

2016 | Faculty of Business Economics

DOCTORAL DISSERTATION

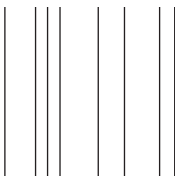
A flexible compliance auditing framework

Doctoral dissertation submitted to obtain the degree of
Doctor of Applied Economic Science, to be defended by

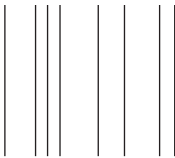
Nour Damer

Promoter: Prof. Dr Koen Vanhoof

Co-promoter: Prof. Dr Benoit Depaire



D/2016/2451/46



*"Having compliance policies but not enforcing them
maybe almost as harmful as not having them at all"*

- Anonymous

*I dedicate this thesis to
my family, my husband, and my beloved children
for their endless support and unconditional love*

Acknowledgements

A page or two are, of course, too short to acknowledge each person who has supported me during my PhD journey. I would like to express my sincere appreciation to everyone who, so generously, contributed to the work presented in this thesis. This project would not have been possible without your support.

Foremost, I would like to express my sincere gratitude to my promoter Prof. Dr. Koen Vanhoof. Thank you for supporting me and my research since the first day I started my life in Belgium. You have been a tremendous mentor for me. Your patience, motivation, and immense knowledge helped me all the time. You learned me how to deal with very complicated problems and simplify them as much as possible to be solved easily. You encouraged me to think more independently about my research.

I would also like to express my very great appreciation to my co-promoter Prof. Dr. Benoit Depaire. Your endless guidance inspired me in the most important moments of making right decisions and had significantly contributed to my research. Thank you for the fruitful discussions which used to end up with new research challenges. Thank you for trusting me and believing in me. I will never forget your opened door, smiley face, and willingness to give me time so generously.

I would also like to extend my thanks to Prof. Dr. Mieke Jans for the nice experience I had with you while doing my PhD. Your continuous support was always helpful. Besides, I would like to thank my PhD committee: Prof. Mario Cools, Dr. George Sammour and Dr. Maikel Leon Espinosa for their insightful comments and constructive suggestions. Without your feedback, this research would not have been reached this stage.

Profound gratitude goes to Naser Damer for his contribution in analyzing the clustering results and Hasan Al_Hamash for his support in developing the tool.

I wish to acknowledge Hasselt University and Princess Sumaya University for Technology for giving me the opportunity to do my PhD. The help provided by the

staff of the two universities was always supportive.

The PhD path is never straight forward, it is full of obstacles and hard moments. I could not handle these moments without the support of my wonderful friends. An, simply, no words can describe my gratitude thanks. Since we first met, I knew that you will be a great friend forever. Jo, thank you for the nice moments we spent together inside and outside Belgium. Working with you was a very nice experience for me. Qiong, thank you for all the fun we had together. Lotte, thank you for being such a great friend. Watching you while you are working at office was enough to encourage me to work harder. Asma, thank you for all forms of support whether it was directly or indirectly. I promise to do my best to achieve our shared dream. Samaneh, Aida and Reza thank you for being nice friends on whom I can count. I will never forget the many wonderful lunches and fun activities we had together. Dalia, thank you for the nights we spent together on skype discussing pure academic topics as well as funny kitchen voices. Thanks are extended to all of you my friends (too many to list here but you know who you are!). Special mention goes to Lieve, Luk, Feng, Katrien Declercq, Kristel, Marijke, Hanne, Katrien Ramaekers, Fatima, Ghada, Arwa, Sarah for providing support and friendship that I needed.

Amer, my soul-mate, I find it difficult to express my great appreciation because it is so boundless. Thank you very much deep from my heart. You have been a great supporter with unconditional love during my good and bad times. Although the last year was very difficult for both of us, you were always beside me to encourage me and provide any form of support you can provide. Literally, this work would not have been completed without your support! There are no words to convey how much I love you. You are my everything.

Mom, the beauty of my life, do not know what to say! Thank you very much for everything. Thank you for dedicating your life to make us happy, for providing unconditional love and care, for leading us towards our goals. Actually, no words can describe my gratitude thanks. You are an amazing mother and a unique person. Sister and brother, thank you for everything. Thank you for taking responsibilities which were not yours. Thank you for spending the nights awake with me just to encourage me. Hanin, you are not just a sister but a companion, a friend, a confidant for life. Thank you for taking care of Maria and preparing the suitable situation for me to work when there was a deadline. Sameer, my strong support system, thank you for hiding your very busy schedule just to let me ask you comfortably when I used to have a latex issue or a problem in the MATLAB code. My

great family, I am very proud that I am one of your members. You are a blessing of God.

My father in law, God bless your soul, I was hoping that you are alive now to share my wonderful feelings with you now. But it was not Allah fate. Thank you for the unforgettable smile which was enough to motivate me for decades. Your life was is a standalone school which is enough to guide any person to achieve his/her goal. My sisters in-laws, thank you very much for supporting me spiritually throughout writing this thesis. I used to have one family, now I have two. Laila and Ola thank you for your endless support. You are real sisters not in law. Laila, given the limited space I have here, I cannot list all the points that I want to thank you for. So, I would simplify it and say thank you for every happy moment you caused in my life since 2005 and till now. Ola, few months were enough to discover your great personality. You used to support me at the time that you yourself needs support more than me. I will never forget leaving your home for nights with two children just to take care of my daughter while I am abroad defending this thesis.

Last but of course not least, I would like to thank two persons who were the main reason behind starting and finishing this project: my hero, daddy, and my little angle, Maria.

This project is started 21 years ago after a short discussion with my daddy God bless his soul. I can still remember each word he said at that time: "Nour, my dream was to go to the university, but it was impossible at my time. I had to leave the school early to work and make money to live. My dream now is to see one of you, my children, a professor at the University. If you like the idea then make sure that I will support you endlessly!". Four years later, he passed away before I even continued my school study. Since that time, my career path has been drawn. Daddy, I can say now that your dream is achieved, thanks Allah. I wish you are here today to watch me putting the final touch on my PhD thesis, the project that you have started!

Maria, my sweet heart, I know you cannot read these lines for the moment but I will keep your copy aside until you are old enough to read. My daughter, thank you for being a part of my life. I had a very difficult time before your birth. It was difficult to the extent that I was asking myself why did I start and shall I continue? I almost lost any type of motivation to proceed. However, your birth has changed everything in my life. It was a turning point to remember the old Nour who was fully motivated, working days and nights. Your bright little smile was enough to give me a huge batch of enthusiasm to work more and more. I promise

you to spend more time together from now on playing, laughing, and waiting for our new family member. I wish you my children a happy life full of love, care, and achievement and I would end with my grand-father words: "we will support you endlessly!"

Every end is a new beginning

Nour Damer

20/6/2016

Contents

List of Figures	ix
List of Tables	ix
List of Abbreviations	xiii
Abstract	xv
1 Introduction	1
1.1 Categories of compliance requirements	2
1.2 Compliance checking approaches	3
1.3 Compliance checking, conformance checking and compliance auditing	5
1.4 Motivation	6
1.5 Research question	7
1.6 Scope of this research	8
1.7 Main contributions	8
1.8 Thesis structure	9
2 Literature Review	11
2.1 Compliance auditing	11
2.2 Fitness measurement	18
2.2.1 The Procurement Process	19
2.2.2 Alignment-based techniques	20
2.2.3 Replaying-based techniques	29
2.2.4 Constraint-based techniques	37
2.2.5 A comparison between existing fitness measure	37
2.3 Process modeling languages	40
2.4 Conclusion	44

3	Business Process Modeling for Compliance Auditing	47
3.1	Declarative process modeling languages	47
3.1.1	Formal Contract Logic (FCL)	50
3.1.2	Linear Temporal Logic (LTL)	53
3.2	A comparison between FCL and LTL	55
3.3	BRCA language	58
3.4	Conclusion	70
4	Flexible Compliance Auditing Framework	73
4.1	The flexible compliance auditing framework	74
4.2	Fitness measurement technique	76
4.3	Aggregation functions	80
4.3.1	Cell fitness metric	84
4.3.2	Case fitness metric	88
4.3.3	Rule fitness metric	90
4.3.4	Process fitness metric	91
4.4	Example	92
4.5	Conclusion	96
5	The Loan Approval Process: A Synthesized Data Set	97
5.1	Generating the log file and defining the business rules	97
5.2	Result analysis	101
5.2.1	The process level fitness degree	101
5.2.2	Rule level fitness degree	102
5.2.3	Case level fitness degree	105
5.2.4	Cell level fitness degree	108
5.3	Conclusion	110
6	The Procurement Process: A Real Life Data Set	113
6.1	Process level fitness degree	115
6.2	Rule level fitness degrees	118
6.3	Case level fitness degree	122
6.4	Cell level fitness degree	122
6.5	Conclusion	123
7	Clustering Based Compliance Checking Approach	125
7.1	Clustering in process mining	126
7.2	Clustering based compliance checking approach	127

7.2.1 Clustering the log file	128
7.2.2 Measuring compliance	129
7.2.3 Profiling the clusters	132
7.3 Results discussion	135
7.4 Conclusion	137
8 BRCA Tool	139
9 Conclusion and Future Work	145
Appendix: The Syntax of BRCA Language	149

List of Figures

2.1	The procurement process in BPMN	19
2.2	The procurement process in Petri net	30
3.1	An example for a log file	59
4.1	Flexible Compliance Auditing Framework	74
4.2	Applying aggregation to measure fitness degrees	79
4.3	OWA weights for special cases	85
5.1	Loan approval process model using the BPMN notations	98
6.1	The procurement process Petri net	117
6.2	The distribution of cases throughput time	120
7.1	The distribution of document types over the six clusters	133
7.2	The distribution of purchasing groups over the six clusters	134
7.3	The distribution of PO values over the six clusters	135
7.4	The distribution of PO creators over the six clusters	136
8.1	The main screen of the BRCA tool	140
8.2	List of predefined business rules	140
8.3	Tuning parameters according to the selected rule	141
8.4	Selecting the suitable aggregation operator	142
8.5	Assigning weights to WAM operator to measure the cases fitness degrees	143
8.6	Fitness matrix	144

List of Tables

2.1	The traces recorded in the log file and their frequencies	20
2.2	A comparison between the fitness metrics introduced in the literature	40
2.3	A comparison between imperative languages and declarative languages with respect to the predefined requirements	45
3.1	Most Frequent Compliance Requirements	57
3.2	Relational operators in BRCA language	61
3.3	Logical operators in BRCA language	62
4.1	Fitness matrix given a log file of C cases and a BRCA model of R rules	75
4.2	The business rules representing the compliance requirements	93
4.3	Originators and modification values required to check the fitness	94
4.4	Fitness matrix for scenario A	94
4.5	Fitness degrees at the case level for the scenario B	95
5.1	Attributes defined for the loan approval log file	100
5.2	The fitness degree at the process level	102
5.3	Rules fitness degrees: statistical information	103
5.4	Rules fitness degrees using the different aggregation operators	104
5.5	The case fitness degrees for some selected cases	107
5.6	The effect of the operator used at the cell level on the fitness degrees for their corresponding cases	110
6.1	The process fitness degree	116
6.2	The fitness degree of the business rules using the different aggrega- tion operators	119
7.1	The number of clusters generated at each run time to determine the best value for k	129

7.2	Fitness degrees of the generated clusters and the original log	130
7.3	The fitness degrees of each cluster against each of the nine rules	131
7.4	Fitness degrees of each cluster against rules 6, 7, 9 and the original log file	131
7.5	Summary of profiling results	136

List of Abbreviations

AM	Arithmetic Mean
BCL	Business Contract Language
BPMN	Business Process Modeling Notations
BRCA	Business Rules Compliance Auditing
CTD	Contrary To Duty
CTL	Computational Tree Logic
FCL	Formal Contract Language
HR	Human Resources
ILP	Integer Linear Program
LTL	Linear Temporal Logic
PCL	Process Compliance Language
PI	Process Instance
PO	Purchase Order
SOD	Segregation Of Duties
SOX	Sarbanes OXley Act
UML	Unified Modeling Language
WAM	Weighted Arithmetic Mean

Abstract

Compliance auditing techniques are of high importance for process managers. They provide quantitative metrics to measure to which extent the executed processes are in accordance with the process model. The result is used later as an input to evaluate the organizational performance. Existing techniques, provide one fitness degree regardless of the context in which the fitness degree is measured. Important factors such as the flexibility of execution and the different interpretation for a compliant process have been underestimated in the literature.

In this work we present a Flexible Compliance Auditing Framework to support the business world with more applicable metrics. The developed framework provides different semantics for a compliant process. The fitness metrics can be customized according to the process analyst needs. In addition, we introduce the so-called BRCA process modeling language to represent the compliance requirements. The developed framework is implemented and evaluated by means of both a synthesized data set and a real data set. Results show the importance of providing different semantics to measure the fitness degree.

Chapter 1

Introduction

Organizations operate by executing a set of prescribed processes to achieve their goals. Once a new organization is established, the set of required processes are defined. The processes are defined according to the need of the organization. For instance, the loan approval process is defined in a lending organization such as banks. However, it is not defined for a new hospital. Some processes are common in most of the organizations such as: the procurement, recruitment and sales processes.

When a new business process is designed, a set of constraints are defined to control the execution of this process so we can achieve the organizational goals. These constraints, referred to as *compliance requirements*, include the set of guidelines, norms, laws, regulations, recommendations or qualities [1]. This set is stemmed from internal sources, such as organizational policies, and/or external sources, such as government legislation [2]. The term *process compliance* is used to describe the alignment of business processes execution against compliance requirements. Hence, *compliance checking* can be defined as the procedure of ensuring that business processes, operations and practice are in accordance with the prescribed requirements [3].

Process compliance is gaining an increasingly importance in the business world especially after the unexpected disastrous scandals such as Enron and Worldcom (USA), Parmalat (Italy), HIH (Australia) and Tyco International (France), to name a few [4] . For organizations, this is considered as a two-edged sword. On the one hand, compliance ensures the quality of their products and services and helps them controlling their operations. On the other hand, it forms a kind of burden since failing to comply is no longer an option [5, 6].

Organizations are concerned about being non-compliant because its associated with risk such as financial penalties, scandals and loss of business reputation [7]. In its 2015 Sarbanes-Oxley Compliance Survey [8], Protiviti indicates that 58% of large organizations spent \$ million or more on Sarbanes-Oxley Act (SOX) compliance costs in the most recent fiscal year and 25% of them spent more than \$2 million. These percentages are after excluding the external audit fees. 54% of the organizations have reported an increase of more than 16% in the total amount of hours devoted to SOX compliance. 78% of the organizations has indicated that the Internal Control over Financial Reporting (ICFR) has improved since SOX 404 has been required.

Process compliance is the concern of all parties involved in the business operation. Governments, process managers, stakeholders, internal and external auditors as well as clients are all interested in compliant processes. Governments aim at increasing the transparency of business operations and expanding the accountability of responsibilities [9]. For process managers, compliance is of high importance to protect their business from failure. It helps them obtaining quality certificates and outperform their peers as a competitive advantage [9]. Auditors share some objectives with process managers with more emphasize on internal controlling [2]. As for clients, executing compliant processes helps them ensuring the quality of the products and services they buy. The final goal is to protect the stakeholders from fraud corruption or corporate misconduct and to safeguard the economic system [9, 10].

1.1 Categories of compliance requirements

Compliance requirements are categorized into four main aspects: sequencing, temporal, human resources (HR), and data objects [10–14].

- Sequencing (also called control flow) requirements focus on the order in which activities are executed within the business process. Taking this aspect into account, we say that the process is compliant when its sequence of activities complies with the required sequence. Usually procedural modeling languages are used to model the order of execution such as Business Process Modeling Notations (BPMN) [3, 15], Unified Modeling Language (UML) Activity Diagrams [16] and Event-driven Process Chains [17]. Control flow requirements show the dependency between activities such as:

- Activity A should be followed immediately by activity B
 - Activity A should be executed before activity B is executed
 - Activities A and B are executed in parallel
- The temporal aspect is concerned with temporal constraints such as deadlines, service level agreements, activity duration, and upper and lower bounds for time intervals between activities [18]. This aspect is concerned with answering questions about when activities should(not) occur within a business process. Examples are:
 - Activity A should be executed before executing activity B by 10 days
 - Each process instance should be executed within 30 working days
- Human resources requirements are focused on the social network behind executing the business process, i.e. the people interacting with the process, and aim to answer the question: who will execute what? A famous example of this type is the segregation of duties requirement.
- The data objects aspect is concerned with the data used and produced when executing the business processes. Data objects flow between the process activities which can modify their values. This includes both the syntax and semantics of the data. For instance, in the procurement process the requirement: Activity C should be executed if the PO value is greater than 20,000 is a data object requirement. In addition requirements related to recording, viewing and accessing data are also included in this category. Examples are: how long should the recorded data be kept? what type of information should be included in the data object? who can access this data? etc. [19–21].

In addition to the above mentioned categories, there is a compound one. A compound requirement is the one in which more than one type of requirements are involved. For instance, "activity A should be followed by activity B within 5 working days" is an example of a compound requirement of two types: sequencing and temporal. Another example is: activity A should be executed by person P if the value of data(x) is greater than 100.

1.2 Compliance checking approaches

The compliance of a business process can be checked at one of three phases of the business process lifecycle: design-time, run-time and after execution [14].

Compliance checking during the design time aims at checking whether the process design is in accordance with the predefined requirements. The output of this phase is a process model holding the compliance requirements. Process models are expressed by means of a process modeling language such as Petri net, BPMN, Linear Temporal Logic (LTL) business rules, etc. Design time compliance checking is important to check the compliance at the next two phases i.e. run time and after execution, because its output is used as input there. Actually, if the process is not designed well to capture the compliance requirements then no point of auditing the process execution.

Once the process is designed, it can be executed. The execution of each process instance is monitored to ensure that it is being executed according to the process design model. This procedure is referred to as *run time compliance checking* as it takes place during the run time. Compliance checking at run time can be seen as a preventative technique. We say that a process instance is compliant when its behavior matches the process model. Otherwise, the process manager needs to react to correct the situation and conform with the process design again (if possible). However, in some cases, the deviation between the two sides reveals a problem in the design model which needs to be resolved before executing new process instances. This is the focus of process re-engineering field which is out of the scope of this research.

After execution, compliance checking techniques are used to detect deviations of executed processes from the process design. In the literature, compliance checking after the fact is called *compliance auditing* [10]. Compliance auditing, which is the focus of this research, is the procedure of checking that executed processes are in accordance with the process design. In general, compliance auditing techniques are based on comparing two parties: the desired behavior and the actual behavior. The desired behavior is what should have done. Normally, the process design model is the used to represent the desired behavior. On the other hand, the actual behavior is what was really done. This is the information recorded about the executed process instances. In automated systems this information is recorded automatically in a log file. The comparison between the two parties is conducted with respect to the process design.

In the literature, different approaches are presented to check the process compliance. Before 2010, the lion share of compliance checking research has been devoted to design time and then run time approaches [14]. Just recently, after the emergence of the process mining field, compliance auditing becomes increas-

1.3. COMPLIANCE CHECKING, CONFORMANCE CHECKING AND COMPLIANCE AUDITING

ingly important. Most of the work presented in the literature are focus on process compliance at one of the phases of the process lifecycle. The rest allow checking the compliance at more than one phase such as [13, 22].

This research is devoted to compliance auditing. Design time and run time compliance checking are out of the scope of this research. Readers who are interested in design-time compliance checking may refer to [3, 11, 15, 16, 23–34]. Readers who are interested in run time compliance checking may refer to [2, 35–39].

1.3 Compliance checking, conformance checking and compliance auditing

The terms: compliance checking, conformance checking and compliance auditing are used in the literature interchangeably. In this section, we explain the term conformance checking and then describe the relation between the three terms [40]. Conformance checking is a subfield of process mining describing the procedure of checking the conformity between a set of executed process instances and a process model. The procedure is executed from two perspectives: the log and the model. Thus, conformance checking techniques can be used for compliance auditing as well as process discovery. If the conformance checking technique is used to measure the conformity between a set of executed processes and a process design then it is compliance auditing. That being the case, the conformance is checked with respect to the model, i.e. it is assumed that the process model is correct and that the recorded cases should have been executed according to that model. In contrast, if the conformance checking technique is used to measure the conformity between the mined model and the log file used to generate it then it supports process discovery techniques. In the latter case, the conformance is checked with respect to the log, i.e. the log is assumed to be correct. The result is used to evaluate to which extent the mined model could represent the actual behavior. The usage of conformance checking for process discovery purposes is out of the scope of this research.

Compliance auditing is a special case of conformance checking in which the process model used is the process design model. However, compliance auditing is also a special case of compliance checking in which the compliance is checked after the fact. To conclude, the relation between the these three terms can be summarized as follows: compliance auditing is at the intersection between compliance checking and conformance checking.

1.4 Motivation

Compliance auditing techniques are of high importance for process managers. It can help them analyzing the performance of their executed processes. The result is a good indicator to evaluate the organizational performance.

Existing techniques provide quantitative metrics which measure to which extent the executed processes are in accordance with a process model. This is usually referred to as the fitness degree of the executed processes. Given the same input, i.e. the process model and the log file, existing metrics produce one fitness degree. However, in practice, some important factors should be considered when measuring the fitness degree such as: the process manager perspective, the ability to consider all requirements, and the flexibility of executing the process under investigation. Accordingly, the output of such metrics can be interpreted differently. Although existing techniques provide some metrics to evaluate the compliance degree. However, none of them consider such important factors when measuring the fitness degree.

Starting with the process manager perspective, we believe that each process manager has his/her own clue. What is considered as compliant from one manager perspective can be seen as non-compliant from another manager perspective. For instance, a process instance, from one perspective, is considered compliant if all compliance requirements are applied. Whereas, from another perspective, the same process instance is considered compliant if some of the requirements are applied. In simple words, existing techniques cannot be customized enough to meet the process managers needs and match their semantics of a compliant process.

Moreover, in the business world, some requirements are more important than others. For instance, in the procurement process, it is more important to have the purchasing order signed by the correct person than to check whether the process instance is initiated by a trigger. The cost of violating the first requirement is higher than that of the later requirement. Although this point is considered in the earliest work in the field of compliance auditing [41], it is discarded in the majority of the related work later on.

The flexibility of executing the process under investigation is yet another issue [42]. Business processes have different natures. Some processes should be executed very strictly. However, these are the minority. In reality, business tends to be more flexible [43]. Business men are interested in achieving their goals rather than the way it is reached. Actually, the aim of defining and executing the business processes is to achieve these goals. Most of the existing techniques do not

consider flexibility when checking the compliance. And even when it is considered, it is not treated well!

Flexibility and compliance are conflicting terms. Thus, it is not easy to handle both of them at the same time. Existing techniques in which flexibility is considered could handle the problem of representing more compliant ways to execute the process [44–47]. However, none of them consider flexibility when measuring the degree of compliance between the process model and the log file. We believe that the compliance degree between a log file and a process model should be calculated with respect to the flexibility of execution. One metric is not enough to measure the compliance of all types of business processes.

When compliance auditing, the executed processes are checked against the set of compliance requirements which are defined at the design time. As we mentioned earlier, a process design model is used to hold the compliance requirements. Thus, the ability of the design model to represent all types of compliance requirements are important. If the model cannot hold all pre-defined requirements, then the compliance checking result is not accurate enough to measure the compliance degree. Although this issue should be solved at the design time, it is not always the case. In addition, existing techniques lack the ability to check the compliance at different levels. This feature can help process managers getting insights and reveal common characteristics between non-compliant cases. So that, to avoid the problem in the future and improve the process.

The main contribution of this work is to provide a flexible compliance auditing framework which overcomes the previous drawbacks of existing techniques and that can be applied in the business context.

1.5 Research question

The main research question of this thesis is: "How should the existing research on compliance measures be extended to produce true business value when applied in a business context?". To answer this question, we define the following sub-questions:

1. What are the drawbacks of the existing compliance auditing techniques?
2. What are the metrics used in the literature to measure the compliance degree of executed process instances with respect to a normative model? How can we improve these metrics taking into consideration the flexibility issue?

3. What is the most suitable process modeling language that can be used to capture compliance requirements for the purpose of compliance auditing? Is there a need to develop one?
4. How can we apply clustering techniques to provide diagnostic information for compliance auditing?

1.6 Scope of this research

1. This research is focus on checking the compliance after the fact. Thus we assume that the process instances to be checked are completed. Process instances which are initiated but not completed are not considered here since they are still at the run time.
2. Design time compliance checking is out of the scope of this research. Hence, we assume that the process design is correct and that it is in accordance with the predefined compliance requirements. However, the capabilities of the process modeling languages in expressing the compliance requirements are taken into consideration.
3. The log file is assumed to be correct as well. Problems related to logging data such as incompleteness and incorrect recording are not considered.

1.7 Main contributions

1. A flexible compliance auditing framework which keeps a balance between flexible execution and compliant processes
2. A new fitness measurement approach based on aggregation. The approach provides different semantics taking into consideration the different interpretation for a compliant process. In addition it can be used to measure the fitness degree at different levels of abstraction
3. A new declarative process modeling language called BRCA used to hold all types of compliance requirements. The language is provided with a graphical user interface. Thus, it can be used easily by non-experts.
4. A new compliance auditing approach based on clustering which can be used to provide diagnostic information.

5. Introduce the concept of a "cannot be checked" compliance requirement to the compliance auditing field and show the importance of considering these cases as neither compliance nor non-compliant.

1.8 Thesis structure

The rest of the thesis is organized as follows. The literature review is discussed in Chapter 2. First the state of the art in the field of compliance auditing is discussed. Next, exiting fitness metrics are studied in details. Each metric is evaluated against the requirements defined in this research. Finally, the two main big families of process modeling languages are discussed and then compared to make a decision about which one is more suitable for compliance auditing.

Chapter 3 is focus on the process modeling language used to represent the compliance requirements. Declarative languages are discussed first. Later, the FCL and LTL languages are chosen as representatives for their logic families. The two languages are evaluated with respect to some predefined requirements. Accordingly, we introduce our new declarative process modeling language which we call the BRCA language.

