rio de Janeiro, RJ, Brazil, 2013. IEEE Computer Society.
- [172] Masahiro Matsubara, Masataka Nishi, and Fumio Narisawa. Modeling safety requirements of iso26262 using goal trees and patterns. In *Formal Techniques for Safety-Critical Systems: Fourth International Workshop, FTSCS 2015, Paris, France, November 6-7, 2015. Revised Selected Papers*, volume 596, page 206. Springer, 2016.
- [173] William E McCarthy. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *Accounting Review*, pages 554–578, 1982.
- [174] John McDermid. Requirements analysis: Problems and the STARTS approach. In *IEE Colloquium on Requirements Capture and Specification for Critical Systems*, pages 4/1–4/4. IET, November 1989.
- [175] Karen L. McGraw and Karen Harbison-Briggs. *Knowledge Acquisition: Principles and Guidelines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

- [176] Odorico Machado Mendizabal, Martin Spier, and Rodrigo Saad. Log-based Approach for Performance Requirements Elicitation and Prioritization. In *Proceedings of the 20th IEEE International Requirements Engineering Conference, RE'12*, pages 297–302, Chicago, Illinois, USA, September 2012. IEEE Computer Society.
- [177] Jan Mendling, Hajo A. Reijers, and Wil M.P. van der Aalst. Seven process modeling guidelines (7PMG). *Information and Software Technology*, 52(2):127–136, 2010.
- [178] Harald Meyer. On the Semantics of Service Compositions. In *Web Reasoning and Rule Systems*, pages 31–42. Springer Berlin Heidelberg, 2007.
- [179] Hamed Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. Business Process Modeling Languages: Sorting Through the Alphabet Soup. *ACM Computing Surveys*, 43(1):4:1–4:56, November 2010.
- [180] Mehdi Mirakhorli, Ahmed Fakhry, Artem Grechko, Mateusz Wieloch, and Jane Cleland-Huang. Archie: A Tool for Detecting, Monitoring, and Preserving Architecturally Significant Code. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, pages 739–742, Hong Kong, China, 2014.
- [181] Marco Montali, Maja Pesic, Wil M.P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative specification and verification of service choreographies. *ACM Transactions on the Web*, 4:1–62, 2010.
- [182] Daniel L Moody. Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, 55(3):243–276, 2005.
- [183] Carl H. Mooney and John F. Roddick. Sequential pattern mining – approaches and algorithms. *ACM Computing Surveys*, 45(2):19:1–19:39, March 2013.
- [184] Evan D. Morrison, Aditya Ghose, Hoa K. Dam, Kerry Hinge, and Konstantin Hoesch-Klohe. Strategic Alignment of Business Processes. In *Proceedings of the Seventh International Workshop on Engineering Service-Oriented Applications, WESOA 2011*, in conjunction with ICSOC 2011, pages 9–21, Paphos, Cyprus, December 2012. Springer Berlin Heidelberg.
- [185] John Mylopoulos, Lawrence Chung, and Eric Yu. From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM*, 42(1):31–37, January 1999.
- [186] Dina Neiger and Leonid Churilov. Goal-Oriented Business Process Modeling with EPCs and Value-Focused Thinking. In *Proceedings of the Second International Conference on Business*

- Process Management*, BPM 2004, pages 98–115, Potsdam, Germany, June 2004. Springer Berlin Heidelberg.
- [187] Janni Nielsen, Torkil Clemmensen, and Carsten Yssing. Getting Access to What Goes on in People’s Heads?: Reflections on the Think-aloud Technique. In *Proceedings of the Second Nordic Conference on Human-computer Interaction*, NordiCHI ’02, pages 101–110, Aarhus, Denmark, 2002. ACM.
- [188] Nan Niu and Steve Easterbrook. So, You Think You Know Others’ Goals? A Repertory Grid Study. *IEEE Software*, 24(2):53–61, March 2007.
- [189] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE ’00, pages 35–46. ACM, 2000.
- [190] Object Management Group. *BPMN version 2.0. OMG Final Adopted Specification*. Object Management Group, 2011.
- [191] Office of Kids and Families, NSW Department of Health, Australia. *Infants and children: Acute management of Head Injury*, 2 edition, March 2011.
- [192] Martyn A. Ould. *Business processes: modelling and analysis for re-engineering and improvement*. Wiley Chichester, March 1995.
- [193] Eray Ozkural and Cevdet Aykanat. A Space Optimization for FP-Growth. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, FIMI’04, Brighton, UK, November 2004. IEEE Computer Society.
- [194] Michael C Panis. Successful deployment of requirements traceability in a commercial engineering organization... really. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*, RE’10, pages 303–307, Sydney, Australia, October 2010. IEEE Computer Society.
- [195] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An Effective Hash-based Algorithm for Mining Association Rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’95, pages 175–186, San Jose, California, USA, 1995. ACM.
- [196] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. *Discovering Frequent Closed Itemsets for Association Rules*, pages 398–416. ICDT’99. Springer-Verlag London, UK, Jerusalem, Israel, January 1999.