The flexible compliance auditing framework is presented in Chapter 4. The chapter discusses the framework components and the interaction between them. In addition, it covers the fitness measurement technique used in the framework and the aggregation operators defined with their semantics.

To evaluate the proposed technique, two data logs are used: a synthesized log file and a real life data set. The results of the synthesized log file is presented in Chapter 5 and the real life case study results are presented in Chapter 6.

In Chapter 7 we introduce the clustering based approach for compliance auditing and show how clustering can be a good option to provide diagnostic information. The algorithm used to cluster the log file in addition to the steps defined in the proposed approach are explained.

The implemented tool called BRCA is described in Chapter 8.

Finally, the conclusion of this research in addition to some recommendations are presented in Chapter 9.

Chapter 2

Literature Review

This chapter is dedicated for the literature review. It is divided into three main sections. In section 1 the compliance auditing techniques presented in the literature are discussed. Section 2 is dedicated for fitness measurement. The fitness metrics introduced in the literature are studied in details and then evaluated with respect to some predefined factors. The process modeling languages are discussed in section 3. There we conduct a comparison between the two main families of modeling languages: procedural and declarative, to make a decision about which one should be used in this research.

2.1 Compliance auditing

In this section we discuss the state of the art in compliance auditing field and the related work presented in the literature.

The work of Cook and Wolf [41] was among the first to check the compliance between the behavior of a real-time system and the expected behavior. The idea is to compare an event stream derived from the design model with an event stream recorded in the execution log file. When running the method, the executed traces are transformed to match the expected event streams by means of inserting required events and/or deleting additional events. The number of inserted and deleted events needed to obtain a match to the model is used to measure the distance between both sides taking into consideration the length of the trace. The method provides the ability to assign weights in order to differentiate between the relative importance of different events. In [48] the work is extended to include time aspects where a timed concurrent model is used. This work is the base to

the alignment based techniques presented later. However, it has some limitations. First, the technique does not provide an overall fitness metric, i.e a fitness degree representing the fitness degree of the entire process. In addition, it cannot be used to check all types of requirements.

In the field of process mining, the work of van der Aalst [49] is one of earliest work for compliance auditing. The author identifies the Delta analysis approach to audit the compliance of a set of executed process instances. The analysis is based on comparing the design model with the actual behavioral model which is obtained through model discovery techniques such as *Heuristics Miner* [50] and *Genetic Miner* [51]. Delta analysis can be used when there is an actual behavior model but not a log file. For instance, when the log file used to generate the model does not exist anymore or when the process manager does not want to provide the data itself, therefore he compensates this by providing a behavioral model that can be used for the analysis. However, the technique does not provide a quantitative compliance measure. Moreover, the result of analysis depends on the actual behavioral model which is derived from the event log. Hence, the results could be misleading if the derived model does not represent the actual behavior appropriately. Notice here that the models extracted from the same dataset depends on discovery algorithm used and the tuning parameters defined. To overcome these issues, the authors extended their work in [52] to include the conformance testing approach. Conformance testing is based on comparing the recorded process instances directly with the design model. In addition, it provides a quantitative compliance measure and can locate the deviation. The concept of conformance testing is used in terms of security in [45]. The authors use the α algorithm to derive a perspective model from acceptable audit trials. The output of this step is a workflow net representing the sequence of activities within the process. The authors discusses the detection of anomalous process executions in the mined process model by means of the "token game". In a next step, the mined model is used as a reference to check the compliance of new audit trials. The approach considers the different security levels ranging from low-level intrusion detection to high-level fraud prevention. However, since it is based on replaying tokens, it suffers from the overestimation problem.

Conformance testing can be seen as an introduction to the conformance checking technique introduced by Rozinat et al. in 2008 [53]. Conformance checking is one of the earliest and most cited work on compliance auditing. The approach, which is based on replaying tokens, can be used to quantify the degree of confor-

mity between an event log and a Petri net representing the process model. The approach provides quantitative metric to measure the degree of conformity between an event log and a process model. Recorded cases are replayed on the Petri net. The fitness degree is defined to measure the conformity of the cases recorded in the log file with respect to a Petri net model. It is calculated by counting the number of missing and remaining tokens while the cases are replayed. We say that the log file and a process model completely fit (100%) if the model can generate all cases recorded in the log file. To measure the fitness of a log file, each recorded case is replayed in the model in a non-blocking way. This means that if there are missing tokens to fire a transition, this is detected as a location of non-compliant. In this case, missing tokens are created artificially to proceed running the replay algorithm. The approach has the advantage of locating deviations. However, since the technique is based on a replaying algorithm, the fitness degree tends to be overestimated. Moreover, it can be used to check all types of compliance requirements. This is because the normative process is represented by means of a basic Petri net model. Petri nets can represent sequencing requirements but they cannot represent the other three types of requirements.

An approach based on database technology is presented in [35] to automate compliance auditing with Sarbanes-Oxley internal controls [54]. It consists of four main components: workflow modeling, active enforcement, workflow auditing and anomaly detection. The authors provide two types of workflow auditing: compliance verification and query-based auditing. To verify the compliance, actual workflows are compared against the required workflow which represents the expected behavior. In the proposed approach, the required workflow holds the compliance requirements. Any deviation is detected as a control violation. This is similar to the Delta analysis approach proposed in [49]. As for query-based auditing, the auditor can express the violation as a query that can be checked for occurrence in the activity log. To detect anomalies the authors use a discovery-driven OLAP analysis.

In [7], a semi-automatic approach is presented to check the compliance of the so-called unmanaged processes. Unmanaged processes are processes which are executed with the absence of an execution engine. The approach is based on the business provenance technology introduced in [55] which aims to trace end-to-end business operations. The authors argue that a combination of both traditional auditing techniques and automatic auditing tools can reduce the level of non-compliant cases. It is not clear in the work how controls are represented.

In [56–58], the authors introduce a conformance checking approach based on trace alignment. The idea is closely related to the idea of Cook et al. [41, 48]. A trace is replayed in a Petri net model by inserting and skipping activities. Skipped activities are those which are not executed while the process model requires them to be executed. Inserted activities are those which appear during the execution time while the model does not allow them to be executed. The approach allows the user to define the severity of deviation for each activity individually. Afterwards, an instance of the Petri net that best matches the recorded trace is constructed. In this context, an instance is a connected part of the Petri net that starts at an initial place and does not contain choices. Later on, the A* algorithm [59] is used to find the one with the least cost. Once the best match instance is found, the number of inserted and skipped activities can be determined and used to measure the degree of conformity between the log file and the process model. This technique overcomes the problem of overestimation encountered in the token replaying based techniques [53]. However, it cannot be used to check all types of compliance requirements. In [60], the same authors introduce a similar alignment approach based on discarding the partial alignments that are “hopeless” so that to reduce the memory required while finding the optimal alignments. Thus, larger logs can be analyzed. As an extension to their previous work, the approach consider all modeling languages for which translation to Petri nets is available.

In 2012 [61], the authors propose their technique to complement the work in [57] by considering both data and resource requirements. The proposed technique is introduced to be independent of the process modeling language. Thus, a causal net is used to represent the process model and extended to include data and resource requirements [62]. An extended casual net is a graph where nodes represent activities and arcs represent causal dependencies.

Another alignment based approach is presented in [63]. However, unlike [61] which uses the A* with a heuristic function, this work is based on using an Integer Linear Program (ILP) to create an optimal alignment between the event log and the process model. The technique allows checking the different types of requirements but cannot be used to check the compliance of one specific requirement. This is because the process model, represented by means of a data aware Petri net, captures the behavior of the whole process as one unit.

In [64] the authors presented a decomposed data-aware conformance checking approach. The idea is to split the process model, which is represented by means of a Petri net with data [65], into smaller partly overlapping model fragments using a

divide and conquer algorithm. Next, a sub-log is created for each model fragment according to the activities used in that specific fragment. The ILP based approach proposed in [63] is used to check the conformance of each sub-log against its corresponding model fragment. The authors argue that this helps eliminating the computational time required to check the conformance.

A similar approach based on partial alignments is introduced in [66]. The authors argue that sequential traces are unable to describe concurrent events. Hence, they propose a technique in which partially ordered traces rather than totally ordered traces are used for alignment. The work is concerned with sequencing requirements rather than the other types.

The work of Ramezani et al. [67, 68] is one of the recent work in the field. The authors introduce a compliance checking approach based on Petri net patterns and trace alignments. In [68] the authors categorized a set of 55 control flow rules into 15 categories. Each rule is formalized as a Petri net pattern describing the desired behavior. The deviation between the log file and the Petri net patterns is measured using the cost-based fitness technique [56]. The work is extended in [67] to include temporal requirements. For this purpose, the authors introduce a technique to formalize temporal compliance requirements. Temporal rules are expressed by means of a data-aware Petri nets. The work distinguish between control flow and temporal compliance checking to the possible extent. However, it provides diagnostic information about both control-flow and temporal violations. Temporal requirements are checked using an alignment based compliance checking technique. Later on, the data and human resources requirements are also included [69].

In [70] the author introduces an approach for obtaining and visualizing diagnostics when checking the compliance of a business process. Compliance requirements are represented by means of Petri net attached with Data [71]. The author considers the four types of requirements. To detect violations, the author uses two of the existing compliance checking techniques: 1) the temporal compliance checking technique introduced in [72] is used to check sequencing and temporal requirements, 2) the technique presented in [73] is used to check data and HR requirements. Once the violations are detected, the cause of a specific problem (selected by the user) is determined using the C4.5 classification algorithm. The problem of this approach is that recorded cases are first classified and then the fitness degree is measured. However, this is not applicable if the number of attributes is relatively high. For instance, suppose that the process manager sus-

pects that attribute A could be the reason of violation and there is a thousand unique values for A then the log file should be classified into 1000 sub-logs. In addition, it assumes that the process manager has a hypothesis about the cause of the deviation.

Alizadeh et al. [74, 75] work on improving the cost function used to find the optimal alignment between a log file and a labeled Petri net [76]. Hence, only sequencing requirements are considered. In the previous work the cost function is defined manually according to the process analyst knowledge and experience. However, this work propose a cost function which is defined automatically according to some information extracted from compliant process instances. The cost function is computed based on the probability of activities to be executed or not in a certain state.

The authors of [77] presented a balanced multi-perspective approach. The work can be seen as an extension to [63]. However, in contrast to [63] where control flow is considered first and then the other types, this work allows for treating the different types equally. The approach in [63] may cause misleading results if control-flow and the other types are closely inter-related. Hence, the authors here provide a customized cost function to have equal contribution to the cost degree.

In [78] an approach based on behavioral profiles is introduced. The idea is to utilize behavioral profiles as a base line to calculate the compliance metric. The concept of behavioral profiles relates pairs of activities according to their sequence. Other types of compliance requirements are not considered. Behavioral profiles are identified for both: actual behavior (log) and desired behavior (process model). The degree of behavioral compliance is defined as a ratio of consistent behavioral relations relative to the number of activity pairings in the log.

In [79] the authors define compliance checking as the degree to which the behavioral process model is in accordance to a reference model. The work identifies five models to be considered for compliance checking: the meta reference model, adopted reference model, to-be process model, the instances of a process model and the as-is process model. The last one is the actual behavioral model that is derived from the event log. The behavioral model is generated by means of a process discovery algorithm [62]. The authors argue that process models can have different compliant structures. Thus, the authors introduce the so-called sequence based compliance algorithm to measure the compliance degree of the last two models (the instances of a process model and the as-is process model)

with respect to the adopted reference model.

Most of the compliance checking techniques introduced in the literature consider a procedural process model. However, few work, focus on checking the compliance with respect to a declarative model [44, 46, 47, 80].

The LTL checker is a process mining technique proposed in 2005 for compliance checking [80]. Compliance requirements are expressed as Linear Temporal Logic (LTL) formulae. Given a log file and a set of LTL rules, the technique is used to check the compliance from two perspectives: the instances and the LTL rules. From the instances perspective, the recorded cases in the log file are divided into two lists. One for correct cases and one for the remaining cases. Correct cases are those in which all rules are applied. From the LTL rules perspective, the technique categorizes the set of LTL rules into two subsets. One for satisfied rules and one for unsatisfied rules. A rule is considered satisfied just in case it is applied in all recorded cases. The technique provides two simple metrics: the health degree (for cases) and the coverage (for rules). The health degree of one specific case is the percentage of applied rules to the total number of rules. The coverage degree of one specific rule is the percentage of correct cases to the total number of cases checked. LTL checker has the advantage of representing the process design by means of a declarative language. Thus, the process can be executed in a flexible way. The LTL logic allows the process analyst to represent the different types of requirements. However, it assumes that all compliance requirements are of the same importance. In addition, the technique does not provide a metric to measure the compliance degree of the entire log file. Moreover, using the LTL checker, the result is either 0 or 1 no matter how many times the rule is checked.

In [44], the authors develop a declarative language entitled with SCIFF and use it to specify the business contracts. SCIFF is a declarative language based on abductive logic programming. A contract in SCIFF is specified by means of two components: a declarative knowledge base and a set of rules. The rules are used to generate expectations about the behavior of the parties involved in the contract. The events are inserted in the knowledge base as facts that happened in the past (history). We say that we are compliant with respect to the contract when the facts are in accordance with the generated expectations. The approach supports the dynamic monitoring of compliance since new facts can be inserted into the knowledge base during the procedure of compliance checking. The work is mainly targeted at checking the compliance of executed processes with respect to a business contract. The work is similar to an earlier work presented by [81] where the

authors present an approach to check the compliance of careflow processes. First, the careflow process is represented in the so-called GOSpeL graphical language developed by the authors. Next, the careflow model is translated into SCIFF language to verify the compliance of the facts, i.e. executed processes, with respect to the model. The approach supports sequencing and temporal requirements but does not support the other types.

In [46], de Leoni et al. introduce a conformance checking approach for declarative models. The presented approach is applied to Declare, a declarative process modeling language presented in [42]. Similar to Adriansyah work [56], this work is an alignment based approach. However, the authors adapted the approach to be able to deal with large search space required in case of declarative models. The work is extended in [47] in which a framework for log preprocessing and conformance checking is introduced. The A* algorithm is used to find the optimal alignment for each trace. Three use cases are considered in the work including the conformance checking use case (the second one). The other two are cleaning the log from traces that should not be used for further analysis and repairing the event log to make sure essential constraints are satisfied before further analysis is conducted.

Recently, a constraint-based approach is presented in [82] to check the conformance of declarative models. Declarative models are represented as constraint satisfaction problems. The process instances are classified into two sets: compliant and non-compliant. Later, a model-based fault diagnosis process is applied for each non-compliant case to determine the problem.

2.2 Fitness measurement

Fitness metrics are defined to quantify the degree of conformity between a process model and the executed processes. We categorize the existing fitness metrics into three groups: alignment-based techniques, replaying-based techniques and constraint-based techniques.

This section is devoted to study the fitness metrics introduced in the literature. First, we start with a simple business process which will be used as an example through out the section. Next, we study each metric separately and apply it to the example if possible. Please notice that this section is devoted to study all metrics introduced in the literature to measure the fitness degree between a log file and a process model from the model perspective. At the end of the section, we conduct

a comparison between studied metrics with respect to some predefined factors.

2.2.1 The procurement process: an example

To explain the idea of the fitness metrics discussed in this section, we will use a simple procurement process as an example throughout the chapter. The procurement process is selected for two reasons. Firstly, it is one of the most common processes in the business world. Organizations need to purchase goods and/or services regardless of their functionalities. Secondly, we think it would be a good introduction for the real life case study used in this research (See Chapter 7).

The procurement process is concerned with the purchase orders created in an organization and how the procedure takes place since the purchase order is created until the ordered products are received and the invoice is paid. In what follows, we describe a simple procurement process and provide a corresponding log file.

The process is initiated when a new purchase order (PO) is created. Next, the order is signed by the direct manager of the PO creator and then sent to the supplier. Once the invoice and the ordered products are received, the payment activity is executed and the process is terminated. The payment should take place within 30 days after the ordered products and the invoice are received. The sequence of activities performed to execute the process is shown in Figure 2.1. The model is represented by means of BPMN languages. According to the model, receiving goods and invoice activities occur concurrently, i.e. in parallel with no preference. The activity 'Receive goods' can be skipped in case the ordered product is a service.

A log file corresponding to this model is shown in table 2.1. This log file is not a real case but an example that will be used together with the process model throughout the section. The traces can be grouped into six unique sequences. The majority (90%) of the traces are in accordance with the sequencing requirements represented in the BPMN model. Notice that the model represents the sequencing requirements only. However, it can be annotated to represent the other two requirements, the temporal (payment within 30 days) and the HR (by the direct manager) requirements. The modeling language is discussed later in chapter 3

The fourth sequence cover the pattern of missing a required activity, 'Sign' in this case. Conversely, sequence 6 covers the pattern of executing an additional activity, 'GR' in this case. Although the additional activity is one of the model activities, its order in the log does not match that in the model. Trace 5 represents two additional activities as well. However, one of them does not appear in the model, i.e. 'Change order'. In all of the 38 cases, the new activity is followed by

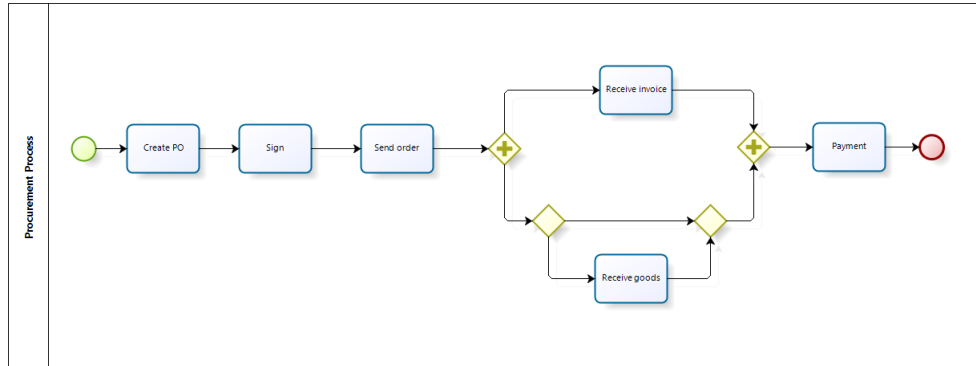


Figure 2.1: The procurement process in BPMN

the 'Sign' activity. Accordingly, we can assume that the order can be changed after it is sent to the supplier. In addition, if it is changed then it needs to be signed again. Although this does not match the model, it could be possible. Nevertheless, these are all assumptions that needs more investigation.

Table 2.1: The traces recorded in the log file and their frequencies

	Sequence of activities	Frequency
1	Create PO, Sign, Send order, GR, IR, Pay	374
2	Create PO, Sign, Send order, IR, GR, Pay	290
3	Create PO, Sign, Send order, IR, Pay	236
4	Create PO, Send order, GR, IR, Pay	53
5	Create PO, Sign, Send order, Change order, Sign, GR, IR, Pay	38
6	Create PO, Sign, Send order, GR, IR, GR, Pay	9
	Total	1000

2.2.2 Alignment-based techniques

These are the techniques based on aligning the recorded case on the process model. There are mainly two techniques under in this group: the process validation and the cost-based compliance checking

Process validation

Process validation is one of the earliest techniques (1999) presented by Cook et al. [41]. The approach is based on comparing two paths, the actual execution

path and the desired execution path. The actual execution path represents the sequence of events for the process instance under investigation. On the other side, the desired execution path, also referred to as the expected execution path, is the path that holds the compliance requirements. This is derived from the designed process model and it represents the expected behavior that we need to comply with. Usually, the designed model represents different expected behaviors. Each of which is represented by one execution path starting at an initial state, ending at a final state, and contains a set of connected activities in between. However, when a new process instance is initiated, only one of these paths is followed.

Later on, when applying the process validation technique, there are only two results, either the executed path completely matches one of the expected paths, which is the best, or that it does not match any of them. In the latter case, it could be closer to one or more of the expected paths. That being the case, the technique consider the different paths and select the lowest cost alternative. For instance, if the executed path is $\langle A, B, C, D, E' \rangle$ and there are two expected execution paths: $\langle A, B, C, D, E, F \rangle$ and $\langle A, B, D, E \rangle$ then which expected path should be considered as a desired path? The executed path can be easily converted to the first path $\langle A, B, C, D, E, F \rangle$ by inserting an activity 'F' after 'E'. Also, it can be converted to the other path $\langle A, B, D, E \rangle$ by simply deleting the activity 'C'. Hence, the technique compares the cost of inserting 'F' with the cost of deleting 'C' and then select the least cost alternative.

The comparison between the two paths is conducted by means of model checking techniques. The executed path is checked against the desired path. As long as the executed path matches the desired path, the process is a straightforward process. When a mismatch is encountered, the necessary events are inserted/deleted to transform the executed path into the expected path. For instance, if the executed path is: $\langle A, B, C, C, D, F \rangle$ and the expected path is: $\langle A, B, C, D, E, F \rangle$, then the second activity C is deleted first. Next, an activity E is inserted after the activity D so that both sides are identical.

The determination of the least cost path is done by means of a best-first search method. The cost of a specific path is the total cost of the insertion and deletion operations required to convert the executed path into one of the expected paths. The costs of inserting and deleting the different activities are assigned by the process analyst. A matching case is considered to cost zero units. Once a mismatch is encountered, the current lowest cost partial solution is evaluated and all its possible continuations are generated. This generates a set of possible solutions,

each of which is assigned a cost according to the insertion and deletion operations needed to obtain this specific solution. This is repeated at each step during the model checking procedure. The generated solutions are placed into a priority queue and the next step is performed. Finally, the lowest-cost solution is selected as the best one.

To determine the correspondence between the two paths, the authors consider each path as a string where events are the characters that form this specific path. Consequently, they could use string distance based metrics to measure the difference between the two paths. Two metrics are introduced to measure this distance: *Simple String Distance Metric (SSD)* and *Non-linear String Distance Metric (NSD)*.

To apply the *SSD* principle, the authors define the *SSD* metric to be the minimum number of inserted and deleted events that are needed to convert the executed path into the expected execution path if there is a mismatch. In this case, inserted events are those which are required to be executed but they are missed in the executed path. On the other hand, deleted events are events which should not occur according to the designed model, however, they appear in the actual execution. The approach penalized inserting and deleting operations by some costs. The insertion cost is interpreted as the cost of missing the execution of some activities. Whereas deletion costs can be interpreted as extra work that was performed. *SSD* is formulated as:

$$SSD = \frac{W_I N_I + W_D N_D}{W_{max} L_E} \quad (2.1)$$

where:

- W_I and W_D : the costs (weights) of insertion and deletion operations respectively.
- N_I and N_D : the number of insertions and deletions (respectively) that are needed to convert the executed path into the expected path.
- W_{max} : maximum of W_I and W_D .
- L_E : the length of the executed path.

For instance, let's find the *SSD* value for the fifth sequence presented in our example (See section 2.2.1) assuming that $W_I = W_D = 1$. The trace has a problem that can be solved by deleting two events 'Change order' and its following event 'Sign'. This will convert it to match its closest expected trace: ⟨'Create PO', 'Sign', 'Send order', 'GR', 'IR', 'Pay'⟩. In this case the *SSD* value will be:

$$SSD = \frac{1 * 0 + 1 * 2}{1 * 8} = 0.25 \quad (2.2)$$

SSD value indicates the distance between the two paths. Given that the value is bounded between $[0,1]$, a path with zero SSD value completely fits the designed model. Notice here that the first three traces in our example have zero SSD values because each of which completely fits one of the three expected paths presented in the model. The authors use a standard statistical correlation rules of thumb to set a cutoff value. Accordingly, if $SSD < 0.2$ then the executed path is strongly corresponding to the expected path. Else, if $SSD < 0.5$ then it is moderate corresponding. Otherwise, it is a weak correspondence. Given that the SSD value for trace 5 is 0.25 we can say that the trace is in the moderate correspondence range and that it is close to the strongly corresponding range.

The SSD metric considers the cost of each insertion and deletion operation separately. However, the authors argue that a sequence of missed activities can be seen as a single deviation from the designed model. So that, they introduce the NSD metric as an enhancement to the SSD. Under NSD, the unbroken series of insertion or deletion operations, referred to as blocks, are considered as one operation whereas the block lengths is used to calculate the NSD value.

$$NSD = \frac{\sum_{j=1}^{N_I^B} W_I f(b_j) + \sum_{k=1}^{N_D^B} W_D f(b_k)}{W_{max} L_E} \quad (2.3)$$

where the new terms over SSD are:

- N_I^B and N_D^B : the numbers of inserted and deleted blocks respectively.
- b : a specific block length.
- $f(b)$: a cost function applied to a block length

The cost function is defined as an exponential function:

$$f(b) = e^{k(b-1)} \quad (2.4)$$

where k is a constant value used in the function as a tuning parameter. For a block of length 1 the value would be 1.0. So, in case all blocks are of length 1, NSD value will yield to the SSD as expected. The value of k is considered to be ($k > 0.7$) so that NSD value is always greater than SSD. Practical experiments show that a value between $[1,3]$ is more suitable to use.

For comparison, let's find out the NSD value for trace 5 in our example assuming that $k = 1.5$. Notice that in SSD, the two additional events are handled separately by two deleting operations. However, under the NSD they are considered as one

block of length 2 that can be handled by one deleting operation. In this case the cost function will be:

$$f(b) = e^{1.5(2-1)} = 4.5 \quad (2.5)$$

So, NSD value will be:

$$NSD = \frac{0 + 1 * 4.5}{1 * 8} = 0.5625 \quad (2.6)$$

Unlike SSD, NSD is unbounded $[0, \infty)$. Therefore, it is not possible to apply the same evaluation criteria used before, i.e. 0.2 for good correspondence and 0.5 for moderate correspondence. To get over this, the authors define an equation to derive cutoff values from the existing values. Given k and the average block length (B_{avg}), the good correspondence cutoff is:

$$cutoff = \frac{0.2e^{k(B_{avg}-1)}}{B_{avg}} \quad (2.7)$$

Assuming that $B_{avg} = 2.5, k = 1.5$, the NSD good correspondence cutoff will be 0.76. This implies that trace 5 is a good correspondence trace. Notice that the same trace was categorized as a moderate correspondence trace under SSD. However it was close to the good correspondence border. This denotes that the approach penalizes deleting a block of length two less than deleting two events.

The work differentiates between insertion cost and deletion cost. However, in practice, the cost depends heavily on which activity will be inserted or deleted. For instance, the process manager may need to penalize an additional 'Payment' event more than an additional 'Sign'. The technique allows overriding the default insertion and deletion costs to match this requirement. The insertion or deletion of each activity can have a different cost according to its importance.

Final output includes the SSD and NSD measures, information about the number of inserted and deleted events, the number of insertion and deletion operations as well as the number of matches per event type. In addition, the deviation is located at both the executed path and the expected path. Nevertheless, only sequencing requirements are considered.

To include temporal requirements, the work is extended in [48]. A timed concurrent model is used to represent the business process. To check the temporal divergence, the difference between the time an event occurred and the time where it is allowed to occur according to the model design is computed. An event is considered divergent if it is executed outside the specified time interval. Hence, the

SSD_t metric equation becomes:

$$SSD_t = \left(\frac{W_i N_i + W_d N_d}{L_E} + \frac{W_t D_t}{T_E} \right) / W_{max} \quad (2.8)$$

where the first part is just the SSD metric discussed above and:

- D_t : the sum of absolute values of time discrepancies that are measured over the executed path.
- W_t : the cost (weight) of time.
- T_E : the total elapsed time of the complete path.
- W_{max} : now the maximum value of (W_i, W_d, W_t)

The interaction between matching the behavior and temporal discrepancies is handled separately for the three possible cases (matching, insertion and deletion). First the behavior is checked. In case of behavior matching, the metric is penalized just in case there is a temporal discrepancies. In case of insertion, the allowable time interval should be considered first. The last case, i.e. deletion, does not have any effect since it is not connected to the model anymore. The metric can be used to check both sequencing and temporal requirements. However, it cannot be applied when the model is not attached with the allowable time interval of each activity.

To the best of our knowledge, Cook et. al. work is the first to measure the compliance degree between an executed process and a model design. It allows assigning different weights to the activities according to their importance. However, it does not consider the data and human resources requirements. Moreover, it does not provide an overall measure for the entire log file. The latter is considered one of its main drawbacks.

Cost-based compliance checking technique

In 2011, Adriansyah et al. introduce an alignment based compliance checking technique using the idea of inserted and skipped activities [83]. The technique considers incomplete cases. Thus it can be used for run time compliance checking as well. A Petri net is used here to model the process. The idea is closely related to the idea of inserting and missing activities introduced in Cook et al. [41, 48].

The deviation between an event log and a process model can be interpreted as either inserting activities, i.e. executing activities that should not happen according to the model, or skipping activities, i.e. activities that should be executed according to the model but they do not occur in reality. The severity of inserting or skipping

activities depends on which activity we are talking about. For instance, in our example, the severity of missing a 'Sign' activity is higher than the severity of missing a 'Send order' activity.

It is worth mentioning here that some skipped activities are not really skipped but they are just not logged in the file. However, the compliance checking technique cannot see more than what is logged in the file. Thus, these activities are considered skipped because they do not appear in the log. This is usually the case of trivial activities for which logging might be costly. For instance, if we have a missing 'Send order' event whereas the goods and invoice are received and the payment took place then most probably this event occurred in reality but not logged.

The severity of inserting and skipping activities, which is defined by the process manager, is translated into cost functions for skipping and inserting the different activities. For instance: saying that $k^i(X) = 2$ indicates that the cost of inserting an activity X costs 2 units, and $k^s(Y) = 1$ indicates that the cost of skipping an activity Y costs 1 unit. Hence, the cost of inserting Y is less than the cost of skipping X.

In their work, the authors argue that there are three issues to deal with when measuring the fitness of a log file. These are: inserted and skipped activities, the severity of deviation and the unobservable activities. Unobservable activities are those activities which are not logged in the file such as OR-split and Or-join. Starting from these issues, the authors propose their cost-based fitness measure.

An executed trace, referred to as an event net, conforms to a process model if there is a complete trace in the event net which appears in the process model. Both sides, i.e. the process model and the event net, are represented by means of a Petri net language. However, the process model is extended with the cost function of insertion and deletion operations.

To compare the two Petri nets, the approach uses a construction based on an extension of the synchronous product of Petri nets. Given the two Petri nets, the idea is to construct a net where the two nets fire the synchronized activities. Non-synchronized activities are also allowed, however, they are penalized by the inserting and deleting cost defined earlier. In case of inserted activities, transitions are fired in the event net without firing a corresponding transitions in the process model. On the contrary, activities are fired in the process model without firing a corresponding transition in the event net in case of skipped activities. During the construction procedure, the costs of inserted and skipped activities are considered as stated in [57].

Once the extended synchronous product is generated, the approach can determine the execution path in the designed model which has the smallest distance from the executed trace. The smaller is the distance, the higher is the fit. The determined path will be the one with the minimal cost of skipped and inserted activities. The computed cost is used later in the fitness measurement formula.

The intersection of the two Petri nets could lead to different scenarios. In the best case, there will be a completed trace in the event net that appears in the process model. This represents a compliant trace where the cost is zero. On the other hand, the worst case occurs when the intersection produces an empty net. In such a case the cost is the sum of the costs of inserting all required activities to produce the event net. The cost in such a case is considered as the upper bound cost and is used to normalize the fitness metric in the formula. The metric is formulated as:

$$f = 1 - \frac{cFit((E, m'_0, m'_f), (N, m_0))}{d(\sigma', \epsilon)} \quad (2.9)$$

Simply, the fitness metric is the closest fit between the event net E and the process model N , divided by the upper bound cost for normalization. The value is always between $0 \leq f \leq 1$.

The work is extended in [57]. The definitions of skipped activities, inserted activities and the severity of deviation are in correspondence with their definitions in the previous work [83]. To measure the conformity, the cost functions are defined. Afterwards, an instance of the Petri net that best matches the recorded trace is constructed. In this context, an instance is a connected part of the Petri net that starts at an initial place and does not contain choices. The instance is constructed by replaying the trace in the process model. Later on, the A* algorithm [59] is used to find the one with the least cost. Once the best match instance is found, the number of inserted and skipped activities can be determined and used to find the fitness value according to the formula:

$$F = 1 - \frac{\sum_{a \in A_s} A_s(a) \times k^s(a) + \sum_{e \in E_i} k^i(a(e))}{\sum_{e \in E_c} k^i(a(e))} \quad (2.10)$$

where:

- k^s and k^i : the cost functions of skipping and inserting activities respectively.
- $a, a(e)$: 'a' refers to an activity type and $a(e)$ is the activity type of an event 'e'
- E_c : the sequence of events of a case 'c' under examination.

- A_s : the multi-set (bag) of skipped activities in case 'c'
- E_i : a set of inserted events

Simply, the fitness value is one minus the ratio of having inserted and skipped activities and the cost of the worst scenario, i.e. when there is no match at all. In the worst case, the best match instance is an empty trace where all activities are considered as inserted activities. Assuming that $k^s = k^i = 1$ and that the cost of skipping the invisible task in the model is zero, then the fitness value of our log file will be:

$$F = 1 - \frac{53 + 85}{265 + 304 + 63} = 0.78 \quad (2.11)$$

The beauty of this technique is that it can deal with events that do not correspond to any of the process activities. These are considered as inserted activities. The fitness value is penalized as more activities are inserted or skipped and it is strongly affected by the definition of the cost functions. Normally, fitness value is $0 \leq F \leq 1$. Mathematically, the value tends to be higher when the cost of inserting activities is relatively higher than the cost of skipping activities. However, if there is a large number of skipped activities and/or the cost function of skipping activities is relatively high then the value could be negative.

In the literature, the cost based fitness approach is used in to measure the conformity degree between a log file and a process model in [63, 66, 68, 73]

In [66], the cost based fitness approach is used to compute an optimal alignment between a partially ordered trace and a partially ordered alignment. According to the authors, a partially ordered trace (p-trace) is a directed acyclic graph defining a partial order over a set of events. On the other hand, a partially ordered alignment (p-alignment) between a p-trace and a process model is a directed acyclic graph defining a partial order over the set of moves between them. A dependency in p-trace from one event to another indicates the first event has lead to the execution of the second event.

To compute an optimal p-alignment between a P-trace and a model, first, the trace is converted to the so-called event net. An event net is a Petri net model representing the behavior of the trace. Next, the event net is joined with the process model to produce the product net. The product net contains three types of transition: log moves, model moves, and synchronous moves. Later, a firing sequence with the lowest cost is computed according to [83] and then replayed on the product net. The result of this replay is an optimal alignment net containing only the visited places, fired transitions and the arcs between them. The optimal

alignment net is converted later to an optimal p-alignment.

To find out the idea alignment, p-alignments are compared. For this purpose, the authors define the true positive, false positives, and false negatives for the three types of transitions in addition to the dependencies. Assuming that the ideal p-alignment is known for a given case then:

For each synchronous move in the optimal p-alignment set, if this move is also found in the ideal p-alignment then it is True Positive (TP), otherwise, i.e. not found, then it is False Positive (FP). For each synchronous move in the ideal p-alignment but not in the optimal p-alignment, this is considered as a False Negative (FN). This definition is also applied to log moves, model moves, and dependencies. Accordingly, the authors introduce the F1-score:

$$F1 = 1 - \frac{2 * TP}{2 * TP + FP + FN} \quad (2.12)$$

The authors of [63] introduce a conformance checking approach based on Integer Linear Programming. The work allows checking all types of constraints. A data aware Petri net is used to represent the process model. The cost based fitness approach is used here to find a control flow alignment between the Petri net and the log file. Next, the control flow alignment is used to construct the ILP problem. In the work, the objective function is to minimize the cost associated with deviations of any constraint different from the control flow.

2.2.3 Replaying-based techniques

Replaying techniques are based on the token games. The idea is to produce each trace in the log file using the process model. Hence, traces are replayed event by event starting from the first one until the last. It is worth mentioning here that replaying based techniques can be used to check sequencing requirements only.

Conformance checker

In 2008, Rozinat et al. [53] presented their conformance checking technique which is considered as a benchmark in the field of compliance auditing. The technique is based on replaying the recorded traces on the process model. The process model is represented by means of a Petri net. The fitness degree of a log file is calculated by counting the number of missing and remaining tokens while the cases are replayed. We say that the log file and a process model completely fit (100%) if the model can generate all cases recorded in the log file.

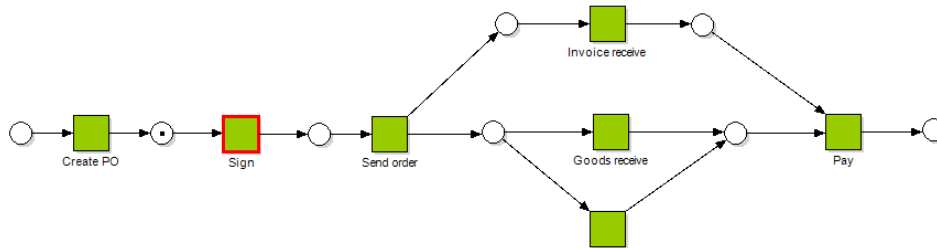


Figure 2.2: The procurement process in Petri net

Figure 2.2 shows the Petri net corresponds to the process model shown in figure 2.1. In Petri nets, a transition (denoted by a square) indicates an activity to be executed while a place (denoted by a circle) indicates a state between two activities. Places are used to hold some tokens during the execution. In this context, a token (denoted by a black dot) refer to a mark that indicates the state of execution. For instance, the place between 'Create PO' and 'Sign' in the figure is marked with a token which means that 'Create PO' has been executed and that 'Sign' is activated to be executed. The unlabeled transition denote an invisible activity.

To replay a trace, a token is produced in the initial place. Afterwards, the events that belong to this specific trace are fired one by one in accordance with their sequence order in the log. Each time an event is fired it consumes a token from each of its input places and produces a token at each of its output places. These are called the consumed and produced tokens respectively. If the trace completely fit the Petri net, then the replay procedure continues without problems. That however is a big if! Usually, there are tokens which are missed during the replay and are required to proceed. Thus, the procedure is carried out in a non-blocking way, i.e. missing tokens are generated automatically and the replay continues until it reaches the end place. Once the end place is reached, the technique checks if there are remaining tokens left behind, i.e. tokens that are produced but not consumed during the replay.

To measure the fitness of the entire log file, recorded traces are replayed. The number of produced, consumed, missing and remaining tokens are counted during the replay of each unique trace and the result is used to find out the overall fitness value according to the formula:

$$Fitness = \frac{1}{2} \left(1 - \frac{\sum_{i=1}^k n_i m_i}{\sum_{i=1}^k n_i c_i} \right) + \frac{1}{2} \left(1 - \frac{\sum_{i=1}^k n_i r_i}{\sum_{i=1}^k n_i p_i} \right) \quad (2.13)$$

where:

- k : the number of traces recorded in the log file.
- n_i : the number of process instances which have the same trace.
- m_i : the number of missing tokens
- c_i : the number of consumed tokens
- r_i : the number of remaining tokens
- p_i : the number of produced tokens

Notice here that the approach groups the similar traces first (n_i) and then replay one of them as a representative for this group instead of replaying all traces individually which will produce the same result but consume more time.

Going back to our example, we can use the equation to find an overall fitness measure for the entire log. First, the log file is filtered to remove all events that do not correspond to an activity in the model. In the example, the only trace affected by filtering is the fifth one because it is the only one which contains such an event, 'Change order'. Next, the recorded traces are grouped into similar traces. In the example, there are six unique traces each of which is replayed separately. We will replay trace 5 here as an example to explain the procedure. After the first step, i.e. filtering, the trace becomes like this: ⟨Create PO, Sign, Send order, Sign, GR, IR, Pay⟩. The first three events can be replayed properly. However, a missing token is required to proceed. The missing token is produced and the procedure continues firing the rest of the events in order until it reaches the end place. The result of replaying this trace is: 8 produced tokens, 8 consumed tokens, 1 missing token, and 1 remaining token. The other recorded traces are replayed in the same manner. Finally, the results are used to measure the fitness of the entire log file.

$$Fitness = \frac{1}{2} \left(1 - \frac{100}{7994} \right) + \frac{1}{2} \left(1 - \frac{100}{7994} \right) = 0.9875 \quad (2.14)$$

In practice, $m_i \leq c_i$ and $r_i \geq p_i$, so that the fitness value is always between 0 (in which none of the traces could be replayed) and 1 (in which all traces could be replayed properly). In this context, a proper replayed trace is the one that could be replayed without missing and remaining tokens.

Although the presented approach is considered as a benchmark in the conformance checking literature, it has some downsides that affect the reliability of the

fitness value. Firstly, the measure is sensitive to the structure of the model used. It punishes a missing event exactly the same as a sequence of missing activities. Secondly, the addition of missing tokens could lead to situations where a sequence of activities is enabled whereas it should never occur, thus the result tends to be overestimated. Concerning duplicate activities, i.e. activities which have the same label in the Petri net, the existence of such cases do not cause a problem unless two events of the same type are enabled at the same time.

In addition to the aforementioned downsides, we do not agree with filtering the log file in the beginning to remove the events which do not correspond to any of the model activities. We believe that these events could be a part of the process which is not represented in the process model. For instance, in the example, 'Change order' activity does not appear in the model although it is normal to take place in reality. Moreover, the removed events could be related to other events in the same trace. Hence, these events should be investigated together. In our example, it could be that whenever an order is changed, it has to be signed again. If this is true then the 'Sign' event should not be penalized as happened when replaying trace 5.

Genetic miner

Genetic miner [51, 84] is a process discovery technique which merges two fields, process mining and genetic algorithm. The mined model is the result of searching a space of randomly generated models. The algorithm starts by generating an initial population of n individuals where each individual represents a process model. To generate the initial population, the dependency between the activities is used to guide the algorithm. Hence, stronger relations have more chances to appear in the generated individuals. Once the initial population is generated, its individuals are evaluated and the fittest are used to generate the next population and so on. Consequently, the final generated population, from which the mined model will be selected, has the fittest individuals.

The conformance between an individual, i.e. a proposed model, and the log file is evaluated by means of a parsing procedure. In this context, parsing is similar to the replay procedure used in [53]. If the activity to be parsed is not enabled, this is recorded as a problem and the parsing proceeds. This is the idea of adding missing tokens when replaying a Petri net.

In the work, the fitness is introduced as an indicator to the number of correctly parsed traces and activities in the event log. According to the authors, a correctly

parsed trace denotes a trace in which the parsing ends with activating the final place. This indicates that the final state is reached properly and that the process is not terminated in between. As shown in the formula below, the metric gives more weight (60%) to the percentage of properly completed log traces rather than the percentage of parsed activities.

$$Fitness = 0.40 * \frac{allParsedActivities}{numberOfActivities} + 0.60 * \frac{allProperlyCompletedLogTraces}{numberOfTraces} \quad (2.15)$$

where:

- *allParsedActivities*: the number of activities that can be parsed without problems.
- *numberOfActivities*: the total number of activities executed by all traces in the log file.
- *allProperlyCompletedLogTraces*: the number of traces that are parsed with no problems.
- *numberOfTraces*: the number of traces recorded in the log file.

The fitness degree is a value between [0,1], where 1 indicates the complete fit. Applying this formula to our example, the result will be:

$$Fitness = 0.40 * \frac{5658}{5758} + 0.60 * \frac{900}{1000} = 0.933 \quad (2.16)$$

Although the metric considers parsing each activity individually, it does not consider the number of missing tokens that are produced to proceed the parsing and the number of extra tokens which are left enabled after the parsing procedure is finished, i.e. remaining tokens. To overcome this weakness, the authors improved their metric in their later work [85, 86]. The new metric, namely *Partial Fitness Complete* ($PF_{complete}$) is in the range between $(-\infty, 1]$, and formulated as the following:

$$Fitness = \frac{allparsedActivities - punishment}{numberOfActivities} \quad (2.17)$$

The punishment value is obtained using the formula:

$$\frac{allMissingTokens}{\#Traces - \#TracesMissingTokens + 1} + \frac{allExtraTokensLeft}{\#Traces - \#TracesExtraTokensLeft + 1} \quad (2.18)$$

where:

- `allMissingTokens`: the total number of activities which cause problems while parsing, i.e. the number of missing tokens of all activities.
- `#Traces`: the number of traces recorded in the log file.
- `#TracesMissingTokens`: the number of traces where there are missing tokens.
- `allExtraTokensLeft`: the number of tokens which are not consumed after parsing is finished, i.e. remaining tokens, plus the number of tokens of the end place minus 1 (decrease 1 for the proper completion).
- `#TracesExtraTokensLeft`: the number of traces in which there are extra tokens left behind after parsing the log file.

In case there are two individuals having the same punishment value, the approach considers the one that can parse more events as better fitting the log. Thus, this individual becomes a better candidate to move to the next generation.

Applying the punishment formula to our example, the result is 0.16. Consequently, the fitness degree is:

$$Fitness = \frac{5658 - 0.16}{5758} = 0.983 \quad (2.19)$$

Comparing with the previous result, the log file has a higher fitness degree. However, this is not always the case.

Heuristics miner

In [50] the authors present a model discovery technique based on the heuristic algorithm. The technique analyzes the dependency between the events of each trace to end up with one model representing the actual behavior. Similar to the genetic miner, the algorithm uses a parsing procedure to evaluate its generated model.

Parsing a trace requires firing its corresponding events one by one according to their occurrence in the log file. Before explaining the parsing procedure used in the heuristics miner, we need first to explain two terms: 'input expression' and 'output expression'. An input expression of an activity 'X' is the set of all possibilities that allow 'X' to be executed, i.e. activate 'X'. For instance, the input expression of 'Sign' in our designed model is {'Create PO'}, and the input expression of 'Payment' is {(IR),(IR ∧ GR)}. Conversely, the output expression of an activity 'X' is the set of all possibilities that are activated after 'X' is executed. For example, the output expression of 'Sign' is {'Release'} and the output expression of 'Release' is

$\{(IR), (IR \wedge GR)\}$. Intuitively, the input expression of the first activity equals the output expression of the last activity equals the empty set $\{\phi\}$.

To explain the idea, we will parse the trace $\langle \text{Create PO, Release, IR, Payment} \rangle$ into our model. The first activity 'Create PO' is fired properly. Firing 'Create PO' will activate 'Sign' as it is its output expression. Next, the second event 'Release' needs to be fired. However, 'Release' is not activated at the moment. This is recorded as a 'missing activated input expression' problem and the parsing continues. Firing 'Release' activates its output expression $\{(IR), (IR \wedge GR)\}$. According to the given trace, 'IR' is fired and finally 'Pay'.

While parsing, the number of correctly parsed events and the number of missing activated input expressions are counted. In addition, the number of activated output expressions that are left behind after parsing each trace is counted. An example of this is the 'Sign' activity in our example which was activated but not fired. Later on, the values of these three counters are used to find out an overall fitness measure referred to as the *Continuous Parsing Measure (CPM)*. CPM indicates the percentage of correctly parsed events and is calculated according to the formula:

$$CPM = \frac{1}{2} \frac{(e - m)}{e} + \frac{1}{2} \frac{(e - r)}{e} \quad (2.20)$$

where:

- e: the number of events recoded in the log file
- m: the number of missing activated input expressions
- r: the number of remaining activated output expressions

According to this formula, the CPM value of our example will be:

$$CPM = \frac{1}{2} \frac{(5758 - 100)}{5758} + \frac{1}{2} \frac{(5758 - 47)}{5758} = 0.987 \quad (2.21)$$

It worth mentioning here that the authors introduce another metric called *Parsing Measure* which indicates the percentage of correctly parsed traces. A trace is considered as correctly parsed trace if it could be replayed in the mined model without any problem. However, this seems to naive because it does not consider the level of deviation. For instance, a trace in which only one event could not be parsed is considered exactly the same as the trace where none of the events could be parsed.

AGNE process discovery

In 2009, Goedertier et al. consider the process discovery issue as a multi-relational classification problem on an event log which is extended with some artificial generated negative events [87]. Accordingly, the authors use the percentage of true positive and true negative to evaluate the output model. For evaluation, the behavioral recall metric is introduced. It indicates the percentage of fitting events in the log.

Similar to the aforementioned replaying techniques, the parsing procedure is used to measure the conformance. The procedure is used to determine the percentage of true positive events. During parsing, each recorded trace is reproduced in the model. Whenever an enabled transition is fired, the number of true positive is increased by one. On the other hand, if a disabled transition is required to be fired, i.e. a missing token, the number of false negative events is increased by one.

To improve the computational efficiency, similar traces are grouped together and only one trace of each group is parsed instead of parsing all recorded traces such [53]. After parsing, the number of true positive and false negative are used to measure the behavioral recall metric value according to the formula:

$$r_B^P = \frac{\sum_{i=1}^k n_i TP_i}{\sum_{i=1}^k n_i TP_i + \sum_{i=1}^k n_i FN_i} \quad (2.22)$$

where:

- k: number of grouped sequences
- n_i : number of similar traces in group i
- TP: the number of correctly classifies positive events
- FN: the number of misclassified positive events.

The behavioral recall of our log file in the example is:

$$r_B^P = \frac{5658}{5658 + 100} = .983 \quad (2.23)$$

Notice here that although the other techniques in the literature which are based on replaying the traces [50, 53] punish for remaining tokens, this technique does not. By remaining tokens here I refer to transitions which are enabled during parsing but not fired. The technique considers these events as extra behavior that are penalized in another metric called the behavioral specificity metric. However, behavioral specificity is out of the scope of this research cause it measures the fitness degree with respect to the log file not the process model.

2.2.4 Constraint-based techniques

These techniques are based on measuring the fitness degree of the log file with respect to some business rules. Thus, they are also called rule based techniques. They are distinguished from the other techniques by using declarative languages to represent the compliance requirements. Hence, the compliance checking procedure is straight forward. A case is compliant if and only if the rule is applied.

LTL checker

The LTL checker introduced in [80] is a compliance auditing technique based on linear temporal logic. The process model is a set of LTL formulae.

Given a log file and a set of LTL rules, the technique introduces two simple metrics: health degree and coverage. The health degree of one case is the percentage of applied rules to the total number of rules. Hence, if there is a case to be checked against 5 rules, 4 of them are applied and one is not, then the health degree of this case is 80%. Coverage indicates to which extent a rule is applied in a log file. For instance, given a log file of 100 cases and a rule that is applied in only 60 cases, we say that the coverage rate for this rule is 60%. The health degree of one specific case is the percentage of applied rules to the total number of rules. The coverage degree of one specific rule is the percentage of correct cases to the total number of cases checked.

Notice here that this technique can be used to measure the fitness degree of one specific case. In addition, it can be used to measure the fitness degree of the log file with against each compliance requirement individually.

2.2.5 A comparison between existing fitness measure

In this section we study the metrics introduced in the literature to measure the fitness degree between a process model and a set of cases recorded in a log file. Doing this, we aim at analyzing the points of strengths and weakness in existing techniques to take this into consideration when proposing our own fitness measurement technique. Hereafter, we conduct a comparison between studied metrics with respect to the following factors.

1. Provide different semantics: does the technique consider measuring the fitness degree from different perspective

2. The type of process modeling language used to represent the compliance requirements.
3. Comparison mechanism: the method used to compare the two parties; the actual behavior, i.e. log file, and the desired behavior, i.e. process model. These are: alignment based, replaying based, and constraint based.
4. Type(s) of compliance requirements which can be checked
5. Which of the following metrics are supported: a) the fitness degree of one case against one compliance requirement, b) the fitness degree of one specific case against all requirements, c) the fitness degree of all recorded cases against one compliance requirement, the fitness degree of the entire process with respect to the process model.
6. The ability to solve mapping problems: can the technique deal with cases in which the log file contains events with no corresponding activities in the process model.
7. Locate the deviation (if any)

Table 2.2 shows the result of comparison. Notice that the majority of existing fitness metrics are introduced to measure the fitness degree with respect to a procedural model. Hence, they can be used to check sequencing requirements only. However, in the business world, it is required to check all types of requirements. This point is related to the process modeling language used to present the normative model. Normally, procedural models are superior in representing the sequencing compliance requirements. However, mostly, they cannot capture the other types. It worth mentioning here that a declarative model can be converted into a procedural model. However, the result of this mapping is not always promising. This depends on the representation capabilities of the process modeling language used. The generated procedural model does not necessary catch all the requirements represented in the declarative model. The capabilities of each modeling language is explained in more details in the next section.