- [197] Gabriele Paul. Approaches to abductive reasoning: an overview. *Artificial Intelligence Review*, 7(2):109–152, 1993.
- [198] Lauren Gibbone Paul. RosettaNet: Teaching business to work together. <http://www.developer.com/xml/article.php/616641/RosettaNet-Teaching-businesses-to-work-together.htm>, October 2011.
- [199] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *Proceedings of the 2000 ACM-SIGMOD International Workshop on Data Mining and Knowledge Discovery*, DMKD'00, pages 21–30, Dallas, TX, USA, May 2000.
- [200] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In *Proceedings of the 17th International Conference on Data Engineering*, ICDE 2001, pages 215–224, Heidelberg, Germany, April 2001. IEEE Computer Society.
- [201] José Luís Pereira and Diogo Silva. *Business Process Modeling Languages: A Comparative Framework*, pages 619–628. Springer International Publishing, 2016.
- [202] Joao Pimentel, Jaelson Castro, Hermano Perrelli, Emanuel Santos, and Xavier Franch. Towards Anticipating Requirements Changes through Studies of the Future. In *Proceedings of the Fifth IEEE International Conference on Research Challenges in Information Science*, RCIS 2011, pages 1–11, Gosier, Guadeloupe, French West Indies, May 2011. IEEE Computer Society.
- [203] Planview, Inc. Enterprise Architecture Software — EA Tools — Planview. <http://www.planview.com>, 2016.
- [204] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [205] Karthikeyan Ponnalagu, Aditya Ghose, Nanjangud C. Narendra, and Hoa Khanh Dam. Goal-Aligned Categorization of Instance Variants in Knowledge-Intensive Processes. In *Proceeding of the 13th International Conference on Business Process Management*, BPM 2015, Innsbruck, Austria, August 2015. Springer International Publishing.
- [206] Nicolas Prat. Goal formalization and classification for requirements engineering, fifteen years later. In *Proceedings of the IEEE 7th International Conference on Research Challenges in Information Science*, RCIS 2013, pages 1–12, Paris, France, May 2013. IEEE Computer Society.

- [207] Liang-Xi Qin, Ping Luo, and Zhong-Zhi Shi. *Efficiently Mining Frequent Itemsets with Compact FP-Tree*, pages 397–406. IIP2004. Springer US, Beijing, China, October 2004.
- [208] Balázs Rácz. nonordfp: An FP-growth variation without rebuilding the FP-tree. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, FIMI'04, Brighton, UK, November 2004. IEEE Computer Society.
- [209] Suzanne Robertson and James Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.
- [210] Colette Rolland. *Goal Oriented Requirements Engineering*, pages 35–51. Springer US, Boston, MA, 2002.
- [211] Colette Rolland, Georges Grosz, and Régis Kla. Experience with Goal-Scenario Coupling in Requirements Engineering. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, RE'99, pages 74–81, Limerick, Ireland, June 1999. IEEE Computer Society.
- [212] Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding Goal Modeling Using Scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, December 1998.
- [213] RosettaNet. RosettaNet Implementation Framework: Core Specification. <http://xml.coverpages.org/RNIF-Spec020000.pdf>, July 2001.
- [214] Douglas T. Ross and Kenneth E. Schoman. Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering*, 3(1):6–15, January 1977.
- [215] Anne Rozinat and Wil MP van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
- [216] Gordon Rugg and Peter McGeorge. Laddering. *Expert Systems*, 12(4):339–346, November 1995.
- [217] Kurt Sandkuhl, Janis Stirna, Anne Persson, and Matthias Wißotzki. *Enterprise Modeling: Tackling Business Challenges with the 4EM Method*. Springer-Verlag Berlin Heidelberg, 2014.
- [218] Metta Santiputri, Aditya K Ghose, Hoa Khanh Dam, and Xiong Wen. Mining Process Task Post-Conditions. In *International Conference on Conceptual Modeling*, pages 514–527. Springer, 2015.