Concerning the techniques in which a the process model is represented by means of a procedural model, the process model is compared with the recorded cases either directly or indirectly. In the direct comparison, the model itself is used to replay the recorded cases such as [50, 53]. In the work of Adriansyah et al., the process model is used to construct a partial instance of the model that best matches the recorded traces [57, 83, 88]. Similarly, Cook et al. [48] use the

model to extract the desired execution paths. In the genetic process miner [85] the so-called causal matrix is used as a representative of the model.

The replaying procedure works properly unless a mismatch is encountered. We agree, to some extent, with the argue of Cook et al. [48] and Adriansyah et al. [83] that a mismatch can be interpreted as either skipping activities or inserting additional activities. Most of the existing techniques penalize inserting and skipping activities. Moreover, some of them give different weights to skipping and inserting the different types of activities [48, 83]. However, none of them count for the relation between the mismatching events. In reality, inserting/skipping an event could affect the other events in the same trace. Taking this trace as an example, $\langle \text{Create PO, Sign, Send order, Change order, Sign, GR, IR, Pay} \rangle$, the log indicates that once an order is changed an additional sign is required. Although this does not match the model, it could be the case that changing an order specification is allowed and that it must be directly followed by a 'Sign' activity. That being the case, the technique should not penalize executing the second 'Sign' event. However, all of the techniques except the LTL do.

Additionally, most of the techniques could not handle the existence of the event 'Change order' in trace 5 in our example. This is because this event does not correspond to any of the model activities. Most of the techniques assume that the events recorded in the log refer to the same set of activities represented in the model. To meet this assumption, the log file is filtered before the parsing is performed. This is conflicting with the idea of procedural models which are designed to represent the process in an easy and understandable way. Normally, these models represent the normal execution especially when the process is very complex and has a lot of activities, choices, and parallel execution. For instance, the 'Change order' activity does not appear in the model but is executed in reality. We assume that changing an order does not reflect the normal execution and thus it is not represented in the model whereas it is recorded in the log file. This issue is solved automatically in the LTL checker. Using a declarative language any behavior is accepted unless it violates one of the business rules.

Regardless of the reason, the log could contain events which do not correspond to any of the model activities. We believe that such events should be considered when measuring the fitness rather than being filtered before. Going back to the same example, filtering is another reason why we do not agree with penalizing the second 'Sign' activity. By looking at trace 5 after filtering 'Change order', its following 'Sign' event does not make sense any more. The event will be treated

as a mismatch although it could be obliged in such a case.

In terms of locating the deviation, compliance checking techniques could provide this feature whereas process discovery techniques do not. This is due to the purpose of each field. When checking the compliance, process managers aim at detecting the deviation and locating it. However, a process discovery technique aims at modeling the actual behavior of the process. The technique generates different models, evaluate them, and choose the optimal one. So, the location of deviation is not of any importance. Process managers are interested in the output model rather than the intermediate evaluation procedure. It worth mentioning here that sometimes there are over 1,000 models to be evaluated. So, it does not worth locating the deviation in all of these models especially when there is no added value for this.

Finally, all of the techniques except the earliest one, process validation [48], provide an overall fitness measure for the entire log file.

Table 2.2: A comparison between the fitness metrics introduced in the literature

Factor	Process validation	Cost-based conformance checking	Conformance checker	Genetic PM	Heuristics miner	AGNE	LTL
Provide different semantics	no	no	no	no	no	no	no
Modeling language	procedural	procedural	procedural	procedural	procedural	procedural	declarative
Comparison mechanism	alignment	alignment	replaying	replaying	replaying	replaying	constraint
Types of requirements	sequencing	sequencing	sequencing	sequencing	sequencing	sequencing	all types
Fitness metrics provided	a	a, d	a, d	d	d	d	a, b, c
Solve mapping problems	no	no	no	no	no	no	yes
Locate deviation	yes	yes	yes	-	-	-	yes

2.3 Process modeling languages

Compliance auditing techniques are based on comparing two parties; the actual behavior (recorded in log files) and the process design (holding the compliance requirements). The process design is expressed by means of a process modeling language. Process modeling languages are categorized into two main categories: declarative and procedural (imperative). In this section we first introduce each of the two categories showing their capabilities in terms of compliance auditing. Then, we conduct a comparison between them with respect to some predefined requirements. Doing this, we aim to answer the following question: which is better to use for compliance auditing purposes: procedural models or declarative models?

Process modeling languages are categorized into two main categories: declar-

ative and procedural (imperative). Each category has a set of modeling languages. For instance, UML, BPMN, Petri nets are categorized as procedural languages. On the other hand, LTL, BCL and FCL rules are categorized as declarative languages. Declarative languages focus on what needs to be done rather than how to do it. These languages depend on the logic to control the relation between the process components (activities, performers, data objects) in terms of time [89]. On the other hand, procedural languages focus on the sequence of activities that should be performed to execute one process instance [90]. It is worth mentioning here that some mapping techniques can be used to convert the model from one format to another such as [91–93]. However, the result is not necessarily trusted. Mapping models is out of the scope of this research.

We agree with the argue of [94] that declarative is no absolute property. Modeling languages are located on a line in between pure declarative and pure imperative languages. In general, control flow models such as Petri nets, BPML, are more regarded as imperative languages while rule-based models (business rules) tend to be more declarative. The authors of [90] position Petri nets at the imperative end and LTL at the declarative end.

Regardless of the process modeling language used these are the main requirements that should be considered for compliance auditing purposes:

- Flexible: keep a balance between flexibility and compliance. In this regard, languages which is capable of representing more compliant ways are more suitable to use.
- Comprehensive: to be able to express all types of compliance requirements (sequence, timing, data, and resources)
- Formal: to have a formal representation that is understandable by the different parties involved in the compliance checking procedure, i.e. compliance checker, process manager, etc.
- Able to represent the compliance requirements at two levels: cases and events. Process managers should be able to represent requirements about individual events such as the performer of one specific event. In addition, they should be able to represent requirements related to the entire process instance such as: the process instance should be closed automatically under some conditions.
- Non-monotonicity: given the dynamic nature of the business world, violations are normal to occur. So, the modeling language should be able to rep-

resent requirements that could be violated under some conditions and how to repair the violation if it occurs.

- Individual requirements: to provide the ability to express each compliance requirement individually to be able to check the compliance against one specific requirement later on.
- Easy to use and understand by non-experts.
- Maintainability: to be able to change the model in case of re-engineering. This can be considered as low level requirement since the design process is compliance auditing. However, it is preferable especially that the compliance auditing results can be used as a feed back to improve the process design.

Comparing the two categories, i.e. declarative and imperative, there are some similarities. However, there are major differences. The most important difference is how each category deal with the flexibility issue. Hereafter, we compare the main characteristics of each group. Accordingly, we tell which category of languages can better support compliance auditing techniques. Next, we conduct another comparison between the languages of the chosen category to select one of its modeling languages.

In declarative languages, any way to execute the process in accordance with the identified rules is an alternative way. Usually, more rules means more constraints and less alternatives[90]. Hence, declarative languages are also referred to as constraint-based models [42, 89]. In contrast, in imperative languages, all execution alternatives should be specified in the model. New alternatives must be explicitly added to the model [89]. A process instance is compliant with respect to an imperative model if the way it is executed is given in the model. On the other hand, the same process instance is compliant with respect to a declarative model if it does not violate any of the business rules.

Business processes today are executed in a flexible nature. It is difficult to be stuck to a limited number of ways to execute the process. Sometimes, there are more efficient ways for execution, however, because of some problems at the design time, they are not represented in the process design and hence they are treated as non-compliant. Normally, in compliance checking it is more important that executed processes are in accordance with the regulations than the way they were executed. Process managers are interested in achieving their goals rather than the way it is reached. Actually, the processes are designed in order to reach these goals.

In terms of comprehensiveness, imperative languages are superior in terms of sequential information (immediately "leading to" and "proceeding by"). On the other hand, declarative languages are superior in defining the relations between the process components such as: if the PO value is greater than a given threshold then the order should be double signed [95]. In practice, control flow models can be enriched with some annotations to represent the other types of requirements, i.e. data flow, temporal and resource requirements [96]. So that to overcome the shortage of representing one dimension. However, these annotations are attached to specific activities. So, they can be used to hold compliance requirements at the event level but not the case. On the other hand, business rules formalism can represent all types of requirements. Nevertheless, there are some limitations depending on the logic which is used for representation.

Regarding the non-monotonicity requirement, it is supported by some declarative languages. For instance, in FCL a repairing chain is defined to deal with non-monotonic rules. However, it is not supported by imperative languages. In imperative languages, the repairing chain concept does not exist. Any path between the start and end is a possible execution path without priorities.

Imperative languages are based on the idea of moving the process objects from one state to another in its state space. In this context, the state space can be seen as the set of locations that the process objects can move in between. The action of moving between two states is the event (also called transition or activity). The focus in imperative languages is to model how the process activities change the process objects between states. Hence, it is easy to represent compliance requirements related to one specific event. However, it is not the case when formulating requirements related to the entire case. For instance, the requirement: "the information about the executed process instances should be kept for at least two years" is not easy to represent in the model. To overcome this limitation, process manager tends to represent such type of requirements in a separate process. For example this requirement can be represented in the archiving business process. Notice that by applying this solution, the process model is separated in (at least) two models. In contrast, in declarative languages states and actions are constructed when interpreting predicates and formulas. Thus, as Fahland et al. mentioned in [90] a room is left for "how the process' changes are continuously linked to each other". Consequently, declarative models are superior in representing compliance requirements controlling the entire case.

As mentioned earlier, maintainability is not highly required for compliance auditing purpose, however, it is preferable. Maintainability is not easy to compare

because there are many factors included in the procedure [97]. However, generally speaking, when changing an imperative model, we need to make sure that we do not add execution ways that should not be allowed and vice versa. Normally, the new model is checked by the process manager to make sure that it represents all execution paths and that undesired behavior is not represented in the new model. On the other hand, when changing a declarative model, we need to make sure that each new rule does not contradict any of the other rules. The contradiction between rules can be checked automatically by means of design time compliance checking techniques. Hence, changing a declarative model is, relatively, easier than changing an imperative model.

In declarative languages, each compliance requirement is expressed by one business rule. Hence, the process manager can check the compliance of each requirement individually. However, this is not possible if an imperative language is used. In imperative languages, all compliance requirements are represented together in one graph.

As for the other two requirements, i.e. ease of use and formality, a general comparison cannot be conducted easily. Each modeling language has its own characteristics regardless of its type, i.e. declarative or imperative. For instance, BPMN and Petri nets are both members in the imperative languages family. However, BPMN model are easy to use and understand by non experts whereas Petri nets requires some knowledge to be able to read the model correctly. Declarative models in general require the process managers to know semantics and syntax of the language to be able to define the business rules. Hence, some work [42] introduce a graphical representation to overcome this problem.

With respect to the above discussion and given our predefined requirements, we believe that declarative modeling languages are more suitable to represent the compliance requirements. Table 2.3 shows the results of the comparison conducted between the two categories. Imperative languages are skipped in the rest of this thesis. Henceforth, the terms "designed process", "business rules" and "rules" will be used interchangeably. Besides, in this context, the term "process model" refer to the "design process model" which is a set of business rules in this case.

2.4 Conclusion

In this chapter we have discussed the related work in the literature. The discussion is divided into three parts: compliance auditing techniques, fitness measurement techniques, and process modeling languages. In the first part, existing compliance auditing techniques are discussed showing the points of strengths and weaknesses of each technique. Secondly, existing fitness measurement techniques are studied

Table 2.3: A comparison between imperative languages and declarative languages with respect to the predefined requirements

Requirement	Imperative	Declarative
Flexibility	–	+
Comprehensiveness	±	±
Formality	+	+
Represent events' requirements	+	+
Represent cases' requirements	–	+
Non-monotonicity	–	±
Individual requirements	–	+
Ease of use	±	±
Maintainability	±	±

thoroughly with an explanatory example. The study of the first two parts, reveal the need for a new compliance auditing approach which can be used to measure the fitness degree at different levels of abstraction. In addition, it shows the importance of supporting different semantics when measuring the fitness degree taking into consideration the process manager interpretation for a compliant process. The last part is devoted to study the process modeling languages used in the literature to represent the process model. There, we conduct a comparison between the two main families of process modeling languages: procedural and declarative and show the importance of using a declarative model to represent the compliance requirements.

Chapter 3

Business Process Modeling for Compliance Auditing

In section 2.3 we conduct a comparison between procedural languages and declarative languages. Accordingly, we choose to represent the normative model using a declarative language. In this chapter we first introduce the concept of declarative languages. Next, we explain the existing declarative languages in the literature, study their capabilities and limitations and then conduct a comparison between them.

The aim is to end up with a declarative modeling language that can be used to represent the normative model in this research. To this end, we went through the following steps: 1) Study existing declarative languages which are used in the field. 2) Conduct a comparison between them with respect to some predefined requirements. Accordingly, tell whether we can use an existing language or that we need to develop a new one. 3) According to the result in (2), if there is a need to develop a new language, then develop it.

In the first section we give an introduction to declarative languages and define our requirements. In the following two sections, we discuss FCL and LTL languages as representatives for their logic families. Then, we conduct a comparison between them with respect to the predefined requirements. Accordingly, we introduce our new declarative process modeling language.

3.1 Declarative process modeling languages

Declarative languages are logic based languages. A declarative process model is a set of business rules controlling the relation between the process objects and the

activities that should be performed in terms of time and space. The relations are characterized by the predicates and They describe the characteristics of a business process regardless of how the process should be executed.

In the literature there are mainly two fields of logic used to model business rules: deontic logic and temporal logic. Most of the research in the field extend the preliminary logic formalism to capture more features that meet their requirements. Some languages use a combination of both, i.e. deontic and temporal logic, such as PENELOPE [25].

Deontic logic is the branch of logic that is concerned with three normative concepts: obligations, permissions, and prohibition. The logic extends the first order logic with three operators to express these concepts: O to express Obligations, P to express Permissions and F to express prohibition (forbidden). For instance, deontic logic can be used to represent the business rule: "It is obligatory that the person who creates a purchase order is not the same as the person who signs it". In the literature, deontic logic is mainly used to represent the norms of business contracts. Examples for the deontic-based languages are: FCL (Formal Contract Language), BCL (Business Contract Language) and PCL (Process Compliance Language). BCL is a domain specific language that is mainly developed to enable monitoring the execution of business contracts [38, 98]. PCL is an extension of FCL with a richer deontic capabilities [1, 99, 100]

As for temporal logic, it is mainly concerned with time. Temporal logic extends the predicate logic by temporal operators that introduce temporalized modalities. Usually, there are two basic temporal operators used in the formalism, "Globally" and "Eventually". In addition, there are operators that denote "Next", "Until", "Release", and "Weak-until". Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) are among the most used temporal based languages for compliance checking. An example of using temporal logic in business rules representation is: whenever the goods are received, next it should be checked and then eventually the supplier is paid.

For the purpose of comparison, we select the most cited language of the each type of logic. Hence, FCL is selected as a representative of deontic logic and LTL is selected as a representative for temporal logic. Although CTL is also mentioned in the literature, we do not consider it here because it does not support the propositional logic which we believe it is important in our study. Moreover, LTL shows more capabilities in practice [101]. In the next sections, both languages are studied in more details. Next a comparison is conducted and finally the conclusion is drawn

up. However, before going further, we first define our requirements which will be used later in the comparison.

1. The ability to check all types of requirements: sequencing, temporal, HR, and data. In addition, it should allow the modeler to represent compound rules and conditional rules. In this work, a compound rule is the rule representing more than one type of requirement. For instance, A should be followed by B within 5 days. A conditional rule is the one which is checked according to some conditions. For instance, if $x > 10$ then A should be followed directly by B. Notice that the later rule is checked just in case the value of x is above 10
2. Allow checking compliance at very low level of abstraction (the occurrence level). This is explained thoroughly in 4, however, the idea is discussed briefly here to explain this point. Suppose that the case to be checked has the trace $\langle A, B, C, B, D, E, B, C \rangle$ and that it needs to be checked against the rule: 'Activity B should be followed immediately by Activity C'. Notice that the rule can be checked three times because the event 'B' appears three times in the trace. Hence, there should be three fitness degrees each of which represents the fitness degree for one of the three B's. We refer to this as the occurrence level fitness degree. Its fitness degree against that rule should reflect that the case is partially compliant. Notice that in this example, the rule is applied in the first and third 'B's whereas it is violated in the second. Hence, it is almost partially compliant. The modeling language should be able to support checking the fitness degree at the occurrence level.
3. Ability to represent cases in which the rule cannot be checked. Assume the case with the trace $\langle A, C, D, F \rangle$ is to be checked against the previous rule, 'B' should be followed directly by 'C'. However, notice that B does not appear in the trace. Hence, the rule is actually not checked! Process analysts should be able to model the scenarios in which a business rule cannot be checked.
4. Ease of use by process auditors analysts. Process analysts should be able to build and understand the models easily.

Moreover, using LTL, the case trace is checked event by event. However, once an event is checked it is not possible to return back to it. Hence, a rule such as A should be proceeded by B cannot be checked using LTL. Moreover, some rules cannot be represented using LTL rules alone. An example for this is rule number

13 in table 3.1. In general, LTL rules are superior in representing temporal requirements. However, it is weak in representing data requirements. To overcome these issues, we have developed a new modeling language called BRCA to represent the compliance requirements.

3.1.1 Formal Contract Logic (FCL)

Also called Formal Contract Language, is a combination of a defeasible (non-monotonic) formalism [102, 103] and deontic logic of violations [104]. The non-monotonicity feature of defeasible logic ensures that there is only one conclusion that can be derived from a set of premises so that contradictory conclusion could not happen [105]. Defeasible logic defines a superiority relation which can be used to give priority for a rule over another in case they are contradictory. As a consequence, the rule with the higher priority will override the conclusion of the rule with the lower priority. Deontic logic provides the ability to reason in case of violations by means of contrary-to-duty obligations (CTD), also called reparation obligations. Due to the dynamic nature of the business world, violations is normal to occur. However, a violation is not always an error. Thus, CTD is used to represent a reparation chain. In this context, a reparation chain is the secondary obligation which needs to be activated as a response to violating the primary obligations.

One of the advantages of using deontic logic in our case is the ability to define new business rules in the future. We do not need to consider checking the consistency between existing rules and the new rules. The modularity feature of FCL is of particular relevance to represent the compliance requirements. It allows the revision of a component of a business process without being forced to perform a complete revision of the business process representation as the case of hard-coded solutions. Moreover, defeasible reasoning is based on constructive proof. Hence, for one conclusion there could be different derivations. These derivations can be used to explain why a specific conclusion has been obtained. This is of high importance in compliance auditing to provide diagnostic information [11].

A rule in FCL is an expression of the form:

$$r : A_1, \dots, A_n \Rightarrow B \quad (3.1)$$

where r is a unique identifier for this rule, A_1, \dots, A_n are the premises and B is the conclusion of the rule. Both premises and conclusion are propositions that are formulated from a finite set of atomic propositions and the operators: O

for (Obligation), P for (Permission), \neg for (Negation) and \otimes for the non-boolean connective to express the CTD operator. FCL formulas apply the following rules:

- Every atomic proposition is a proposition
- The negation of an atomic proposition is a proposition, for instance: if p is an atomic proposition, then $\neg p$ is a proposition
- If p is a proposition, then O_p is an obligation proposition and P_p is a permission proposition. Both obligation and permission propositions are deontic propositions.
- Prohibitions can be represented either as $O\neg$ or $\neg P$.
- If p_1, \dots, p_n are obligation propositions and q is a deontic proposition, then $p_1 \otimes \dots \otimes p_n \otimes q$ is a reparation chain.

The reparation chain can be interpreted as: if P_1 is violated then this can be repaired by P_2 . If P_2 is violated then P_3 is activated and so on until q is activated as a reparation for P_n . Rationally, the permission can only occur at the end of a reparation chain because it can be used to repair a violation but it does not make sense to violate a permission. Moreover, the reparation chain, if any, always appears in the conclusion side. It goes beyond the scope of this research to explain the details of the FCL language. However, interested readers may refer to [104, 106] for detailed explanation.

In some work, the deontic operators are indexed with the party/parties concerned in the rule. In [11] the deontic operators are indexed with the normative position corresponding to the operator. For example, in the procurement process, the proposition $O_{Supplier}SendInvoice$ means that the supplier has the obligation to send the invoice to the purchaser. In the work where FCL is used to represent the norms of business contracts, deontic operators are indexed with the beneficiary in addition to the role who is responsible to perform the activity. For instance, $O_{Supplier,Contractor}SendInvoice$ indicates that the Supplier is obliged to send an invoice to the contractor [26].

To integrate the temporal dimension, Sadiq et al. [11] timestamp all propositions in the language and then adopt the persistence mechanism presented in [107]. The work of Guido et al. [108] does not consider temporal requirements. However, it is concerned with the different types of obligations and when they are fulfilled or violated. To this end, the obligations are classified into: persistent maintenance obligations ($O^{p,m}$), persistent achievement obligations ($O^{p,a}$), and non-persistent obligations (O^n) according to the classification proposed in [109]. Persistent maintenance obligations indicates that given a state s in the executed process instance,

the obligation prevails for all states that come after s . Hence, any state that comes after s and does not fulfill the obligation is considered as non-compliant. This is similar to the "Globally" operator in LTL. For persistent achievement obligations, the obligation is fulfilled if it occurs at some point after s and before the deadline. This is similar to the "Eventually" operator in LTL. However, since temporal constraints are not considered in the work, the authors set the deadline for all obligations to be the deadline of the entire case. Non-persistent obligations are evaluated on a state-by-state basis, so when they arise they must be fulfilled and they do not have consequences on the other states.

After FCL rules are defined, they can be normalized to remove redundancy and identify conflict rules (if any). The normalization process aims at cleaning up the FCL representation: identifying loopholes, deadlocks, contradictory rules, inconsistencies, and making hidden rules explicit. The generated Normal Form FCL (NFCL) are derived from the defined FCL.

Given a set of FCL rules, normalization process consists of three steps [98]:

1. Merging related rules together by linking an obligation with the obligations that are triggered in response to violating this obligation so that to produce the reparation chains. For instance, the rules $\Gamma \vdash OA \otimes OB$ and $\neg A, \neg B \vdash OC$ can be merged in one rule: $\Gamma \vdash OA \otimes OB \otimes OC$
2. Remove redundant rules. Given the rules $\Gamma \vdash OA \otimes B \otimes C$ and $\Gamma \vdash OA \otimes B$ redundancy issue is solved by simply removing the second rule since the first one includes it.
3. Solve the conflicts between the generated rules if any.

To deal with contradictory conclusions of two rules in the last step, FCL is equipped with the superiority binary relation (\prec). The relation is used to give priority for one rule over the other. For example, given the rules: $r_1 : A \Rightarrow B \otimes C$, $r_2 : D \Rightarrow \neg C$ and the relation $r_1 \prec r_2$ we conclude that r_1 has the priority over r_2 . Consequently, if both rules are fired concurrently, the conclusion will be that of rule 1. Notice that C and $\neg C$ cannot be concluded at the same time.

To measure the compliance degree between the rules and execution paths, FCL is equipped with ideal semantics (ideal, sub-ideal, non-ideal, and irrelevant). An ideal situation is the situation where execution paths do not violate the FCL rules. Hence, the process is said to be fully compliant with the FCL rules. A sub-ideal situation describes the situation where at least one rule is violated. However,

this can be repaired according to the reparation chains defined in the rule. Sub-ideal situations are still compliant although they provide a sub-optimal situation. Non-ideal situations are situations where violation(s) occur but they could not be repaired. These situations are considered non-compliant and are caused either by executing prohibited activities or missing some required activities. Finally, an irrelevant situation denotes a situation where none of the FCL rules is applicable.

3.1.2 Linear Temporal Logic (LTL)

LTL is an extension of temporal logic that is widely used to represent temporal constraints. In LTL, each state has one possible future. Hence, formulas are interpreted over a linear sequence of states. In the field of business process management, the linear sequence of states represents one of the possible execution paths. LTL formulas consist of three types of alphabet:

- atomic proposition symbols such as p, q, r
- boolean connectives: $(\neg, \wedge, \vee, \rightarrow)$
- temporal connectives: (X, F, G, U, W, R)

The boolean connections are the same operators which are used in standard propositional logic formulas: \neg for negation, \wedge for and, \vee for or, and \rightarrow for the implication. Temporal connectives, which distinguish the LTL logic among other logics, hold the temporal constraints. The first three operators (X, F, G) are unary, whereas the rest, (U, W, R) , are binary. The operator X denotes next time which is the next state in the sequence. The eventually operator F indicates that the formula will be true once in the future, i.e. next states of the sequence before the end. The global operator G indicates that the formula is always true. This implies that the formula is true in all future states. U represents the until operator. It states: if the first formula is hold until the second formula is true. We say that the formula pUq holds on a path if p holds continuously until q holds. The weak-until operator W can be considered as a special case of U . The only difference is that in pUq , it is required that q does hold in some future state. However, this is not the case in pWq . Release is the dual operator of until, that is pRq is equivalent to $\neg(\neg pU\neg q)$. Release and weak until are quite similar, the difference is that they swap the roles of the two operands p, q . The formula pRq indicates that q must remain true until and including the moment when p becomes true, if any [110].

To formulate an LTL rule, the following rules should be considered:

- If $p \in A$ where A is a non-empty set of atomic propositions then p is a formula.

- If p and q are formulas, then $\neg p, p \wedge q, p \vee q, p \rightarrow q$ are formulas.
- If p and q are formulas, then $X p, F p, G p, p U q, p W q, p R q$ are formulas.

Given that π is an execution path of the business process and according to Backus Naur form [110], we can tell whether π satisfies an LTL formula according to the following syntax:

$$\begin{aligned} \pi := & p \mid True \mid False \\ & \mid \neg p \mid p \wedge q \mid p \vee q \mid p \rightarrow q \\ & \mid X p \mid F p \mid G p \mid p U q \mid p W q \mid p R q \end{aligned} \quad (3.2)$$

In the literature, LTL is used to represent the compliance requirements in different compliance checking approaches. This include design-time and run-time compliance checking techniques as well. In [30] the business rules are first captured by means of a graphical representation language called Business Property Specification Language (BPSL). Next, they are translated into LTL formulas. The authors argue that LTL is difficult to understand and use by compliance experts [89]. Thus, they start with LTL representation directly. The approach is considered as a design-time verification approach where the process is designed first and then verified against the set of regulations. The business process is modeled using BPEL which is then transformed into pi calculus and then transformed again into finite state machine representations to meet the model checker requirements.

The linear temporal logic is used in the literature as a base to introduce some declarative languages such as: ConDec [111], DecSerFlow, CIGDec, and the LTL language presented in the LTL checker work [80]

LTL checker, introduced in 2005, is a compliance auditing technique based on LTL [80]. Compliance rules are expressed as (LTL) formulae. The idea is to verify if a given LTL formula is hold in a set of process instances. The standard LTL logic is extended to include two quantification operators: for all (\forall) and exists (\exists) to check some business rules that cannot be represented using the standard LTL operators alone. Given an LTL rule, the technique divides the log file into two new subsets: one contains the process instances where the rule is followed and the other contains the remaining cases where the rule is not followed. The approach can be used to check the different aspects of compliance requirements, i.e. sequence of activities, temporal constraints, human resources, and data objects. However, it focuses more on sequencing and temporal requirements.

ConDec is a constraint based language which uses the LTL to formulate the con-

straints [111]. To overcome the complexity problem of LTL formulas, the authors introduce a mapping method to associate a graphical representation to each constraint [89]. Hence, non-experts can easily understand and create process models. However, the semantic domain of the language is limited to a specific finite set of activities. For instance, if we need to define a relation between two activities then we are restricted by the given predefined relations [90].

DecSerFlow [112] and CIGDec [113] are also graphical languages grounded in LTL. DecSerFlow is presented to support the design and monitoring of web-services processes. CIGDec is presented to model business processes in clinical organization. Both languages are suitable to be used in their fields. However, they have limited capabilities to deal with all types of business processes.

3.2 A comparison between FCL and LTL

To conduct a comparison between LTL and FCL, we select a set of 13 rules among the most common compliance requirements in the business world [68]. The rules represent the four types of requirements: sequencing, temporal, human resources, and data requirements. Then we present them in LTL and FCL. Table 3.1 shows the 13 rules with their corresponding LTL and FCL rules.

The table shows that both languages have the ability to represent all types of requirements. However, each language has its own points of strengths and weaknesses.

LTL is superior in representing the sequencing and temporal requirements as it is mainly developed for this purpose. It can be used to represent direct follow, eventual follow, and until requirement easily. On the other hand, FCL is superior in representing the consistency between rules and the semantic of contrary-to-duty obligations (CTD) as its mainly developed to represent the compliance requirements emerging from business contracts. Hence, it is equipped with the non-monotonicity feature that is important for designing and monitoring business contracts. The feature allows giving priorities to one rule over another. So that, non-monotonic rules can be represented as a reparation chain. Concerning complexity, both languages are not easy to use and understand by non-experts. Notice that compliance auditors are usually not familiar with such types of modeling. Therefore, some of the related work propose using a graphical model as a first representation [30, 31].

For compliance auditing, LTL better matches our requirements. Although, it has

some limitations such as representing CTD and consistency checking. However, compared to the FCL limitations, these are less importance in compliance auditing. Furthermore, capturing the sequencing requirements is of high importance for this study which is efficiently supported by LTL.

Table 3.1: Most Frequent Compliance Requirements

No.	Rule description	FCL	LTL	Req. type
r1	Activity B should be executed at some time after Activity A	$A : t \vdash O(B : k \wedge k > t)$	$G(A) \rightarrow F(B)$	sequencing
r2	Activity B should be executed immediately after Activity A	$A \vdash OB$	$G(A) \rightarrow X(B)$	sequencing
r3	Activity B should be executed immediately after Activity A otherwise C should be executed	$A \vdash OB \otimes OC$	cannot be represented	sequencing
r4	Activity A should not start until Activity B is performed	cannot be represented	$G(!AUB)$	sequencing
r5	There is no loops of Activity A	$A \vdash O(!A)$	$G(A) \rightarrow X(!A)$	sequencing
r6	Activity A should be executed before Time t	$O(A) : t$	$GA : t' \rightarrow t' < t$	temporal
r7	Activity A should be executed after Time t	cannot be represented	$GA : t' \rightarrow t' > t$	temporal
r8	Activity A should be executed in the time between $(t1, t2)$	$O(A) : t2, t2 > t1$	$GA : t' \rightarrow t_1 < t' < t_2$	temporal
r9	Activity A should be performed by Person P	$O(A_P)$	$G(Activity = A) \rightarrow person = P$	HR
r10	Person P should perform Activity A	$person = P \vdash O(Activity = A)$	$G(person = P) \rightarrow (Activity = A)$	HR
r11	Activities A & B should be performed by two different people	$A_{(person(P))}; B \vdash O_P ! B$	$G(A_{(person(P))}) \rightarrow G(! (B_{(person(Q))}))$	HR
r12	If the value of parameter $Param$ is above a given threshold then Activity A should be performed	$Param > threshold \vdash O(A)$	$G(param > threshold) \rightarrow F(A)$	data
r13	If A is executed then B should be executed where $(A.key) = (B.key)$	cannot be represented	cannot be represented	Multi

To conclude, FCL is more suitable for design-time compliance checking. On the other hand, LTL is more suitable for compliance auditing. The comparison between FCL and LTL logic shows that LTL is more capable of representing the requirements for compliance auditing purposes. However, it still has some limitations which makes it difficult to use in this research alone. The two most important limitations of LTL is: 1) it cannot be used to represent cases in which the business rules cannot be checked, 2) it cannot be used to represent rules which needs to be checked at levels lower than the case level. Hence, we develop a new declarative process modeling language entitled with BRCAL language. The new language extends the important feature of LTL which is required in for this research. In addition, it overcomes the limitations of LTL. In the next section, we describe the BRCAL language and show how it can be used to build process models.

3.3 BRCA language

BRCAL stands for Business Rules for Compliance Auditing Language. It is developed to represent the compliance requirements which will be used to audit the executed process instances. A BRCAL model is a set of business rules each of which represents one of the compliance requirements.

BRCAL inherits some of the LTL operators such as next and eventually. It can be used to represent the four types of compliance requirements. It overcomes the limitations of existing languages in terms of compliance auditing. It can be used to write business rules that can be checked at very detailed levels (occurrence level). In addition, the modeler can select cases in which the rule cannot be checked. Moreover, the language can be used to write conditional rules. A conditional rule, in this context, is a rule which is checked according to some conditions.

Notice here that the language is developed mainly for compliance auditing. However, it can be used to check the compliance at the run time.

The input format

The BRCA language is developed to define constraints on a log file of recorded cases. A log file can be seen as a set of cases, also called process instances. Each case has a set of attributes and a trace. The attributes hold some data corresponding to the case. Some attributes are essential such as the timestamp and the performer. The trace shows the sequence flow of events performed to

execute this case. The number of events in one trace is referred to as the trace length. Each case has a unique identifier (normally called case ID) in the log file.

Figure 3.1 below (resource: [62]) shows information about the first three cases in a log file. In addition to the case ID, information about the event ID, the timestamp, the activity name, the resource and the cost are recorded. Notice here that an event is a performed activity (also called task), thus the two terms are used interchangeably. As mentioned earlier, the case ID is a value that uniquely identifies each case. The timestamp attribute shows the time at which the activity is performed. The sequence flow (trace) is generated according to the timestamp. The resource (also called originator) in the figure shows the person who performed the corresponding activity. The activity, the timestamp, and the resource are among the most important attributes that are recorded in the log files. The figure shows two more attributes, event ID and cost. Notice that the number of events performed in each case, i.e. the trace length, is not fixed. For instance, case 1 has five events whereas case 3 has nine events.

In the figure, case 2 is executed by performing five events: register request, check ticket, examine casually, decide, and pay compensation. The register request event was performed on the 30th of March 2010 at 11:32. It was performed by Mike and costed 50 unit.

Case id	Event id	Properties				...
		Timestamp	Activity	Resource	Cost	
1	35654423	30-12-2010:11.02	Register request	Pete	50	...
	35654424	31-12-2010:10.06	Examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	Check ticket	Mike	100	...
	35654426	06-01-2011:11.18	Decide	Sara	200	...
	35654427	07-01-2011:14.24	Reject request	Pete	200	...
2	35654483	30-12-2010:11.32	Register request	Mike	50	...
	35654485	30-12-2010:12.12	Check ticket	Mike	100	...
	35654487	30-12-2010:14.16	Examine casually	Pete	400	...
	35654488	05-01-2011:11.22	Decide	Sara	200	...
	35654489	08-01-2011:12.05	Pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	Register request	Pete	50	...
	35654522	30-12-2010:15.06	Examine casually	Mike	400	...
	35654524	30-12-2010:16.34	Check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	Decide	Sara	200	...
	35654526	06-01-2011:12.18	Reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	Examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	Check ticket	Pete	100	...
	35654531	09-01-2011:09.55	Decide	Sara	200	...
	35654533	15-01-2011:10.45	Pay compensation	Ellen	200	...
4	35654641	06-01-2011:15.02	Register request	Pete	50	...
	35654643	07-01-2011:17.06	Check ticket	Mike	100	...

Figure 3.1: An example for a log file

The attributes are of two types: event attributes and case attributes. Event at-

tributes are data fields which hold information about one event such as the timestamp and performer. Case attributes are data fields which hold information about the case itself regardless of the executed events. The attributes in the figure are all event attributes.

The log file is checked case per case. The information recorded about one case is independent from other cases.

Definitions

Variables are defined as objects regardless of their data type. In BRCAL, it is not required to define variables explicitly, however, they are allocated on their first use. Once a new attribute is defined, its value is assigned to 0 automatically. Notice that an attribute can hold any type of data, integer, real number, text, date, event, etc. Recall here that there are two types of attributes: case attributes and event attributes. A case attribute holds information corresponding to one process instance, i.e. a case. An event attribute holds information corresponding to one event in the case trace.

Comments

Comments are useful to add explanation to the model. Commented lines are not considered by the parser. Using comments helps creating an easy readable model, especially by non-modelers. A one line comment starts with a percentage symbol (%). Any text written to the right of the (%) symbol is discarded by the parser as shown in the following example.

Example:

% This is a one line comment, this text is skipped by the parser

Multiple line comments are written between the */** and **/* symbols. The */** symbol defines the start of a multiple line comment. The **/* symbol defines the end of a multiple line comment. These comments are handy when the modeler wants to write a comment that extends over many lines.

Example:

/ This model holds the procurement process model, it contains 5 business rules*

Date: 25/4/2016

Author: Nour Damer

*End of multiple line comment */*

Operators

Relational operators: Rules are checked according to some relational operators. The operators used depend on the data type to be compared. The comparison process includes three elements, these are (in order from left to right): the left hand side operand, the operator, and the right hand side operand. For example in the ($x < 10$) comparison process, the left hand side is the value assigned to the variable called x , the right hand side is 10, and the operator is less than.

The evaluation starts at the left hand side, then the right hand side, and finally the values of the two sides are compared. The result of the comparison is either a True or a False. In this example, if the value of x is less than 10 then the result is True, otherwise, the result is False. According to this comparison, if the value of x equals 10 the result is False. Table 3.2 summarizes the logical operators used in the language with some examples assuming that: Person = "Maria", $X = 10$, $Y = 5$

Table 3.2: Relational operators in BRCA language

Operator	Name	When True?	Example
==	Is equal	When the value in the right hand side equals the value in the left hand side	Ex1: Person == John Result = False Ex2: X==10 Result = True
!=	Not equal	When the values in the two sides are not equal	Ex1: Person != "John" Result = True Ex2: Y != X/2 Result = False
<	Less than	When the value at the left hand side is less than the value in the right hand side	Ex: Y < 5 Result = False
<=	Less than or equal	When the value at the left hand side is less than the value in the right hand side or equal to it	Ex: Y <= 5 Result = True
>	Greater than	When the value at the left hand side is greater than the value in the right hand side	Ex: X > 5 Result = True
>=	Greater than or equal	When the value at the left hand side is greater than the value in the right hand side or equal to it	Ex: X >= 5 Result = True

Temporal comparison operators: Two operators are defined to compare values of type date. These are: Before and After. Both operators reflect their meaning. Suppose that $T_1 = 1/1/2015$, and $T_2 = 1/1/2016$ then:

T_1 Before $T_2 = \text{True}$

T_1 After $T_2 = \text{False}$

Logical operators: BRCAL extends three logical operators from the predicate logic. These are: AND, OR, and NOT. The first two are binary operators. However, NOT is a unary operator. The results of logical statement are defined as follows:

- AND: the result of A and B is true if and only if both of them are evaluated to True.
- OR: the result of A or B is true if at least one of them is evaluated to True.
- NOT: is simply the negation of A. The result is True if and only if A is false.

Table 3.3 shows the results of the three operators.

Table 3.3: Logical operators in BRCA language

A	B	A AND B	A OR B	NOT B
True	True	True	True	False
True	False	False	True	True
False	True	False	True	False
False	False	False	False	True

Functions

The BRCAL has a set of functions used to formulate the business rules. Hereafter, we define these functions showing the parameters (input), result (output) and an example for each. For the examples, assume that:

t: ⟨A, B, D, C, E, D⟩

x= "D", z = "A", w = "M"

- **count(x in y)**

Parameters:

x: an event of a activity type x.

y: the trace of the case under investigation.

Return: An integer holding the number of occurrences of event x in trace y.

Example:

count(x in t) = 2.

- **exist(x in y)**

Parameters:

x: an event of a activity type x.

y: the trace of the case under investigation.

Return: Boolean. True if trace y contains an event of type x. False otherwise.

Example:

x: exist (x in t) = True.

exist(w in t) = False.

- **first(y)**

Parameters:

y: the trace of the case under investigation.

Return: The first event in trace y.

Example:

first(t) = A.

- **last(y)**

Parameters:

y: the trace of the case under investigation.

Return: The last event in trace y.

Example:

last(t) = D.

- **previous(x)**

Parameters:

x: an event of activity type x.

Return: The event proceeding x in the trace. The result is null if x is the first

event.

- **next(x)**

Parameters:

x: an event of activity type x.

Return: The event which follows event x directly. The result is null if x is the last event in the trace.

Example:

$\text{next}(B) = D$.

- **follow(x,y)**

Parameters:

x: an event of a activity type x.

y: an event of a activity type y.

Return: The first event of activity type y that follows event x eventually. The result is null if x is not followed by an event of type y.

Example:

$\text{follow}(z, x) = \text{first D in the trace}$

$\text{follow}(x, z) = \text{null}$

- **length(y)**

Parameters:

y: the trace of the case under investigation.

Return: An integer holding the number of performed events in trace y.

Example:

$\text{length}(t) = 6$.

- **position(x)**

Parameters:

x: an event of a activity type x

Return: the 1-based index of event x in the trace.

Controls

Two types of controls are defined in BRCAL: the conditional control (if statement) and the looping control (for loop).

Conditional control

The if statement is defined and used as a conditional control. As usual, if statement is defined here to control the execution of some statements. According to some conditions, some statements are either executed or not. The general structure of the If conditional statement is:

```
if (condition)
    Statement
```

The statement is executed only if the condition is evaluated to 1. For instance, if person = "John", y=0, given the following if statement:

```
If (person == "John")
    y = y+1
```

The condition is first evaluated. In this case, the result is True (5 is greater than 0). Accordingly, the statement (y=y+1) is executed. Hence, the value of y becomes 1.

Looping control

As a looping control, a for control is defined. In this research, looping is mainly defined and used to go through all events in a given trace. The beauty of for looping is that it start from an initial state until an end state. Hence, it better matched the requirement of this research. For instance, suppose I need to write a business rule in which I need to check that the events of type "A" are always executed by person "J". In this case, we can use a for loop to check each event in the trace starting from the first event until the last one and check the value of its HR.

The structure of for looping in the language is defined as the following:

```
for (param = event in case)
    Statement
```

Writing rules using BRCA language:

The BRCA model is a set of business rules each of which representing one of the compliance requirements. Hence, each rule can be seen as a constraint which should be satisfied. Rules are checked per occurrence, i.e. one occurrence against one business rule per time. To explain the idea of occurrence in this research, we consider the following example: suppose the trace $\langle A, B, C, B, D \rangle$ is to be checked against a business rule indicating that activity B should be followed directly by activity C. However, the trace contains two events of type B. Hence, the rule is checked two times; one for each B. In this case we say that the rule can be checked

twice (two occurrences) each of which has its own result. In our example the rule is applied in the first occurrence however, it is violated in the second. Notice that the number of occurrence could be zero. For instance, suppose that B does not appear in the trace of our example. The occurrence concept is discussed in more details in Chapter 4.

Given a case and a business rule to be checked. Each occurrence in the case is checked against the business rule. Hence, each occurrence has its own fitness degree. The result at the occurrence level is either zero or one. The variable (result) is declared automatically to hold this result. It is initiated with the value zero.

Rules are written according to the following syntax:

The languages has a set of predefined business rules which can be used immediately by the end user. These rules are among the most common rules which are frequently checked.

Hereafter, we use the BRCA to represent the most common business rules. First, the rules which can be checked only once are discussed, then the others.

1. **A should be the first activity:** this rule is defined to checked whether the first event is of type A.

It is written using an if conditional statement and two functions: first and position. The result is one if the position of the first occurrence of A is 1 and 0 otherwise. Notice that the result value is increased by one just in case the rule is applied. Also, notice that this rule can be checked exactly once. Hence, no need to use a for statement to represent it. The rule is used in case the process should be started in a specific way.

if position(first(A in Case)) = 1

result + 1

2. **A should be the last activity:** this is very similar to the previous rule, however, it is defined to checked whether the last event is of type A. Thus, the position value is compared with the result of calling the length function. Notice here that the length function returns the number of events in the case trace which is equal to the position of the last event. The rule is used in case the process should be terminated in a specific way.

if position(last(A in Case)) = length(Case)

result+1

3. The process is executed within D days: this is a temporal rule. It does not depend on the occurrences. However, the difference between the timestamps of the last and first events is calculated. If it is less than the value D then the rule is applied in that case. This rule is used to analyze the throughput time of the process.

If last(A in Case)[time] - first(A in Case)[time] <= D days

result+1

4. A should be executed eventually: simply, if an event of activity A appears in the case trace then the result is 1. Otherwise it is zero. This rule is used to check whether a specific activity is performed or not.

if exist(A in Case)

result+1

5. If A exists, then B exists: the rule is checked in cases where executing activity B depends on executing activity A regardless of their order in the case trace.

if exist(A in Case) and exist(B in Case)

result+1

6. If A exists then B exists where $A(X) = B(Y)$: this can be seen as an extension of the previous rule. Suppose I have two activities that can be performed multiple times in the same case. However, each time A is executed a corresponding B should be executed as well. This rule can be used to represent such a case. *key* in the rule is an event attribute. It is used recognize the corresponding event B. The result is one just in case there is a B event in the trace with a key value equals to the key value of A. Notice that once the corresponding event is found, the checking procedure is terminated by a break statement.

for(x = A in Case)

for(y = B in Case)

if x[key] == y[key] % attribute key of x

result+1

exit % break

7. A should be followed directly by B: this rule is equivalent to (A next B) in LTL. It is a sequential rule representing that if an event of type A is performed

then it should be followed immediately by an event of type B. Hence, the rule is checked as much as there are A's in the case trace. The for loop is used in the rule to go through all events in the sequence. Notice that each time the rule is checked, i.e. each occurrence, the result of this check (whether 0 or 1) is recorded.

```
for(x = A in Case)
  if next(x) == B
    result+1
```

8. A should be followed eventually by B: this rule is equivalent to (A eventually B) in LTL. The rule indicates that once an event of type A is performed, it should be followed by an event of type B, however, unlike the previous rule, B should follow A anytime before the process instance is terminated. This rule is checked for each A. Notice that regardless of the number of A's in the trace, one B at the end is enough to consider all of them compliant.

```
for(x = A in Case)
  if follow(x, B)
    result+1
```

9. A should be followed eventually by B or C: this rule use a complex conditional statement. Using the Or operator between the two conditions indicates that at least one of them should be evaluated to True to consider the occurrence under investigation as completely compliant. The rule is checked as much as there are A's in the case trace.

```
for(x = A in Case)
  if follow(x, B) or follow(x, C)
    result+1
```

10. A should be followed directly by B then directly by C: this rule indicates that if an event of type A is executed then this A should be followed directly by B and this B should be followed directly by C. In other words, if A is performed, then the chain (A, B, C) should appear starting from this A.

```
for(x = A in Case)
  if next(x) == B and next(next(x)) = C
    result+1
```

11. A should be followed eventually by B then eventually by C: this rule indicates that if an event of type A is executed then this A should be followed eventually by B and this B should be followed eventually by C. In other words, if A is performed, then the chain (A,..., B,..., C) should appear.

```
for(x = A in Case)
if follow(x) == B and follow(next(x)) = C
result+1
```

12. B before A directly: unlike the above mentioned rules which checked the following events in the sequence, this rule checks the proceeding events. It indicates that an event of type B should be preceded by an event of type A in the trace. Notice that this is not equivalent to (A should be followed directly by B). Although both rules are used to detect the sequence (A, B). However, this rule is checked according to the existence of B. However, that rule is checked according to the existence of A.

```
for(x = B in Case)
if previous(B) == A
result+1
```

13. SOD(A, B): this is the famous segregation of duties rule (also called the 4 eyes principle). It is used to check whether two activities are executed by the same person or not. The result is True if there is no intersection between the performers of A and the performers of B.

```
for(x = A in Case)
for(y = B in Case)
if x[person] == y[person] % attribute person of x
result+1
```

14. A should be executed by person P: this is a simple rule used to control that an activity A is executed by a specific person P. It is checked for each A in the case trace.

```
for(x = A in Case)
if x[person] <> P
result+1
```

15. A is executed between time t_1 and t_2 : this a temporal rule which is used to check whether an events of type A are executed in a given time slot.

for(x = A in Case)

x[time] >= X and x[time] <= Y

result+1

16. If (Case attribute T [$<$, $<=$, $=$, $>$, $>=$] value) then A should be followed eventually by B:

Notice that this rule can be divided into two parts, the if clause and the main clause. The if clause contains the condition that should be applied to check the rule in the main clause. For instance, in this rule, the part after then, i.e. A should be followed eventually by B, is checked just in case the condition in the if clause is evaluated to True. Notice that the condition here depends on the value of a case attribute. As mentioned earlier, case attributes are different from event attributes. A case attribute is a property for the whole case. However, an event attribute is a property for one specific event such as the person who performed the event and when it is performed. For instance, suppose the process manager aims at checking a rule for a specific type of customers then he/she can define such a rule.

if Case[T] <= value

for(x = A in Activity)

if follow(x, B)

result+1

3.4 Conclusion

In this chapter, we study the declarative process modeling languages used in the literature to represent compliance requirements and conduct a comparison between them with respect to some predefined requirements. Results show that existing languages have two main problems. They do not allow for: 1) representing cases in which a business rule cannot be checked, 2) representing rules that can be checked at the occurrence level. Hence, we introduce a new process modeling language called the BRCA language which is developed mainly to represent the compliance requirements for compliance auditing purposes. The introduced language can be used to define a business rule which can be checked at different

levels. It also provides the ability to represent when the rule should be considered neither applied nor violated but (not checked)!

Chapter 4

Flexible Compliance Auditing Framework

The study of existing compliance auditing approaches reveals the need for a new approach which supports different definitions of a compliant process in the business context. Given a log file and a process model, all existing techniques provide one output measure. However, process analysts have different interpretation for the same value. What is seen as compliant from one perspective, is considered non-compliant from another perspective. For this purpose we develop a compliance auditing framework entitled with *Flexible Compliance Auditing Framework*.

The framework is developed to provide a flexible approach to measure the fitness degree at different levels of abstraction. It is based on using some different aggregation operators to measure the compliance degree. Hence, it provides the foundation to construct compliance measures which have the proper semantics. This is an important feature for process auditors. It provides them with fitness metrics that match the context in which they are applied.

In mathematics, aggregation is the process of replacing a set of values by a single representative one using an aggregation operator such as the average [114, 115]. For instance, the average of the students' scores in one course is a representative value for the students performance. In this research we apply aggregation to combine a set of fitness degrees at low levels to end up with one value representing the fitness degree at higher levels.

Hereafter, we first introduce the general framework. Then, we discuss the fitness metrics used to measure the fitness degree at different levels of abstraction.

Later, we explain the concept of aggregation functions and how they are applied in this research.

4.1 The flexible compliance auditing framework

The framework is developed for compliance auditing purposes. Unlike existing compliance auditing approaches, which provides one fitness degree regardless of the context, this framework can be used to check the compliance for a specific business context. It provides a fitness measurement technique which can be customized according to the process manager clue.

The framework, presented in Figure 4.1, starts by reading a log file and a normative model. The log file, which is extracted from operational databases, contains information about the executed processes. The normative model represents how the process should be executed. In this framework, the normative model is expressed by means of the BRCA language which is introduced in chapter 3. A BRCA model is a set of business rules. Each of which represents one of the compliance requirements.