- [219] Ayu Saraswati, Chee-Fon Chang, Aditya Ghose, and Hoa Khanh Dam. Learning Relationships Between the Business Layer and the Application Layer in ArchiMate Models. In *Proceedings of the 34th International Conference on Conceptual Modeling*, ER 2015, pages 499–513, Stockholm, Sweden, October 2015. Springer International Publishing.
- [220] Lionel Savary and Karine Zeitouni. *Indexed Bit Map (IBM) for Mining Frequent Sequences*, pages 659–666. PKDD 2005. Springer Berlin Heidelberg, Porto, Portugal, October 2005.
- [221] August-Wilhelm Scheer and Markus Nüttgens. *ARIS Architecture and Reference Models for Business Process Management*, pages 376–389. Springer Berlin Heidelberg, 2000.
- [222] Masakazu Seno and George Karypis. SLPMiner: An Algorithm for Finding Frequent Sequential Patterns Using Length-Decreasing Support Constraint. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM'02, pages 418–425, Maebashi City, Japan, December 2002. IEEE Computer Society.
- [223] Pradeep Shenoy, Jayant R. Haritsa, S. Sudarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. Turbo-charging vertical mining of large databases. *ACM SIGMOD Record*, 29(2):22–33, May 2000.
- [224] Sebastian Siegl, Kai-Steffen Hielscher, and Reinhard German. Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*, pages 345–350, Sydney, Australia, October 2010. IEEE Computer Society.
- [225] Alberto Siena, Anna Perini, Angelo Susi, and John Mylopoulos. A Meta-Model for Modelling Law-Compliant Requirements. In *Proceedings of the Second International Workshop on Requirements Engineering and Law*, RELAW 2009, in conjunction with RE'09, pages 45–51, Atlanta, Georgia, USA, September 2009. IEEE Computer Society.
- [226] Fabrizio Smith, Michele Missikoff, and Maurizio Proietti. Ontology-Based Querying of Composite Services. In *Business System Management and Engineering*, pages 159–180. Springer Berlin Heidelberg, 2012.
- [227] Fabrizio Smith and Maurizio Proietti. Rule-based Behavioral Reasoning on Semantic Business Processes. In *Proceeding of the 5th International Conference on Agents and Artificial Intelligence*, ICAART 2013, Barcelona, Spain, February 2013.

-
- [228] Software AG. ARIS Architect and Designer — Software AG. https://www.softwareag.com/uk/products/aris_alfabet/bpa/products/architect_design/overview/default.asp, 2016.
- [229] Ian Sommerville. *Software Engineering*. Addison Wesley, USA, 2015.
- [230] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.
- [231] Sparx Systems Pty. Ltd. Enterprise Architect - UML Design Tools and UML CASE tools for software development. <http://www.sparxsystems.com/products/ea/>, 2016.
- [232] Ramakrishnan Srikant and Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology, EDBT'96*, pages 3–17, London, UK, UK, 1996. Springer-Verlag.
- [233] Yudho Giri Sucahyo and Raj P. Gopalan. CT-ITL: Efficient Frequent Item Set Mining Using a Compressed Prefix Tree with Pattern Growth. In *Proceedings of the 14th Australasian Database Conference - Volume 17, ADC'03*, pages 95–104, Adelaide, Australia, 2003. Australian Computer Society, Inc.
- [234] Yudho Giri Sucahyo and Raj P. Gopalan. CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI'04*, Brighton, UK, November 2004. IEEE Computer Society.
- [235] Sagar Sunkle, Vinay Kulkarni, and Suman Roychoudhury. Intentional Modeling for Problem Solving in Enterprise Architecture. In *Proceedings of the 15th International Conference on Enterprise Information Systems, Volume 3, ICEIS 2013*, pages 267–274, Angers, France, July 2013.
- [236] Mustapha Tawbi, Camille Ben Achour, and Fernando Velez. Guiding the process of requirements elicitation through scenario analysis: Results of an empirical study. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications*, pages 345–349, Florence, Italy, September 1999.
- [237] The Object Management Group (OMG). UML Profile for Enterprise Distributed Object Computing (EDOC). <http://www.omg.org/spec/EDOC/>, February 2004.