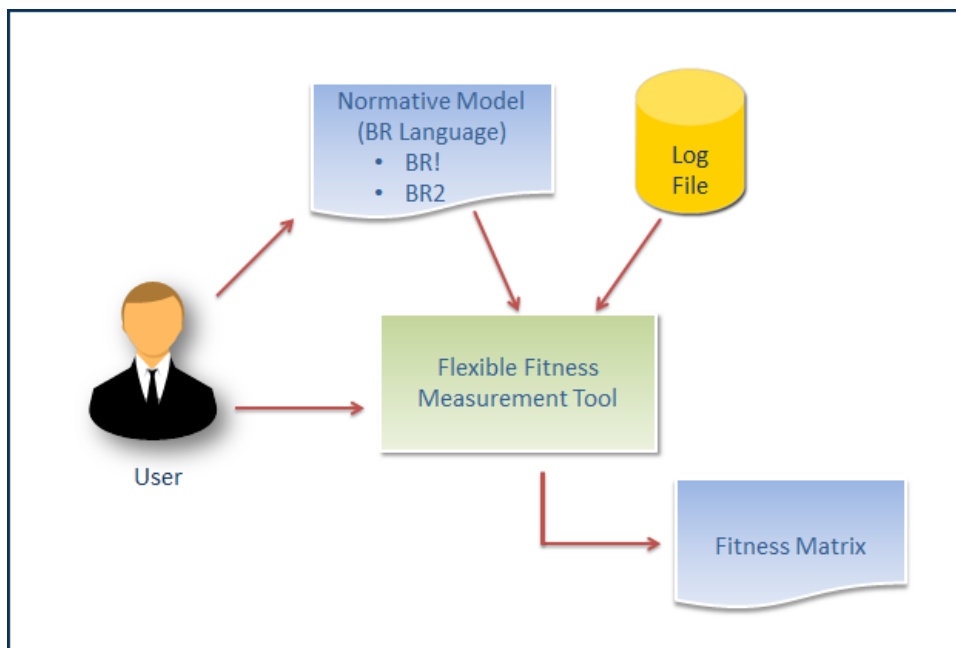


Figure 4.1: Flexible Compliance Auditing Framework

After reading the log file and the BRCA model, the process auditor needs to

select a suitable aggregation operator. An operator is considered suitable if has a proper semantic. It is selected according to the process under investigation. Hence, there is no absolute best operator. Each operator is the most suitable for a specific business context (environment).

The output of the framework, which we call the *fitness matrix*, is a two dimensional table. It shows the fitness degrees measured at four levels: process, case, rule and cell. The process fitness degree is the fitness degree of the entire log file, i.e. all executed processes with respect to the normative model. The case fitness degree is the fitness degree of one specific process instance against the normative model. The rule fitness degree is the fitness degree of all recorded cases against one business rule (compliance requirement). Recall here that the normative model is a set of business rules each of which represent one of the compliance requirements. The cell fitness degree represents the fitness degree of one case against one business rule.

Table 4.1 shows the fitness matrix for c cases and r rules. Cases information are aligned horizontally whereas rules information are aligned vertically. The intersection between a row $_i$ (case) and a column $_j$ (rule) holds the fitness degree of case $_i$ against rule $_j$. For instance, $f(C_2, R_1)$ is the fitness degree of case 2 against rule 1. This could be any value in the interval [0-1].

Table 4.1: Fitness matrix given a log file of C cases and a BRCA model of R rules

	Rule 1	Rule 2	Rule(r)	Fitness _(Cases)
Case $_1$	$f(C_1, R_1)$	$f(C_1, R_2)$	$f(C_1, R_r)$	$F(C_1)$
Case $_2$	$f(C_2, R_1)$	$f(C_2, R_2)$	$f(C_2, R_r)$	$F(C_2)$
.
.
.
Case $_c$	$f(C_c, R_1)$	$f(C_c, R_2)$	$f(C_c, R_r)$	$F(C_c)$
Fitness _(rules)	$F(R_1)$	$F(R_2)$	$F(R_r)$	F

The last column, called Fitness_(Cases), holds the fitness degrees of each case individually. Each cell in that column is the aggregation of all values in the same row. For instance, $F(C_1)$ represents the fitness degree of Case $_1$. The last row, called Fitness_(Rules), holds the fitness degrees against each of the business rules individually. Each cell in that row holds the fitness degree of the entire log file against its corresponding rule. For instance $F(R_2)$ represents the fitness degree against Rule $_2$. The cell at the bottom right, called (F), holds the fitness degree of the entire log. This value is generated by one of three ways: aggregating the

values in the last column (cases), aggregating the values in the last row (rules), or aggregating all the fitness degrees at the cells level.

In addition to these four metrics, the framework generates a fifth metric which we call the occurrence level fitness metric. However, it is not shown in the output matrix. The occurrence level is the level at which the compliance is actually checked. It is the atomic unit used to produce the other four metrics. Once the fitness degrees at the occurrence level are measured, the results can be used to generate the fitness degrees one level upper, the cells level. Later, the cells fitness degrees are used to generate the fitness degrees at the remaining three levels: cases, rules and process.

The beauty of this framework is that it provides the foundation to construct compliance measures which have the proper semantics. Hence, process auditors can use it to measure the fitness degree with their business environments. In addition, it provides fitness metrics at different levels of abstraction. Moreover, it allows checking all types of compliance requirements: sequencing, temporal, HR, and data because it is based on a BRCA model which enables representing all types of compliance requirements.

In the next section we discuss the fitness measurement approach used to measure the fitness degrees at each of the five levels individually.

4.2 Fitness measurement technique

In this section we explain the fitness measurement approach used in our framework to measure the fitness degree at the different levels of abstraction. To have a standard measure, we define the fitness degree to be bounded between $[0,1]$ where 0 indicates the lower bound (completely violated) and 1 indicates the upper bound (completely fit).

As mentioned earlier, the developed framework provides five different metrics: occurrence, cell, case, rule and process. The occurrence fitness degree is the atomic unit. Hence, the approach starts by measuring the fitness degrees at the occurrence level. Next, the results are used to measure the fitness degrees at the cells level. Later, the fitness degrees at the remaining upper levels, i.e. case, rule, and process can be measured using the cell level fitness degrees.

The occurrence level is the most detailed one. It is the level at which the log file and the normative model are actually compared. But what is an occurrence? Suppose we need to measure the fitness degree of trace $\langle A, B, C, D, B, E, B, C, G \rangle$

against rule “Activity ‘B’ should be followed directly by Activity ‘C’”. That being the case, we look for B first and then check its direct follow event in the trace. If it is C then the rule is applied, otherwise, it is not. In our trace there are three events of type B . Thus, the rule is checked three times. Each time a rule is checked in a specific case is referred to as an *occurrence* of this rule in that case.

Sometimes, the number of occurrences k depends on the rule to be checked regardless of the case under investigation. For instance, the rule “Activity ‘A’ should be always the first activity” is checked exactly once in any case because there is always one first activity whose value is compared with ‘A’. The rule in our example can be checked as many as there are ‘B’ activities in the trace (three times in here).

If k denotes the number of occurrence and L denotes the trace length of the case under investigation then: $k \in \mathbb{Z}_{\geq 0}, k \leq L$.

Each occurrence has its own fitness degree which we refer to as the *occurrence fitness degree*. Given an occurrence and a rule to be checked, there are one of two possibilities: the rule is either violated or applied at this occurrence. If the rule is violated, then the fitness degree of the occurrence is 0. In contrast, if the rule is applied, then the fitness degree of this occurrence is 1.

Definition: Let c be the case under investigation, r be the rule to be checked, O_k is occurrence number k of rule r in case c , $Fitness_k(c, r)$ is the fitness degree of O_k , then:

$$F_k(c, r) = \begin{cases} 0 & , \text{ if the rule is violated at occurrence } k \\ 1 & , \text{ if the rule is applied at occurrence } k \end{cases} \quad (4.1)$$

According to this definition, the fitness degrees of the three occurrences in the above mentioned example are (in order): 1, 0, 1.

However, what if there is no occurrence? Assume the trace $\langle A, C, D \rangle$ is to be checked against the same rule: “Activity ‘B’ should be followed directly by Activity ‘C’”. In such a case, the rule is not checked because, simply, there is no ‘B’.

We believe that in such cases (i.e a rule cannot be checked in a specific case) the result should be neither compliant (1) nor non-compliant (0). Intuitively, including these cases in the measurement will bias the result to the compliant border (if they are considered compliant) or to the non-compliant border (if they are considered non-compliant). One may argue that in such cases the rule is not violated, thus these cases should be considered as compliant. On the other hand, one may argue that they should be treated as non-complaint because the rule is not applied. Nonetheless, both arguments are not valid! We prove our debate here by means

of a synthesized data set. Given the log file and a business rule, we computed the rule fitness degree three times. Once after discarding the non-checked cases, once with considering them as compliant, and once with considering them as non-compliant. Details are discussed in section 5.2. Accordingly, we exclude these cases when computing the fitness degree. However, to distinguish them from other cases we return a dash (-) in the fitness matrix.

Once the fitness degrees of all occurrences of one rule in a specific case are measured, the results are aggregated to end up with the cell fitness degree of this case against that rule. This is called the *cell fitness degree* in this work. Notice that although the occurrence fitness degree is the atomic unit to measure the other metrics, however, it does not represent a valuable information individually. Hence, we keep it hidden in the fitness matrix.

The cell fitness degree (one case against one rule) denoted by $F(C, R)$ is the aggregation of the fitness degrees of all occurrences of that rule against this specific case.

Suppose g is an aggregation function, $F(c, r)$ is the fitness degree of case C against rule R , $F_i(C, R)$ is the fitness degree of occurrence i of rule R in case C , k is the number of occurrences of rule R in case C , then

$$F(C, R) = g(F_1(C, R), F_2(C, R), \dots, F_k(C, R)) \quad (4.2)$$

Notice here that the aggregation function is not applied in two scenarios: 0 and 1 occurrence. If there is no occurrence then the result is simply a dash because there is no values to aggregate. If there is only one occurrence then there is no need to run the aggregation function because the result is simply the occurrence fitness degree.

The cell fitness degree can be seen as an intermediate value between the occurrence fitness degree and the other three levels. It is used to compute the fitness degrees at the upper levels, i.e. case, rule, and process. Figure 4.2 shows the relation between the fitness degrees shown in the fitness matrix. The case fitness degree is the aggregation of this case against all rules.

Suppose g is an aggregation function, $F(C)$ is the fitness degree of case C , $F(C, R_i)$ is the fitness degree of case C against rule i then

$$F(C) = g(F(C, r_1), F(C, r_2), \dots, F(C, r_r)) \quad (4.3)$$

Similarly, the rule fitness degree is the aggregation of fitness degrees of all

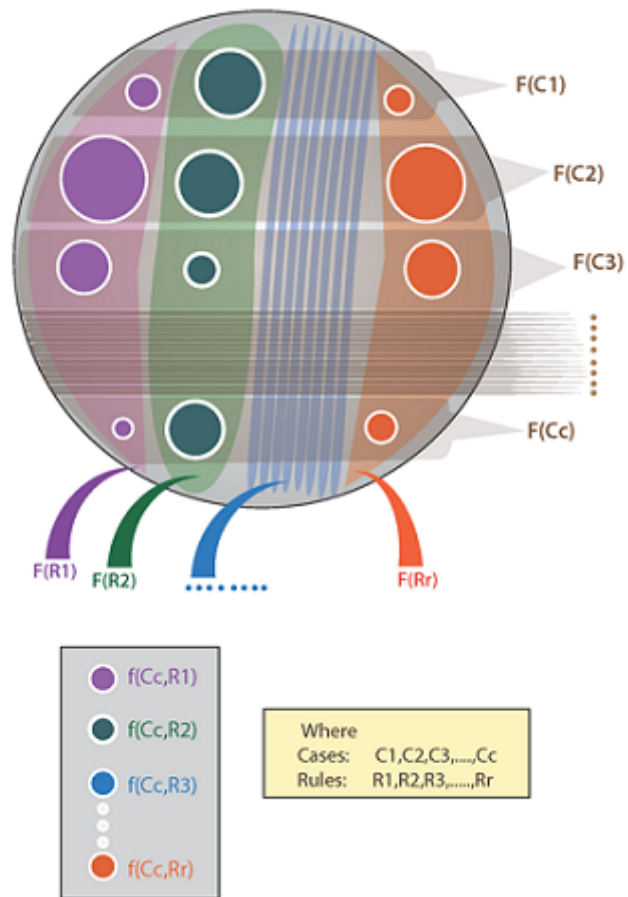


Figure 4.2: Applying aggregation to measure fitness degrees

cases against this specific rule. If g is an aggregation function, $F(R)$ is the fitness degree of all cases against rule R , $F(C_c, R)$ is the fitness degree of case c against rule R then

$$F(R) = g(F(C_1, R), F(C_2, R), \dots, F(C_c, R)) \quad (4.4)$$

The process fitness degrees can be obtained by three ways: either aggregating the fitness degrees of all rules, or aggregating the fitness degrees of all cases, all aggregating the fitness degree of all cells.

Suppose g is an aggregation function, $F(C, R)$ is the fitness degree of case C against rule R , $F(C)$ is the fitness degree of case C , $F(R)$ is the fitness degree against rule R then, the process fitness degree can be measured by aggregating the fitness degrees at the cells level:

$$\begin{aligned} F = &g(F(C_1, R_1), F(C_1, R_2), \dots, F(C_1, R_r)), \\ &F(C_2, R_1), F(C_2, R_2), \dots, F(C_2, R_r)), \dots \\ &F(C_c, R_1), F(C_c, R_2), \dots, F(C_c, R_r)) \end{aligned} \quad (4.5)$$

or aggregating the fitness degrees at the cases level

$$F = g(F(C_1), F(C_2), \dots, F(C_c)) \quad (4.6)$$

or aggregating the fitness degrees at the rules level

$$F = g(F(R_1), F(R_2), \dots, F(R_r)) \quad (4.7)$$

4.3 Aggregation functions

In mathematics, aggregation is the process of replacing a set of values by a single representative one using an aggregation function. Aggregation functions (also called aggregation operators, both terms are used interchangeably in the literature) are used for this purpose. An aggregation function such as the average combines a set of input values into one representing the input values [116].

There exist a variety of aggregation functions such as: arithmetic mean, minimum, maximum, weighted arithmetic mean, etc. Each function has some characteristics which make it suitable for some cases [114–116]. For instance, in multi-criteria decision making problems, the weighted arithmetic mean can be used to

evaluate each alternative. It allows the decision maker assigning different weights to the set of criteria according to their importance. In this research we apply aggregation to combine a set of fitness degrees at low levels to end up with one value representing the fitness degree at higher levels.

Aggregation functions have two fundamental characteristics: the preservation of bounds and monotonicity [114]. The preservation of bounds indicates that the aggregation of n 0s should yield to 0 as well. Similarly, the aggregation of n 1s yields to 1. Suppose $g(x_1, x_2, \dots, x_n)$ is an aggregation function then

$$g(0, 0, \dots, 0) = 0, \quad g(1, 1, \dots, 1) = 1 \quad (4.8)$$

In this research, the preservation of bounds characteristic ensure that if all fitness degrees at lower levels are 100% compliant/non-compliant, then the fitness degree at the upper level(s) will be the same. Suppose we aim at measuring the fitness degree against a rule which is not applied in any of the recorded cases. In such a case, the rule fitness degree is simply 0 because it is never applied. Similarly, if all rules are applied, then the case fitness degree is 1.

The monotonicity property indicates that the function is either of never increasing or of never decreasing as the values of the input increase. Suppose x and y are inputs, $(x_i \leq y_i)$ for all $i \in \{1, \dots, n\}$ then

$$g(x_1, x_2, \dots, x_n) \leq g(y_1, y_2, \dots, y_n) \quad (4.9)$$

Assume we have three rules that are checked in two cases. Case 1 scores $(0, 1, 1)$, Case 2 scores $(0.5, 1, 1)$. That being the case, the fitness degree of the second case is less than the fitness degree of the first one.

In the literature, there exist a variety of aggregation functions. However, the question is: which function(s) should be used in this research? To answer this question we need first to determine the properties which describe the suitable operators to aggregate the fitness degrees for compliance auditing purposes.

Hereafter, g refers to the aggregation function. x, y, z indicates three inputs. $f(x), f(y)$ denotes the output the aggregating x, y, z using function g respectively.

- Compensation (internal): the aggregation function g has the compensation property if its output is a value in the range between the minimum and maximum values of the input.

$$\min(x) \leq f(x) \leq \max(x) \quad (4.10)$$

- Strict monotonicity: an aggregation function is considered strictly monotone increasing if:

$$x \leq y \text{ but } x \neq y \text{ then } g(x) < g(y) \text{ for every } x, y \in [0, 1]^n \quad (4.11)$$

Notice that this part: $x \leq y$ but $x \neq y$ cannot be replaced by $x < y$ because it has a different meaning. $x < y$ indicates that for all components of x and y , $x_i < y_i$. However, $x \leq y$ but $x \neq y$ indicates that at least one component of y is greater than that of x . The later is the requirement in this research.

- Idempotency: an aggregation function is considered idempotent if for every input x :

$$x = (t, t, \dots, t), t \in [0, 1] \text{ the output is } g(x) = t \quad (4.12)$$

- Symmetry: the function f is symmetric if its output does not depend on the permutation of input values. Hence,

$$g(x_1, x_2, \dots, x_n) = g(x_{P(1)}, x_{P(2)}, \dots, x_{P(n)}) \quad (4.13)$$

- Bisymmetry: is an important property when selecting the aggregation function(s) at the process level. Recall here that the process fitness degree can be obtained by aggregating the values at one of three levels; cells, cases or rules. This is called an extended aggregation function in the literature because it is a combination of more than one function, i.e. the process fitness degree can be measured by aggregating the rules fitness degrees which is in turn the aggregation of some other degrees. Having a bisymmetric function ensures that the results are always the same regardless of the level at which the values are aggregated.

If g_{ij} represents the fitness degree of case i against rule j , then the aggregation function is bisymmetric if for all $i, j = 1, 2, \dots$ and for all $x \in [0, 1]_{ij}$

$$\begin{aligned} g_{ij}(x) &= g_i(g_j(x_{11}, \dots, x_{1j}, \dots, g_j(x_{i1}, \dots, x_{ij}))) \\ &= g_j(g_i(x_{11}, \dots, x_{i1}, \dots, g_i(x_{1j}, \dots, x_{ij}))) \end{aligned} \quad (4.14)$$

- Neutral element: the neutral element is an element which does not affect

the result such as zero in addition and 1 in multiplication. Having a neutral element is important in this research to deal with dashes, i.e. cases in which the rule cannot be checked. An aggregation function g has a neutral element $e \in [0, 1]$ if for every $t \in [0, 1]$ in any position it holds

$$g(e, \dots, e, t, e, \dots, e) = t \quad (4.15)$$

The above mentioned points are general properties to measure the fitness degrees at the different level. It is not necessary to have an aggregation function which applies all these behaviors. However, we will select the properties which are suitable to measure each metric individually. For instance, bisymmetry is suitable for one metric only, the process fitness degree, because it is the only one which is extended. On the other hand, having a neutral element is suitable to measure all metrics because in each metric there is a probability of have a dash (unchecked rule).

Existing aggregation functions can be categorized, according to their semantics, into four main categories: averaging, conjunctive, disjunctive, and mixed. An aggregation function g is said to be an averaging function if $\min(x) \leq f(x) \leq \max(x)$. It is used when there a compensation behavior is required. Conjunctive functions are bounded by the minimum value at the upper bound, $f(x) \leq \min(x)$. In contrast, disjunctive functions are bounded by the maximum value at the lower bound, $f(x) \geq \max(x)$. Conjunctive and disjunctive are both required when a reinforcement behavior is required. Mixed functions are simply the functions which cannot be categorized under any of the other three classes. These are a mixed of more than one category.

Conjunctive and disjunctive functions do not fit in our research because they are not bounded by the minimum and maximum values. Both types not allow for compensation behavior which we consider important in this research. Hence, they are discarded in the procedure of selecting the aggregation functions. Mixed functions are discarded as well for the same reason. Hence, our discussion will be limited to averaging functions.

Hereafter we first discuss the properties which should be considered to measure the fitness degree at each level separately. Then, accordingly, we select a set of suitable aggregation functions. Notice that the occurrence fitness metric is out of the discussion here because it is not generated using an aggregation function. However, it is the input for the cell fitness metric.

4.3.1 Cell fitness metric

The cell fitness degree ($F(c, r)$) represents the fitness degree of one case against one rule. It is generated by aggregating the fitness degrees of all occurrences of that rule in this specific case.

As an aggregation operator we select the ordered weighted average (OWA) function because it can represent a set of aggregation operators which are suitable for this approach. OWA was first introduced in 1988 by Ronald R. Yager to be used in the fuzzy sets field. It can be seen as the abstract for a set of well-known aggregation operators including the *maximum*, *minimum*, *arithmetic mean*, *weighted arithmetic mean*, *the k-order statistics*, etc. [114]. The OWA general formula is given as the following:

$$OWA(x_1, x_2, \dots, x_n) = \sum_{i=1}^n w_i x_{\sigma(i)} \quad (4.16)$$

where w_i is between $[0,1]$ and the summation of weights is equal to 1. σ is a function which orders the values to be aggregated before applying the formula. In the equation $x_{\sigma(i)}$ denotes the value at the position i after the input values are ordered.

In order to represent one of the operators, we need first to customize the σ function and then to assign the weights accordingly. The σ function is defined according to the function we want to present. For instance, we can use the non-increasing function or the non-decreasing function to represent the operators: maximum, minimum, arithmetic mean, median and the k-order statistics. Applying any of these two functions (i.e non-increasing or non-decreasing functions) will produce a set of ordered elements. In the next step we can assign the weights with respect to the ordering function. For instance, if the non-decreasing function is used to represent the maximum operator, the result will be a list of ordered values in which the maximum value is at the last position. That being the case, the value in the last position is assigned the weight of 1 whereas all other values are assigned the weight of zero. Figure 4.3 shows the weights corresponding to each operator if the non-decreasing function is used.

As mentioned earlier, the OWA function is an abstract for a set of well-known aggregation functions. Among these function we select the: *minimum*, *arithmetic mean*, *weighted arithmetic mean*, and *median* since they are appropriate in our context.

The OWA operator has the following property: monotonic, idempotent, and has

	OWA
Minimum	$\begin{cases} w_1 = 1 \\ w_i = 0 & \text{if } i \neq 1 \end{cases}$
Maximum	$\begin{cases} w_n = 1 \\ w_i = 0 & \text{if } i \neq n \end{cases}$
Median	$\begin{cases} w_{\frac{n+1}{2}} = 1 & \text{if } n \text{ is odd} \\ w_{\frac{n}{2}} = \frac{1}{2} \text{ and } w_{\frac{n}{2}+1} = \frac{1}{2} & \text{if } n \text{ is even} \\ w_i = 0 & \text{else.} \end{cases}$
k-order statistics	$\begin{cases} w_k = 1 \\ w_i = 0 & \text{if } i \neq k \end{cases}$
Arithmetic mean	$w_i = \frac{1}{n} \quad \text{for } \forall i$

Figure 4.3: OWA weights for special cases

a compensation behavior. The later ensure that its result is bounded by the min and max value of the input. However, it does not have a neutral number.

Minimum function:

the *Min* function is an extreme! By using it the process manager indicates that a single violation is enough to identify the cell as a completely non-compliant case against the given rule. Hence, it represents a very strict interpretation. This function should be used when it is important to be stuck to the business rule. For instance, when the process is very critical and any deviation may cause dangerous consequences. Or that the manager tends to use an old managerial approach in which managers do not believe in the flexibility of execution. Normally, this type of managers would prefer to be stuck to the given prescribed process (this is not common these days)!

Notice here that the *Maximum* function represents the other extreme. However, it is inappropriate in our context. The maximum function indicates that a single compliant occurrence is enough to consider the cell as fully compliant regardless of the number of violating occurrences. However, this does not match the compliance auditing definition. Assume a case in which a given rule is checked 100 times and that it was applied only once out of the 100 times. In this case, the cell fitness degree will be considered as completely compliant if the maximum function is used!

Median function:

The median value is the value separating the higher half of the data from the lower half. Using either the ascending or descending order as a σ function, the value at the center is returned. If the number of input values is odd then there will be one value at the center. Otherwise, there will be two values at the center. That being the case, the arithmetic mean of the two values is the result.

The Median is suitable to use when the process manager has a moderate approach to interpret the compliance checking result. Using the median operator, the cell is completely compliant if the majority of occurrences are compliant and non-compliant otherwise. Suppose a case in which there are seven occurrences and that the fitness degrees of the seven occurrences are (in order): 0, 1, 1, 0, 0, 1, 0. Using the median operator, the values are first ordered either ascendingly or descendingly. The result, using an ascending order, is: 0, 0, 0, 0, 1, 1, 1. Then, the value in the center is selected as the output of the function. However, in case

there is an even number of input values, then the result is the average of the two values in the center after the input values are ordered.

Arithmetic mean function:

The arithmetic mean (AM) is a member of the simple means family which contains the geometric and harmonic means as well. It is an additive operator which is used to average a set of abstract values. Thus, it is suitable for the purpose of this research. Usually, we refer to the arithmetic mean as the average in our daily life. The arithmetic mean formula for a set of n values is:

$$AM = \frac{1}{n} \sum_{i=1}^n x(i) \quad (4.17)$$

AM is yet another moderate approach. It can be used when input values are considered to have equal contribution to the output value. However, this is not usually the case when checking the process compliance. For instance, if the input is 0, 1, 1, 0, 0, 1, 0 then the fitness degree using the AM is 0.43.

OWA function:

Using AM, occurrences are considered to have equal contribution to the fitness degree. However, in some cases, process managers need to assign different weights. For instance, suppose the process manager has the following semantics: if the rule is violated three times out of 5 then the cell is completely non-compliant. Otherwise, less violations can be compensated to some extent by the compliant occurrences. To achieve this semantics, we use the OWA function in which the process manager provides a specific weight vector to each occurrence.

To measure the fitness degree using OWA operator, the input values are first ordered ascendingly. Then the first m occurrences are assigned weights equal $1/m$ where m is the number of violating occurrences (3 in our example). Accordingly, if the number of violating occurrence is ≥ 3 then the cell is completely non-compliant. Otherwise, if all occurrences are compliant then the cell is completely compliant. Otherwise, the cell fitness degree is compensated to some extent by the compliant occurrences. For instance, if $m = 2$ then the cell fitness degree will be 0.33

The value of m is provided by the process manager. However, the value of k is not fixed. The process manager does not know the value of k to provide a suitable value for m . For instance, $m = 5$ is relatively high when $k = 4$. However, it is

relatively very small when $k = 200$. Moreover, what if the entered value for m is less than the value of k . To overcome this issue, we define a new parameter called the Rel_v . It represents the percentage of violating cases to the overall number of occurrences. Rel_v stands for the relative number of violating occurrences. It is simply, m/k which is 60% in our example. Having the value of Rel_v and k which is generated automatically, $m = Rel_v * k$.

4.3.2 Case fitness metric

This is the aggregation of the fitness degrees of one case against all rules (cells in one row in the fitness matrix). The case fitness metric is important to study one specific case. Normally it is used to measure the fitness degrees of suspected cases. As mentioned earlier, dashes are not included in the aggregation procedure. Hence, if the fitness degree of a given case is a dash then it could not be checked against any of the business rules. However, this is a weak assumption.

To measure the case fitness degree we define: Minimum, Median, AM, Hereafter, we discuss the semantics of each operator and their description. However, we skip the discussion of the operators that are already discussed in the cell fitness metric section, i.e Min and AM.

Minimum function:

as used earlier, the minimum represents an extreme. It can be used just in case all rules should be applied strictly to accept the case as a compliant one. Any violated rule is enough to penalize the case and reject it in the compliance context.

Arithmetic mean function:

The arithmetic mean (AM) represents a moderate approach to measure the case fitness degree. It is the most suitable operator when the checked rules have equal contribution in the fitness degree. It should be used when the process manager has a moderate clue with no preferences. Hence, it is defined as the default operator, i.e. in case the process manager did not select an operator then this one is used.

WAM:

unlike the AM, in which rules are assigned equal weights, the WAM allows the process manager to assign different weights. Each business rule is assigned a

weight showing its importance in the aggregated value.

$$WAM = \sum_{i=1}^r (F(C, R_i) * w_i) \quad (4.18)$$

Where, r is the number of business rules, $F(C, R_i)$ is the fitness degree of case under investigation against rule i , w_i is the weight assigned to rule i . Notice that the AM is a special case of the WAM where input values are assigned equal weights.

The WAM is the most appropriate operator to use when some rules are important than the others. Suppose the normative model contains five business rules and that one of them is more important than all other rules whereas the others are almost equally important. That being the case, the process manager can assign 60% of the weights using the WAM operator and distribute the remaining 40% among the remaining rules. If a case can be checked against all these rules (no dashes), then the cell fitness degree corresponding to the 60% weighted rule has the highest contribution on the fitness degree of this case.

In WAM, the summation of weights assigned to rules should be equal to one. However, what if there are some rules which cannot be checked, i.e. a dash in the input. In this case, the rule should not be considered in the calculation. To solve this, we define a flexible weighting function which can deal with the any scenario. First, the user should enter a value between 0 to 10 for each business rule to show the importance of this rule. We call this value the importance degree. A 0 importance degree indicates the least importance whereas a 10 indicates the most importance. Once the importance' degrees are determined, the weighting function is used to assign the weight for each rule automatically taking into consideration the dashes in the aggregated values.

$$W_{R_i} = \frac{\text{importance degree of } R_i}{\sum \text{importance degree of all checked rules}} \quad (4.19)$$

Assume the normative model contains four business rules, the importance' degrees are (in order): 5, 8, 3, 10, the case under investigation has the cells fitness degrees: 1, 0.5, -, 1) against each of the four rules in order. Notice that the case is not checked against rule number 3. Given these values, the weights will be: 1/23, 8/23, -, 10/23 where 23 is the summation of the importance' degrees (5, 8, 10) of all checked rules (R_1, R_2, R_4).

Add Chouquet integrals ...

4.3.3 Rule fitness metric

The rule fitness degree is the aggregation of the fitness degrees of all cases against this rule. In the fitness matrix it is the aggregation of the cells values in one column. To measure the rule fitness degrees we use: minimum, AM and WAM functions. Hereafter we discuss the semantics of each aggregation function. The description of the functions are not given here because it is already discussed earlier in the cell fitness degree point. However, we discuss the weighting function which is used to assign weights in the WAM function. As mentioned earlier, when measuring the rule fitness degree dashes are not considered regardless of the aggregating function used. It could happen that the fitness degree of one rule is a dash. Although this is a weak assumption, however, if it happens then it indicates that the normative model and the compliance requirements should be re-engineered.

Minimum function:

This function is used when the rule should be applied very strictly to consider it as a completely compliant one. Any violating case is penalized heavily. *AM*: this function is used for a moderate semantics. Thus, we select it as the default function.

WAM:

The WAM function is used when cases do not contribute equally to the rule fitness degree. The weights are assigned according to the cases attributes. For instance, in the procurement process, cases in which the PO value is high are more important than those with low PO value. The process manager should select the attribute which control the weighting function. For the moment, only one attribute is considered. Intuitively, to be able to assign a weight to a specific case, it should hold a value for this attribute. Cases in which it does not appear are treated as less important cases. Notice that this does not apply to cases with a dash value.

Weighting function depends on the data type of the selected attribute, i.e. numerical, nominal, date, etc. If the attribute has a numerical value then, the user enters some thresholds which can be used in the weighting functions. For instance, if the user defines two thresholds: 1000 and 2500. Then the cases which could be checked are categorized into three groups: cases in which the selected attribute value is less than 1000, cases in which it is between 1000 and 2500 and cases in which the attribute value is above 2500. Hence, there are three weights. The

groups are ranked according to their importance, i.e. the least important group gets the rank 1, then the second least important is ranked by 2 and so on. Once the groups are ranked, each group is assigned a weight equal to its rank over the summation of ranks.

$$W_{group_i} = \frac{rank_i}{\sum_{i=1}^{\# \text{ of groups}} i} \quad (4.20)$$

In our example, suppose that the user consider the first group as the least important, then the three groups are assigned the weights (in order): 1/6, 2/6, 3/6.

In case the attribute contains a nominal data, things become easier. Unique values are grouped according to their importance. The process manager can categorize two or more values to the same group to give them equal weights. In addition, values that are not categorized by the process manager are grouped together automatically and are assigned the least rank. Later on, the weights are assigned according to the above mentioned formula.

Attributes holding date values are treated in a way similar to that used for numerical values. The process manager should provide some thresholds to rank the values. However, in date values the comparison condition is before and after instead of greater than and less than. After ranking the cases, they are assigned weights in the same manner.

Regardless of the data type, just in case the selected attribute does not appear in some checked cases then these cases are categorized in a new group and will be assigned the least weight. In the above example, suppose that some cases do not have this attribute then the weights are shifted one step. New weights are: 1/10, 2/10, 3/10, 4/10, where the first group is the one in which the attribute does not appear.

Notice that normally the selected attribute is related to the business rule whose fitness degree is measured. Hence, having cases in which the attribute does not appear is a weak scenario. However, we consider it her to keep the technique as flexible as possible.

4.3.4 Process fitness metric

If the aggregation function is a bisymmetric one, then there are three approaches to measure this degree: either aggregating the fitness degrees at the cells level, or aggregating the fitness degrees of cases, or aggregating the fitness degrees of

rules.

Among the above mentioned functions, there are only two functions which can be considered: minimum and AM functions. However, the minimum function is not suitable to measure this degree. Suppose 1000 cases are checked against 5 rules and that all rules could be checked in all cases. This generates 5000 fitness degrees at the cells level. Lets assume that all these values are 1 (completely compliant) except one cell in which the result is zero. That being the case the process is almost 100% compliant. However, using the minimum function the result will be zero. So, the whole process is completely violated just because the result of one cell is zero. Hence, the minimum function is not used to measure this degree.

However, what is the need for three approaches if the result is always the same? Actually, the first approach, i.e. aggregating the values at the cells level, is the default one. However, the other two approaches are defined to allow the process manager using the WAM function. Suppose the process manager aims at measuring the process fitness degree taking into consideration the weights of the business rules. In this case, he can measure the fitness degree of cases using the WAM function and then measure the process fitness degree. Similarly, if the recorded cases are not equally important and the process manger aims at measuring the process fitness degree taking into consideration the different contribution of each case then the WAM can be used at the rules level and then another fitness degree to measure the process level fitness degree.

4.4 Example

Hereafter we introduce a simple example to illustrate how the aggregation is applied. Assume that the business process under investigation is the procurement process and that it is a collection of seven activities; 'Create PO', 'Sign', 'Release', 'Invoice receive', 'Goods receive', 'Pay', and 'Change line'. For the sake of simplicity we will use symbols instead of the real activities' names. So: A represents the event 'Create PO', B for 'Sign', C for 'Release', D for 'Invoice receive', E for 'Goods receive', F for 'Pay', and G for 'Change line'. Given a log file of 10 cases and a set of five business rules, we aim to fill out the fitness matrix shown in Table 4.1) as presented earlier in this chapter. The traces of the cases recorded in the log file are (in order):

1. $\langle A, B, C, G, E, D, F \rangle$

2. $\langle A, B, C, E, D, F, F, F \rangle$
3. $\langle G, A, B, C, D, F \rangle$
4. $\langle A, B, C, G, E, G, E, E, G, B, C, E, D, F, F, F \rangle$
5. $\langle F, A, B, C, G, D, E \rangle$
6. $\langle A, B, C, E, G, B, G, B, C, G, B, C, E, E, E, D, D, F \rangle$
7. $\langle A, B, C, G, D, F \rangle$
8. $\langle A, B, G, C, G, E, D, D, F, F \rangle$
9. $\langle A, G, B, C, D, F \rangle$
10. $\langle A, C, D, F \rangle$

The five business rules holding the compliance requirements are:

Table 4.2: The business rules representing the compliance requirements

Rule	Description
R1	A should be the first activity
R2	B should be followed directly by C
R3	C should be followed eventually by F
R4	SOD (A, E)
R5	If G.modification_value > 1,000 then G should be followed eventually by B

The first three rules represent sequencing requirements. The fourth one represents an HR requirement and the last one represents a data requirement. To check the fourth rule we need to know the performers of events A and E for each individual case. Similarly, we need the modification values to check the last rule.

Modification value mentioned in R5 is an attribute for activity G. It indicates to which extent the PO value has been changed. According to the rule, if this value is above a threshold of 1,000 Euro the purchase order should be signed before the case is closed. The information required about performers and modification values are given in table 4.3.

Below we consider two different scenarios and select the suitable aggregation functions for each scenario.

Scenario A

The process manager has a moderate clue. He aims at measuring the fitness degree using a moderate function in which all cells have equal contribution to the fitness degree.

That being the case, the AM is the best to use. It provides a moderate semantic and equal weights. Table 4.4 shows the fitness degrees at the cells level. The values in the table are the result of using the AM function to aggregate the values

Table 4.3: Originators and modification values required to check the fitness

T	Originators
1	$\langle A(OrigX), B, C, G, E(OrigY), D, F \rangle$
2	$\langle A(OrigX), B, C, F, E(OrigZ), D, C \rangle$
3	$\langle G, A(OrigX), B, B, C, D, F \rangle$
4	$\langle A(OrigX), B, C, G, E(OrigX), G, E(OrigZ), E(OrigX), G, B, C, E(OrigW), D, F \rangle$
5	$\langle F, A(OrigX), B, C, G, D, E(OrigY) \rangle$
6	$\langle A(OrigX), B, C, E(OrigY), G, B, G, B, C, G, B, C, E(OrigZ), E(OrigX), E(OrigW), D, D, F \rangle$
7	$\langle A(OrigX), B, C, G, D, F, C, C \rangle$
8	$\langle A(OrigX), B, G, C, G, E(OrigX), D, D, F, F \rangle$
9	$\langle A(OrigX), G, B, C, D, F \rangle$
10	$\langle A(OrigX), C, D, F \rangle$
T	Modification values
1	$\langle A, B, C, G(mod. = 200), E, D, F \rangle$
2	$\langle A, B, C, F, E, D, C \rangle$
3	$\langle G(mod. = 0), A, B, B, C, D, F \rangle$
4	$\langle A, B, C, G(mod. = 0), E, G(mod. = 1500), E, E, G(mod. = 0), B, C, E, D, F \rangle$
5	$\langle F, A, B, C, G(mod. = 1750), D, E \rangle$
6	$\langle A, B, C, E, G(mod. = 0), B, G(mod. = 7560), B, C, G(mod. = 1354), B, C, E, E, E, D, D, F \rangle$
7	$\langle A, B, C, G(mod. = 5163), D, F, C, C \rangle$
8	$\langle A, B, G(mod. = 0), C, G(mod. = 0), E, D, D, F, F \rangle$
9	$\langle A, G(mod. = 1100), B, C, D, F \rangle$
10	$\langle A, C, D, F \rangle$

Table 4.4: Fitness matrix for scenario A

	R1	R2	R3	R4	R5	Fitness _(Cases)
Case ₁	1.00	1.00	1.00	1.00	-	1.00
Case ₂	1.00	1.00	0.50	1.00	-	0.88
Case ₃	0.00	0.50	1.00	-	-	0.50
Case ₄	1.00	1.00	1.00	0.00	1.00	0.80
Case ₅	0.00	1.00	0.00	1.00	0.00	0.40
Case ₆	1.00	0.66	1.00	1.00	1.00	0.93
Case ₇	1.00	1.00	0.33	-	0.00	0.58
Case ₈	1.00	0.00	1.00	0.00	-	0.50
Case ₉	1.00	1.00	1.00	-	1.00	1.00
Case ₁₀	1.00	-	1.00	-	0.00	0.25
Fitness _(Rules)	0.80	0.80	0.78	0.67	0.50	0.73

at the occurrence level. Starting from these values, the fitness degrees at the other three levels are measured using some aggregation functions.

Cases (1 and 9) are completely fit. Case 6 comes on the second place. It is completely fit against all rules except for rule 2. Case 5 scores the least fitness degree. It violates three rules out of the five. Rule 5 could not be checked in 4 cases and the fitness degree against this specific rule is 0.50 which is lower than the fitness degree against all other rules. Notice that all fitness degrees against rule 1 are either 0 or 1 because the rule can be checked only once in each case. This is also the case against rule 4 as well. The rule represents an HR compliance requirements. Thus, only the first occurrence is checked so that the values are either 1 or 0. On the other hand rules 2 and 3 are sequencing rules so are checked as much as they occur.

Scenario B

The process manager aims at measuring the cases fitness degree. However, he/she considers R3 as the most important business rule and then R4. Rules 2 and 5 are equally important. R1 is the least important one.

Given this scenario, we can use the WAM operators to measure the cases fitness degrees. According to the process manager, the rules are assigned the following degrees of importance: 1, 2, 4, 3, 2 to the five rules in order. Notice that rules 2 and 5 are assigned equal values. Accordingly, if all rules are checked, the weights would be: 1/12, 2/12, 4/12, 3/12, 2/12. Using these weights, the result will be as shown in table 4.5.

Table 4.5: Fitness degrees at the case level for the scenario B

	Cases fitness degrees
Case ₁	1.00
Case ₂	0.80
Case ₃	0.71
Case ₄	0.75
Case ₅	0.41
Case ₆	0.94
Case ₇	0.48
Case ₈	0.50
Case ₉	1.00
Case ₁₀	0.71

4.5 Conclusion

In this chapter we have introduced the Flexible Compliance Auditing Framework to support the business world with more applicable metrics. The framework provides different semantics for a compliant process. Hence, it can be used to check the compliance taking into consideration the business context in which it is applied. A set of fitness metrics are defined to measure the fitness degree at different levels of abstraction. In addition, the concept of "not checked rules" is introduced to the field of compliance auditing. We show why it is important to distinguish them from the remaining cases.

Chapter 5

The Loan Approval Process: A Synthesized Data Set

To evaluate the proposed technique we use a real log file and a synthesized log file. In this chapter we discuss the results obtained by running the implemented tool on the synthesized log file. The process of generating the log file and defining the business rules is discussed in section 1. Results are discussed in section 2 in which the results of each metric is discussed separately. The description of the implemented tool is given in **Chapter 9**.

5.1 Generating the log file and defining the business rules

To evaluate the proposed technique, a synthesized log file is generated using the *the SecSy tool*. The file is generated for the loan approval process which is executed daily in banks and other lending organizations. When selecting the business process, we aim at using a common process which is easy to understand and define business rules for. Figure 5.1 below shows the BPMN model representing the sequence of activities to perform the process.

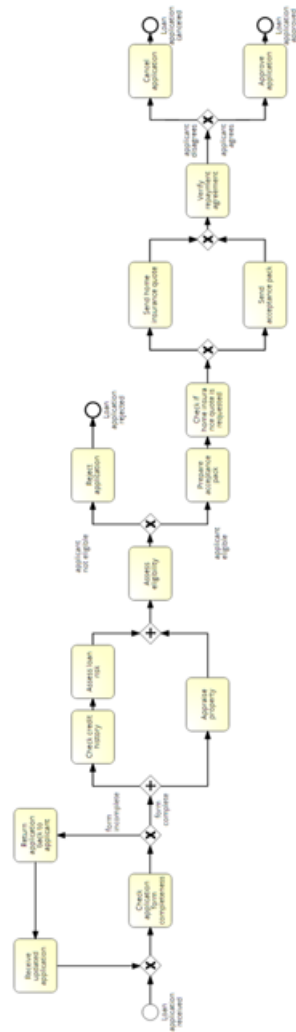


Figure 5.1: Loan approval process model using the BPMN notations

Once a loan application is received, a new process instance is triggered. The staff in the lending department works on checking the application form completeness. If there is any missing information then the form is returned back to the client until the form is completed. Next, the property is appraised in parallel with checking the history of the client credit and assessing the loan risk. The risk analyst is responsible for assessing the loan risk. The estimated value of the property in addition to the loan risk analysis are forwarded to the eligibility analyst who is responsible for assessing the eligibility. Accordingly, the application is either accepted or rejected. If it is rejected then the process is terminated. If it is accepted then an acceptance pack is prepared and the home insurance quote is checked to continue the process and verify the payment agreement. Next, it is the applicant decision to agree or disagree. If he/she agrees then the loan is approved and the process is terminated. Otherwise, the process is terminated with a canceled application.

To generate the log file, we started with a synthesized data set from the 3TU.Datacentrum website [117]. The selected data is suitable to check sequencing and temporal requirements. However, it cannot be used to check the human resources and data requirements. This is because the log file does not contain the information required to check these two types of requirements. Hence, we use the *Alpha Miner* plugin [118] under ProM 6.5 [119] to build a Petri net representing the data set. In a next step, we use the mined model to generate the required data set using the SecSy tool. Alpha miner is used because it generates a Petri-net model which is the required input format for SecSy.

SecSy is a log generator tool [120]. It is based on simulating the control flow of a business process using a Petri net model. SecSy can be used to define some originators and attributes and attach them to the generated events. It can also be used to include detailed timing properties such as activity durations and delays between activities. The most important feature of the SecSy tool is the ability to consider compliance requirements. For instance the tool can be configured to generate cases which deviate a given business rule such as the segregation of duties and delay of events.

The generated log file contains 1000 cases with a total of 10662 events. The originators defined are: loan department manager, risk analyst, eligibility analyst and loan department staff. The attributes defined are shown in table 5.1 with their corresponding data.

The log file is generated in two steps: first the process is simulated to generate

Table 5.1: Attributes defined for the loan approval log file

	Attribute	Type	Activity	Value
1	Loan type	String	Loan application received	housing, car, personnel
2	Client group	String	Loan application received	G1, G2, G3, G4
3	Property value	Numeric	Apprise Property	
4	Risk level	String	Assess risk	low, medium, high
5	Number of points	Numeric	Assess eligibility	
6	Loan value	Numeric	Verify payment	
7	Payback period	Numeric	Verify payment	

the required number of cases. In this step, events are attached with the defined data according to the requirements. Next the modifiers are applied to modify the generated cases in a way that creates some non-compliant cases. In our case we used two types of modifiers: unauthorized execution and SOD. Unauthorized execution modifier works on the originator field. It adjusts the originator in a way that violates some HR requirements. For instance if an event should be executed by an originator X then this modifier changes the value X to have a non-compliant case. The SOD modifier is used to create cases that violates segregation of duties requirement.

The log file generated using the SecSy tool does not contain cases violating sequencing requirements. So, we run the *Add Noise* plug-in [reference] under ProM 6.5 framework to have some violating cases for the purpose of this research. The plug-in reads two parameters; the percentage of adding events and the percentage of removing events. Both parameters are adjusted to 5%.

To formalize the set of business rules which will be checked, we took into consideration the process model provided to the log generating tool and the data attached to the events. Accordingly, we define five business rules: three sequencing requirements and two HR requirements. It was difficult to check the other two types because of the limitations of the log generating tool. The defined rules are:

- R1: Prepare acceptance pack should be followed directly by Check if home insurance quote is requested
- R2: Check application form completeness should be followed eventually by Assess eligibility
- R3: Check application form completeness should be the first activity
- R4: SOD(Assess loan risk, Assess eligibility)
- R5: Approve application should be executed by the Loan department manager

According to the process model, Rule 1 is checked whenever the application is not rejected. The activity 'Prepare acceptance pack' appears in 476 cases out of the 1000 cases forming the entire log. Since it detected 490 times in the entire log then it should have been performed more than once in some cases. The activity 'Check application form completeness' appears 1564 times in the log file, however, it does not appear in 24 cases although it should be always the first activity according to Rule 3!

Rationally, Rule 3 can be checked only once in each case. The result is either one or zero, no other possibilities. This is because there is always a first activity for each cases that is either 'Check application form completeness' or not. Rule 4 is one of the most common business rules for HR compliance requirements. The rule implies that the activities 'Assess loan risk' and 'Assess eligibility' should not be performed by the same person. According to the results, the rule is checked in all cases.

Rule 5 is checked, normally, as much as there are approved applications. According to the process model, the process is terminated in one of three ways; either the application is approved, rejected, or canceled by the client. Apparently, 238 (1000-762) applications are just approved. When generating the log file, we assume that this is the most important rule that should be strictly applied.

Notice here that the number of occurrences does not reflect the number of cases in which the rule is checked. It could be the case that the rule is checked different times in the one case but is never checked in another case. In our example this happened in Rules 1, 2 and 5.

5.2 Result analysis

The results are studied at the different levels starting by the most abstract level (the log file level) and moving down to the most detailed level (one case against one rule level). The tool is run using the different aggregation operators except for the median since it is not implemented yet.

5.2.1 The process level fitness degree

According to our definition, the process fitness degree is measured by aggregating the fitness degrees of: cells, cases, or rules. Regardless of the aggregated values, the AM operator is used to end up with one value representing the fitness degree of

the entire log. Table 5.2 shows the fitness degrees using different aggregation operators. The operator used at the cells level is given in the first column. The other columns show the fitness degrees measured by aggregating the fitness degrees of cells, rules and cases in order. The fitness degree of each rule (third column) and case (last column) is measured using the AM operator. The reason behind using this operator among the others is to avoid the bias towards any of the two edges.

Table 5.2: The fitness degree at the process level

Aggregation operator for cells	AM of cells	AM for rules	AM for traces
Min	92.74%	92.24%	92.42%
AM	93.33%	92.83%	92.97%
WAM	93.22%	92.75%	92.86%

Results show that the majority of cases are compliant. As expected, results are bounded between the Min and the Max results. AM and WAM outputs are in between. Notice that the difference between the results of the different operators is relatively small. This is because the input values are very close. For instance, there is only 39 cases making the difference between the results of the different operators.

The process fitness degree is a good indicator to measure the overall performance. Nevertheless, it cannot be used to check more details such as which cases are completely compliant, non-compliant, partially compliant? Which rule is the most applied one and which one is most violated one? How many times a given rule is checked in a specific case? What if some rules are more important than others? To answer such questions, we use the tool to check the fitness degree one level lower. At this level, two measures are provided; cases and rules.

5.2.2 Rule level fitness degree

This measure is of high importance for process managers. It helps them measuring the fitness degree against each compliance requirement separately. Table 5.3 shows some basic statistics about the results using the AM operator for both cells and rules. A general look shows that the different rules have close fitness degrees (range [89% - 95%]). However, this is not necessarily true! The fitness degree is affected by the number of aggregated values, i.e. how many times each rule is checked. For instance, comparing the results of rules 4 and 5, the difference between the two values is less than 1%. However, Rule 4 is checked in all cases

(100% of the log file) whereas Rule 5 is checked in only 238 cases. Mathematically, using one of the averaging operators, the fitness degree against one specific rule is affected by the number of cases that are checked. More checked cases means less contribution of each case on the final result. For instance, in our example, the fitness degree of each case used to calculate the fitness degree of Rule 4 affects the final result by (+/-) 0.0010 whereas each trace used to calculate the fitness degree of Rule 5 affects the final result by (+/-) 0.0042!

Table 5.3: Rules fitness degrees: statistical information

	R1	R2	R3	R4	R5
Fitness degree	89.81%	94.08%	94.50%	93.30%	92.44%
No. of cases with Fitness degree = 1	421	903	945	933	220
No. of cases with Fitness degree = 0	42	47	55	67	18
No. of cases with Fitness degree = (0-1)	13	26	0	0	0
No of cases which are not checked	524	24	0	0	762

Table 5.4 shows the fitness degree against each of the five rules using the different operators. Given that the fitness degree is the aggregation of the fitness degrees of individual cells (i.e. one case against one rule), the operator used to compute the fitness degree of cells is important. Recall here that the proposed technique provides three aggregation operators at the cell level; Min, AM, and WAM. In this section, we use the AM and WAM operators. The other operators are discarded because they bias the result to one of the two edges. For this case, there is a very small deviation between the results of AM and WAM (less than 1%) except for one value, Rule 2. The result is 95% using the AM to measure the cells fitness degrees and 94% using the WAM operator. The reason is that the number of occurrences in this data is relatively small compared to other log files (at most 7 occurrences). According to the result, Rule 3 is the most applied one. On the other hand, Rule 1 is the most deviated one.