-
- [238] The Object Management Group (OMG). Unified Modeling Language: Infrastructure version 2.0. <http://www.omg.org/spec/UML/2.0/>, March 2006.
- [239] The Object Management Group (OMG). Business Process Definition MetaModel Volume I: Common Infrastructure. <https://docs.oasis-open.org/wsbpel/2.0/PR02/wsbpel-specification-draft-diff.pdf>, November 2008.
- [240] The Open Group. ArchiMate 3.0 Specification. <http://pubs.opengroup.org/architecture/archimate3-doc/toc.html>, 2016.
- [241] The Process Mining Group, Eindhoven Technical University. ProM Tools. <http://www.promtools.org/doku.php>, 2010.
- [242] The Workflow Management Coalition. The Workflow Management Coalition Specification-Terminology & Glossary. Technical Report WFMC-TC-1011, The Workflow Management Coalition, 1999.
- [243] Oliver Thomas and Michael Fellmann. Semantic EPC: Enhancing Process Modeling Using Ontology Languages. *SBPM*, 251, 2007.
- [244] Pedro Valderas, Vicente Pelecha, Oscar Pastor, et al. Requirements Engineering for Pervasive Systems. A Transformational Approach. In *Proceedings of the 21st IEEE International Requirements Engineering Conference, RE'13*, pages 351–352, Rio de Janeiro, RJ, Brazil, July 2013. IEEE Computer Society.
- [245] Wil Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011.
- [246] Wil M. P. van der Aalst. Verification of Workflow Nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets, ICATPN'97*, pages 407–426, Toulouse, France, June 1997. Springer Berlin Heidelberg.
- [247] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 08(01):21–66, 1998.
- [248] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business Process Management: A Survey. In *Proceedings of the International Conference on Business Process Management, BPM 2003*, pages 1–12, Eindhoven, The Netherlands, June 2003. Springer Berlin Heidelberg.

- [249] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
- [250] Wil MP van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.
- [251] Boudewijn van Dongen and Wil M. P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In *Proceedings of the 23rd International Conference on Conceptual Modeling, ER 2004*, pages 362–376, Shanghai, China, November 2004. Springer Berlin Heidelberg.
- [252] Axel Van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *Proceedings of the 2000 International Conference on Software Engineering, ICSE 2000*, pages 5–19, Limerick, Ireland, June 2000. ACM.
- [253] Axel Van Lamsweerde. Goal-oriented Requirements Engineering: A Guided Tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering, RE'01*, pages 249–262, Toronto, ON, Canada, August 2001. IEEE Computer Society.
- [254] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. In *Proceedings of the 12th IEEE International Requirements Engineering Conference*, pages 4–7, Kyoto, Japan, September 2004.
- [255] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, August 1998.
- [256] Axel van Lamsweerde, Robert Darimont, and Philippe Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering, RE'95*, pages 194–203, York, UK, March 1995. IEEE Computer Society Press.
- [257] Axel van Lamsweerde and Emmanuel Letier. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering. In *Proceedings of the 9th International Workshop on Radical Innovations of Software and Systems Engineering in the Future, RISSEF 2002*, pages 325–340, Venice, Italy, October 2002. Springer Berlin Heidelberg.
- [258] Axel Van Lamsweerde and Laurent Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. *IEEE Transactions on Software Engineering*, 24(12):1089–1114, December 1998.

- [259] Philippe Fournier Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S Tseng. SPMF: A Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research*, 15:3389–3393, 2014.
- [260] Visual Paradigm International. TOGAF Architecture Development Method (ADM) Tools. <http://www.visual-paradigm.com/features/togaf-adm-tools/>, 2016.
- [261] W3C. Web Services Choreography Description Language Version 1.0. <http://xml.coverpages.org/RNIF-Spec020000.pdf>, November 2005.
- [262] Jianyong Wang and Jiawei Han. BIDE: Efficient Mining of Frequent Closed Sequences. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004*, pages 79–90, Boston, Massachusetts, USA, March 2004. IEEE Computer Society.
- [263] Jianyong Wang, Jiawei Han, and Jian Pei. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03*, pages 236–245, Washington, D.C., 2003. ACM.
- [264] Yiqiao Wang, Sheila A. McIlraith, Yijun Yu, and John Mylopoulos. An Automated Approach to Monitoring and Diagnosing Requirements. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2007*, pages 293–302, Atlanta, Georgia, USA, 2007. ACM.
- [265] Ingo Weber, Jörg Hoffmann, and Jan Mendling. Semantic Business Process Validation. In *Proceedings of the 3rd International Workshop on Semantic Business Process Management*, volume 472 of *SBPM'08, in conjunction with ESWC 2008*, Tenerife, Spain, June 2008.
- [266] Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: on the verification of semantic business process models. *Distributed and Parallel Databases*, 27(3):271–343, 2010.
- [267] AJMM Weijters, Wil M.P. van der Aalst, and AK Alves De Medeiros. Process Mining with the HeuristicsMiner Algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.
- [268] Lijie Wen, Jianmin Wang, Wil M.P. van der Aalst, Biqing Huang, and Jianguang Sun. A novel approach for process mining based on event types. *Journal of Intelligent Information Systems*, 32(2):163–190, April 2009.

- [269] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [270] Wikibon. A Comprehensive List of Big Data Statistics. <http://wikibon.org/blog/big-data-statistics/>.
- [271] Marianne Winslett. Reasoning about action using a possible models approach. *Urbana*, 51:61801, 1988.
- [272] Dennis Wixon and Judith Ramey, editors. *Field Methods Casebook for Software Design*. John Wiley & Sons, Inc., New York, NY, USA, 1996.
- [273] Peter Y.H. Wong and Jeremy Gibbons. A Relative Timed Semantics for BPMN. In *Proceedings of 7th International Workshop on the Foundations of Coordination Languages and Software Architectures*, volume 229 of *ENTCS*, July 2008.
- [274] Jane Wood and Denise Silver. *Joint Application Development (2Nd Ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1995.
- [275] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining: Closed Sequential Patterns in Large Datasets. In *Proceedings of the 2003 SIAM International Conference on Data Mining, SDM'03*, pages 166–177, San Fransisco, California, USA, May 2003.
- [276] Lili Yang, Raj Prasanna, and Malcolm King. GDIA: Eliciting information requirements in emergency first response. *Requirements Engineering*, 20(4):345–362, 2015.
- [277] Eric Yu and John Mylopoulos. Why Goal-Oriented Requirements Engineering. In *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, REFSQ 1998*, pages 15–22, Pisa, Italy, June 1998.
- [278] Eric S. K. Yu and John Mylopoulos. An Actor Dependency Model of Organizational Work: With Application to Business Process Reengineering. In *Proceedings of the Conference on Organizational Computing Systems, COCS '93*, pages 258–268, Milpitas, California, USA, 1993. ACM.
- [279] Eric S.K. Yu. Modeling Organizations for Information Systems Requirements Engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering, RE'93*, pages 34–41, San Diego, CA, USA, January 1993.

- [280] Eric S.K. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Toronto, Canada, 1995.
- [281] Eric S.K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE'97*, pages 226–235, Washington, DC, USA, 1997. IEEE Computer Society.
- [282] Eric S.K. Yu and John Mylopoulos. Understanding “Why” in Software Process Modelling, Analysis, and Design. In *Proceedings of the 16th International Conference on Software Engineering, ICSE '94*, pages 159–168, Sorrento, Italy, 1994. IEEE Computer Society Press.
- [283] Eric S.K. Yu and John Mylopoulos. Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering. *Intelligent Systems in Accounting, Finance & Management*, 5(1):1–13, 1996.
- [284] Yijun Yu, Yiqiao Wang, John Mylopoulos, Sotirios Liaskos, Alexei Lapouchnian, and Julio Cesar Sampaio do Prado Leite. Reverse Engineering Goal Models from Legacy Code. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering, RE'05*, pages 363–372, Paris, France, August 2005. IEEE Computer Society.
- [285] John Yunker, David Webber, Dale Moberg, Kenji Nagahashi, Stephen Green, Sacha Schlegel, and Monica J. Martin. ebXML Business Process Specification Schema Technical Specification v2.0.4. <http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en-html/ebxmlbp-v2.0.4-Spec-os-en.htm>, December 2006.
- [286] John A Zachman. Enterprise architecture: The issue of the century. *Database Programming and Design*, 10(3):44–53, 1997.
- [287] Osmar R. Zaïane and Mohammed El-Hajj. COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI'03*, Melbourne, Florida, USA, November 2003. IEEE Computer Society.
- [288] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May 2000.
- [289] Mohammed J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the Ninth International Conference on Information and Knowledge Management, CIKM '00*, pages 422–429, McLean, Virginia, USA, 2000. ACM.

-
- [290] Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.
- [291] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 326–335, Washington, D.C., 2003. ACM.
- [292] H. Zang, Y. Xu, and Y. Li. Non-Redundant Sequential Association Rule Mining and Application in Recommender Systems. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 3, pages 292–295, Toronto, Canada, August 2010.
- [293] Jelena Zdravkovic, Eric-Oluf Svee, and Constantinos Giannoulis. Capturing consumer preferences as requirements for software product lines. *Requirements Engineering*, 20(1):71–90, 2015.