To prove our idea about discarding cases which cannot be checked, we repeat the calculation if these cases are encountered. The results of rules 3 and 4 are not affected because all cases are checked. However, the results of rules 1 and 5 are totally different! If these cases are considered non-compliant, then the result of Rule 1 is 52% instead of 90% and only 27% for Rule 5 instead of 92%. The result of Rule 2 is a little bit less 93%. This is because the number of non-checked cases is relatively smaller than those of rules 1 and 5. This shows the importance of not considering these cases as compliant or non-compliant.

Table 5.4: Rules fitness degrees using the different aggregation operators

	R1	R2	R3	R4	R5
Min	0.00	0.00	0.00	0.00	0.00
AM	0.90	0.94	0.95	0.93	0.92
WAM	0.90	WAM 0.94	0.95	0.94	0.92
Percentage of cases checked	47.6	97.6%	100%	100%	23.8%

Regardless of the operator used, results can help managers making important decisions such as re-engineering the process model, i.e the business rules, in the future. Process re-engineering is out of the scope of this research. However, the output of our technique is important to make such decisions. A rule that is never applied or checked in the entire log needs to be revised to make sure it represents the required performance accurately. In this example all rules are checked at least once. None of the rules is completely applied or completely violated. However, some of them are checked in the entire log, whereas others are checked in less than one fourth of it. Rule 1 is checked just in case the application is not rejected. Rule 5 is checked just in case the application is approved.

Rule 2 represents a requirement that should be checked in each case. However, there are 24 instances in which the rule is not checked. If it is a real data then process managers can check these cases in more details to detect the problem. It could be that the activity "Checking the application form completeness" is executed but simply not logged or maybe not executed. In this example, if it is not executed then it may affect the approval decision made by the loan department because missing information can be considered as negative point against the applicant. Otherwise, the process instance may take more time to be executed. It is also possible that some applicants have enough experience about the procedure, and hence, it is not required to check their applications' forms. Unfortunately, studying these cases is not possible since this is a synthesized log. Notice that this activity is common between rules 2 and 3. According to Rule 3, this activity should be executed as first state in all cases which indicates the importance of executing this specific activity. According to Rule 2 this activity should be followed eventually by the assess eligibility activity.

Starting by the Min operator, using this operator we assume that the rule should be followed strictly, one non-compliant case is enough to say that this rule is violated. This indicates that each rule is completely violated at least once. According to this result, the process manager can go through the aggregated values to check

which cases are completely violated. In the case study, we assume that Rule 5 should be followed strictly. Thus, this operator is suitable to be used to check the fitness degree against this rule. That being the case, Rule 5 is considered completely violated. According to the result of the AM operator, the application is not approved by the correct person in 8% of the approved applications. These cases can be checked in more details when going one level down, i.e. the cells level.

We compare our results with the LTL checker [80]. The technique is used to check the compliance of a log file against a set of linear temporal rules. In LTL checker, business rules are categorized as either satisfied (completely applied) or unsatisfied (completely violated). A partial satisfaction is not an option. Given the generated log file and the set of business rules for this case study, all rules are categorized as unsatisfied rules. All rules are considered unsatisfied because none of them is applied 100% in all cases. Moreover, it does not differentiate between violating cases and not-checked cases. The cases in which the rule cannot be checked is considered as non-compliant. The technique does not provide a fitness measure but a coverage rate showing the percentage of cases in which the rule is applied to the total number of cases in the log file. In this case study, the coverage rate of the five rules are; 43%, 92%, 94%, 94%, 22% in order. However, the definition of the SOD rule (Rule 4) in the LTL checker is not the same as the one defined in this work. In the LTL checker, the rule used is (exists person doing tasks A and B). The rule is considered satisfied if there is a person doing the two tasks. In contrast, and with respect to the business rules agreed on with the process manager, we consider this as a violation in this research.

5.2.3 Case level fitness degree

The fitness degree of case C_i is generated by aggregating the fitness degrees of this specific case against all business rules. In our example, there are at most five values to be aggregated (five rules). Since there are no dashes in the result, then all cases are checked at least once. A first look indicates that almost 76% of the recorded cases completely fit the five rules, i.e. their fitness degrees are equal to 1. On the other hand, there are only two cases which are completely violated (fitness = 0). The rest are partially compliant. By partially compliant here we refer to the cases in which some rules are applied and some are not. The fitness degree of these cases depends on the aggregation operator used. For instance, using the Min operator these cases are considered non-compliant. However, using the averaging operators, the same cases are considered partially compliant, i.e.

score fitness degrees between 0 and 1.

We had a closer look at the completely violating cases (207 & 995). We noticed that both cases violate only two rules: 3 and 4 whereas the other rules could not be checked. This proves our point that one fitness degree does not necessarily reflect the compliance situation accurately. For a process analyst there is a difference between a case which is completely non-compliant and a case in which only 40% of the rules can be checked. In the loan approval process and given that there are three paths to terminate the process, it is very normal that some rules, such as Rule 5, cannot be checked. We have discussed this earlier in section 5.1.

To reveal the importance of discarding dashes (not checked cases) before measuring the fitness degree of each individual case, we recalculate the fitness degrees of the two non-compliant cases another two times with different inputs. In the first time, the dashes are replaced by ones based on the debate that the three rules are not violated. That being the case, the new fitness degree becomes 60% for each case. Accordingly, the two cases are as compliant as case 719 in which three rules are applied and two rules are violated. In the second time, dashes are replaced by zeros based on the debate that the three rules (1, 2, and 5) are not applied. Although this does not affect the fitness degrees of the two cases. However, it will lead to a totally different interpretation as follows:

- Rule 1: it seems as if the "Prepare acceptance back" activity is executed but it is not followed directly by the activity "Check if home insurance quote is requested".
- Rule 2: it seems as if the "Check application form completeness" activity is executed but is not followed by the "Assess eligibility" activity.
- it seems as if the application is approved but by the wrong person.

Notice here that none of these three points are true. Saying this, we assume that all cases will go through the same execution path. However, this is not true in our case study. The process is terminated by one of three ways: it is either accepted, rejected, or canceled. For instance the third point is reached just in case the application is approved which occurs in less than 25% of the recorded cases as mentioned earlier. Actually, this is the main reason why declarative models are more suitable to represent the compliance requirements than the procedural models. See section 2.3.

The results, i.e. fitness degrees, of the different operators are equal for completely compliant and completely non-compliant cases. However, they are not

equal for partially compliant cases. These cases are considered once non-compliant (using the Min operator) and once partially compliant (using an averaging operator). Table 5.5 shows the fitness degrees of cases using the four operators. The AM operator is used to measure the fitness degree at the cells level.

Table 5.5: The case fitness degrees for some selected cases

	Min	AM	WAM1	WAM2	WAM3	Range
7	0.00%	37.50%	57.14%	17.50%	39.47%	39.64 %
260	0.00%	80.00%	70.00%	60.00%	95.00%	35.00%
284	0.00%	80.00%	90.00%	60.00%	80.00%	30.00%
647	0.00%	75.00%	85.71%	33.33%	78.95%	52.38%
82	50.00%	87.50%	92.86%	66.67%	89.47%	26.19%
121	50.00%	87.50%	85.71%	95.83%	81.58%	14.25%
6	66.67%	88.89%	88.89%	91.67%	84.44%	7.23%
269	75.00%	91.67%	91.67%	93.75%	88.33%	5.42%

All selected cases are partially compliant. Other cases are not considered because the result is the same no matter which aggregation operator is used. The results are ordered ascending according to the result of the Min operator. The fifth, sixth and seventh columns of the table shows the results of the WAM operator using different set of weights. The weights assigned to WAM1, WAM2, WAM3 are: (10%, 20%, 10%, 30%, 30%), (40%, 5%, 7%, 8%, 40%), (5%, 30%, 30%, 30%, 5%) respectively. Notice here that the weights assigned to run WAM1 is the most realistic one for this example. The other two are used for comparison reasons only. When assigning the weights for WAM2 and WAM3 we take into consideration the number of checked cases and the fitness degree against each rule. The aim is to study the effect of the input weights on the case fitness degree. Notice here that according to our research, the weights are assigned by the process analyst. The entered values should reflect the importance of the business rules.

All fitness degrees are bounded by the minimum and maximum values of the inputs. Recall that this is because the aggregation function used have a compensation feature. The result of the WAM operator is interesting. It shows the importance of using it as an aggregation operator to measure the cases fitness degrees'. To explain our idea we will take case 647 as an example. This case is only 33% fit in WAM2, however, it is 85% in WAM1. This big difference (52%) is just because of the weights assigned at each run time. The last column shows the range of difference between the highest and lowest fitness degree using WAM. Normally, rules are not of the same importance. For instance, the fitness degree of

Case 7 is 37% if all rules are treated equally (AM result) however, it is 57% if the realistic weights are used (WAM1). Case 7 has the second highest range (almost 40%) after case 647. In contrast, case 269 has the smallest range of difference (around 5%) and then case 6 (around 7%). The reason is that the range between the Min and Max output is already small. Recall here that the WAM is one of the median approach operators and its output is a value between the Min and Max values inclusively.

Cases 82 and 121 score the same fitness degree using Min and AM. However, the result of the WAM is different. We checked the detailed value for these two cases. Both of them are checked against all rules except Rule 5. The fitness degrees for Case 82 are (0.5, 1, 1, 1) and the fitness degrees for Case 121 are (1, 0.5, 1, 1). Swapping the two values caused this deviation in the case fitness degree. The same situation is repeated with cases 260 and 284.

Comparing with the LTL checker [80], the technique provides two lists: one for correct PI(s) and another one for incorrect PI(s). A correct PI is the case in which all business rules are applied. In our case study 12 cases are listed in the correct PI(s) list. Others are listed in the incorrect instances no matter how many rules are violated. The LTL checker provides a measure called the "*health degree*" of the PI. It is used to measure the percentage of the applied rules to the total number of business rules. The "*health degree*" of a given case is similar to the AM result in our proposed technique, however, with some limitations. For instance, the aggregated values are either zeros or ones, no fractions. In addition, it does not consider the cases in which the rule cannot be checked. Using the LTL checker, these cases are listed in the incorrect PI(s). Recall here that the result of the LTL checker when checking the SOD rule is not equal. Thus, using our technique, the result of Rule 4 is different.

5.2.4 Cell level fitness degree

Given 1,000 traces and 5 rules, the tool generates 5,000 values (cells). Each value tells to which extent a single case is compliant against one specific rule. This is the most detailed measure which the end user have. It is important because it is used to calculate the other three metrics: cases, rules and the process. Recall here that the fitness degree of one case against one rule (cell) is computed by aggregating the fitness degrees of all its occurrences. There are four possibilities for the output: a dash, 0, 1 or in between (0-1). A dash indicates that the case

could not be checked against this specific rule. Rationally, this is the same no matter which aggregation operator is used. In our case, almost one fourth of the values are dashes. The majority (around 70% of the cells) are completely compliant. Around 5% of the cases are completely non-compliant. The remaining (39 cases) are partially compliant.

Mathematically, the output of the different operators are the same if there is only one occurrence. The result is also the same if the fitness degrees of all occurrences are equal. Recall here the aggregation operators have preservation of bounds property. The process manager perspective does not affect the result in any of these two cases and the result will be either 1 (completely compliant) or 0 (completely non-compliant). However, what about the cases in which some occurrences are compliant and some are not. These cases are considered partially compliant. As mentioned earlier, we have 39 cases which are partially compliant.

The fitness degree at the cells level is the atomic unit to measure the other three metrics. Hence, the aggregation operator used at this level affects the results of all other metrics. To study this effect, we select the cases fitness degree as an example. The cases fitness degree is chosen among the other two metrics, i.e. process and rules, because it has the minimum number of input values. Consequently, the contribution of each input value on the final result is more obvious. In this case study, the contribution of each cell in its corresponding case fitness degree is (at most) 20% whereas it is 0.1% and 0.02% for rules and process respectively.

For the comparison, six cases are checked against Rule 2. Rule 2 is selected because it is the most common one in the log file. As for the six cases, they are selected to cover the different scenarios. Cases 23 and 24 represent one occurrence for Rule 2. One of them is a compliant occurrence and one is not. The other four cases represent multiple occurrences. The scenario of zero occurrence is not represented because it is not considered in the calculation. For the purpose of comparison, the six cases are selected to have equal fitness degrees against the other four rules no matter which operator is used. For instance, the fitness degree of Case 121 is 1 against Rules 1, 3 and 4 and is not checked against Rule 5. This is the output regardless of the aggregation operator used. As an aggregation operator at the case level we select the AM operator. The AM is selected because it generates moderate values. In addition, it assigns equal weights to the cases no matter how long the trace is.

Results are shown in table 5.6. Starting by cases 23 and 24, Rule 2 is checked only once, thus, the result is the same no matter which operator is used. However,

it is not the case for the other four cases . Each of these cases is considered once partially compliant (AM and WAM) and once completely non-compliant (Min) according to the operator used.

Table 5.6: The effect of the operator used at the cell level on the fitness degrees for their corresponding cases

Case#	Min		AM		WAM	
6	0.00	0.67	0.67	0.89	0.50	0.83
23	0.00	0.80	0.00	0.80	0.00	0.80
24	1.00	1.00	1.00	1.00	1.00	1.00
121	0.00	0.75	0.50	0.88	0.33	0.83
206	0.00	0.67	0.80	0.93	0.67	0.89
269	0.00	0.67	0.75	0.92	0.60	0.87
average	0.17	0.76	0.62	0.90	0.52	0.87

The operator is chosen according to the situation. For a process manager, some business rules cannot be violated. Recall here that when we define the business rules for this process we consider the last two rules as the most important rules which should be applied strictly. One violating occurrence is enough to say that the case violates the given business rule. For instance, according to Rule 5 if the loan department manager approves the application once then this is enough to consider it as compliant. The AM and WAM is used when it is important to consider each occurrence. This is normally the case when checking sequencing requirements such as Rule 2 (the rule under investigation). Using the AM, all occurrences have equal weights. In contrast, using the WAM occurrences are assigned different weights according to their order in the trace. The first occurrence is assigned the highest weight. This is yet another perspective to tell whether a case is compliant or not. Notice that the AM result is higher than the WAM result in the four cases. However, this is not always the case.

5.3 Conclusion

In this chapter we have used the *Flexible Compliance Auditing Framework* to check the compliance of a synthesized data set. The loan approval business process is selected for this purpose. The log file is generated using the *SecSy* tool. The implemented tool is run to measure the fitness degree at the different levels of abstraction and using different aggregation operators. Results show that given

the one log file and a *BRCA* model, the tool produces different results. Each of which can be interpreted in a different way according to the business context in which it is used.

Chapter 6

The Procurement Process: A Real Life Data Set

The proposed technique is applied to a real data set of 10,000 cases. The cooperative organization is an international financial services provider. The input data file was derived from the procurement process cycle configured in an SAP® system. The cycle starts with creating a purchase order and ends with the payment of the associated invoice(s). Each case in the log file represents an item line of a purchase order. The procedure of process selection and data preparation was described thoroughly in the work of Jans [121].

A new process instance is triggered by creating a purchase order with its item line(s) and then signed before being released. In some cases the PO can be released without an additional signature. After a release, goods and an invoice are received. Finally, after both goods and the invoice are received, the payment can be made. However, if the purchased product is a service then no goods are received. The item line of the PO can be changed according to some business rules.

There are seven activities executed in this process: Create PO, Sign, Release, GR, IR, Pay. GR and IR refer to Goods receive and Invoice receive respectively. The business rules representing the process model are:

1. Sign should be followed eventually by release
2. SOD (Sign and Release)
3. SOD (GR and IR)
4. SOD (Release and GR)
5. Every case should be released at least once
6. A process instance should be executed within 60 day

7. If the PO value is 12,500 or less then an order change up to 5% is permitted without a new approval. Otherwise, if the PO value is between 12,500 and 125,000 then an order change up to 2% is permitted without a new approval. Otherwise, if the PO value is above 125,000 then it cannot be changed without a new Sign
8. Pay should be the last activity
9. If 'Pay' exists and then 'IR' exists where $\text{Pay}(\text{payObjectKey}) = \text{IR}(\text{IRObjectKey})$

The rules represent all types of compliance requirements: sequencing, human resource and data, temporal. Rule 1 is an example of a sequencing requirement. It is defined to ensure that each 'Sign' activity is followed by a 'Release activity' in the future. Rules 2-4 are of the same type; human resource. They are set to check that each pair of activities are not executed by the same originator. At the cell level, the result of the SOD rule is 0, 1, or a dash. A dash indicates that at least one of the two activities is not executed. Hence, the rule is neither applied nor deviated. If there is no intersection between the originators of the two activities then the rule is applied and the result is one. Otherwise, it is zero. According to Rule 5, the 'Release' activity should be executed at least once per case. Rule 6 is an example of a temporal rule. A case is compliant if it is executed within 60 days. Rule 8 ensures that the last event in the trace is of type 'Pay'. According to the process analyst, Rule 9 is the most important one. It is used to ensure that whenever there is a payment there should be a corresponding invoice received. According to the rule, it does not matter which one comes first so any order is considered compliant.

Rule 7, is the rule which controls the 'Change line' activity. The rule is checked according to the PO value (case attribute) and the relative modification (an activity attribute). The Change line activity allows the user to do some changes on the purchase order created per line. This can be a change in quantity, description, etc. If the requested change increases the PO value then it needs to be controlled. The relative modification attribute is used to quantify the percentage of change on the PO value. The relative modification is zero (in case the PO value is not affected), positive (if the PO value after change is greater than the original one) or negative (otherwise). This rule is concern with cases in which the relative modification is positive. The rule is divided into three branches according to the PO value. If the PO value is less than 12,500 then a change up to 5% is permitted without a new signature. If it is between 12,500 and 125,000 then a change up to 2% is

permitted without a new signature. Otherwise, i.e. above 125,000, there must be a new signature. A new signature mean that the 'Change line' activity should be followed by a 'Sign' activity and then this 'Sign' should be followed by a 'Release' activity. Notice here that according to the rule, it is not necessary that the three activities followed each other directly but sometime in the future. Notice that this rule is not checked if the trace does not contain a 'Change line' activity. Also, if the relative modification is negative or zero then the rule is not checked as well.

Before discussing the results we would like to notify the reader that the aim here is to show the importance of the following points rather than interpreting the violating cases. The interpretation of the violations is discussed in details in the work of Jans [121].

1. The importance of measuring compliance at different levels of abstraction starting from the top level (entire log) and moving down to the bottom level (cells)
2. The importance of using different aggregation operators to measure the fitness degree and how each of which support a different perspective
3. The importance of differentiating between the cases in which the rule can be checked and others
4. How declarative modeling languages can support compliance checking techniques better than procedural languages

Hereafter, we start with the most abstract level, the entire log. Next we discuss the results of the rules and traces and then the cells results.

6.1 Process level fitness degree

The fitness degree of the entire log using the proposed technique is affected by the aggregation operator used at the cells level. In our case there are 10000 cases which are checked against 9 rules. So, if all rules can be checked, there are 90000 values to be aggregated. Hence, the contribution of each value is very small. The fitness degree of the entire log using the different operators at the cells level are shown in Table 6.1. The first column shows the operator used at the cells level. The other three columns show the process fitness degrees using the AM function as an aggregation operator. In the second column, the fitness degrees are computed by aggregating the fitness degrees of all cells (unless it is a dash). Column 3 shows the results if the rules fitness degrees are aggregated. In our case, nine values are aggregated. The last column shows the results if the fitness degrees of cases are

aggregated. In the third and fourth column the AM operator is used to calculate the fitness degree of rules and cases respectively.

Notice that the results of one column are very close no matter which operator is used. It seems as if the operator used does not affect the final result. However, this is not the case. The difference between the minimum and maximum of the input values is (at most 0.03%) indicates that there is a very small deviation in the input values. Talking about the second column, we find out that there are only 20 values out of the 90,000 making the difference!

Table 6.1: The process fitness degree

Aggregation operator for cells	AM of cells	AM for rules	AM for traces
Min	97.67%	90.10%	97.69%
AM	97.68%	90.11%	97.70%
WAM	97.68%	90.11%	97.70%

A noticeable remark about the results is the difference between the results of cells and cases on one hand and the results of rules on the other hand. For instance, if the Min operator is used, the log file fitness degree is 97.67% and 97.69% using the cells and cases values respectively. However, it is 90.10% if the rules fitness degrees are used for the calculation. This is because the number of aggregated values (input) in case of rules is very small comparing to that of cells and cases. By aggregating nine values using the AM operator, the contribution of each value is 0.11. Whereas, aggregating 10,000 values (in case of cases), the contribution of each value is 0.00001!

To show the importance of the modeling language used for compliance checking purposes, we compare our model with the Petri net model for the same process. Figure 6.1 shows the Petri net for the procurement process. It is taken from Jans work [121], as agreed on with the process manager.

As a procedural model, Petri nets represent sequencing requirements only. Hereafter, we compare the capabilities of the two process models, i.e. the Petri net and the business rules. Before starting, we would like to notify the reader that both models are agreed on with the process manager. Of course, this does not mean that the process manager has two different prescribed models for the same process. However, he is restricted with the capabilities of the process modeling language used in each case. Notice here that using a Petri net, the process manager cannot differentiate between following directly and eventually. Moreover, it

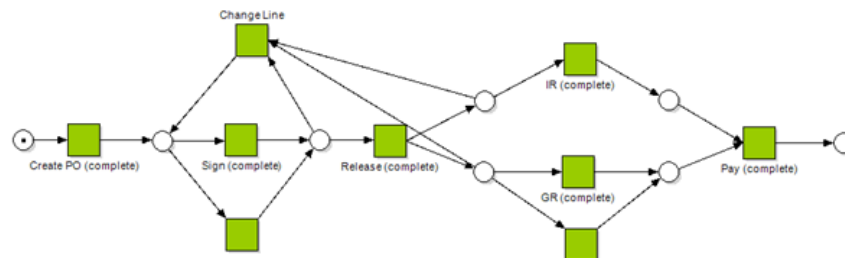


Figure 6.1: The procurement process Petri net

does not differentiate between proceeding and following. It can just represent the sequence of activities.

Starting with Rule 1, the rule indicates that a 'Sign' activity should be followed by a 'Release' activity in the future, no matter what is in between (if any). According to the Petri net model, there is a limited number of compliant paths. For instance, the path ('Sign', 'GR', 'Release') is compliant according to the rule but not compliant according to the Petri net. Similarly, Rule 5 is represented in the Petri net with new constraints including what should follow and precede a 'Release' activity.

Rules 2,3, and 4 cannot be represented using this type of Petri nets. These rules can be represented if a colored Petri net is used. However, the Replay technique reads a Petri net not a colored one. Hence, these requirements are not checked here. Rules 6 and 7 are yet another two rules that cannot be represented using the Petri net. The former is a temporal requirement and the later is a compound one. Rule 8 is represented in the Petri net with some constraints about which activities should proceed it. The last rule, Rule 9, is weakly modelled. The Petri net shows that 'IR' is followed by a 'Pay'. The data value checking which 'IR' is corresponding to which 'Pay' is not considered. Moreover, according to the rule the order of 'IR' and 'Pay' is not important. However, it is important according to the Petri net.

Someone may argue that we conduct the comparison here with respect to the rules. However, it is not the case. Recall that both models are agreed on with the process manager. The difference between the two representation refers to the capabilities and limitations of each language. In case of the Petri net, the process manager has to specify all execution paths. On the other hand, using LTL rules, the process manager has to specify the violating cases. Even if a colored Petri net is used, as a procedural model, only specific paths are accepted, others are

considered non-compliant. Whereas, using a declarative language such as LTL, everything is allowed unless it is defined as a violating case. Taking into account the nature of the business world and the flexibility issue, process managers prefer to keep a balance between compliant processes and flexible execution. This cannot be guaranteed using a procedural language in which only specific compliant paths are defined.

We believe that Petri nets are capable of representing the process behavior in process discovery techniques. Nonetheless, they are weak in representing the compliance requirements in compliance checking techniques unless there are very specific paths to follow while executing the process and a flexible execution is not an option.

6.2 Rule level fitness degrees

Although the process fitness degree can be used as a performance indicator. However, it does not tell something about the fitness degree of each rule separately; the most applied rule, the most violated rule, rules that are 100% applied or violated, etc. To answer such question, we need to check the fitness degree of each business rule individually. The implemented tool is run different time to compute the fitness degrees of the nine rules using the different operators. Results are shown in table 6.2.

Checking the fitness degree of the nine rules, Rule 3 (SOD (GR and IR)) and 8 (Pay should be the last activity) are completely applied. Both rules score a fitness degree of 1 no matter which aggregation operator is used. On the other hand, Rule 7 (the Change line rule) is the most violated one. Its fitness degree using the AM operator for both cells and rules is around 22% whereas the next lowest fitness degree is around 86%. However, it is checked just in 9.27% of the cases. Hereafter, we discuss the results of each rule and then compare the results of the different operators taking into consideration cases in which the rule is not checked. In addition, we show the importance of considering a dash in the result.

Rule 1 is a sequencing requirement. The rule is checked as much as there are 'Sign' events in the case trace. This rule is not checked in 9.54% of the log file which indicates that we have 954 cases in which the 'Sign' does not appear.

The SOD rules (2-4) cannot be checked if (at least) one of the two activities is not executed. If the rule is checked, then the result is either 0 or 1. The result is 1 if there is no intersection between the originators of the two activities and zero

Table 6.2: The fitness degree of the business rules using the different aggregation operators

	1	2	3	4	5	6	7	8	9
Min	50.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	100.00%	7.69%
AM	99.99%	99.36%	100.00%	99.79%	99.99%	89.63%	22.37%	100.00%	99.89%
WAM	99.98%	98.66%	100.00%	99.79%	99.99%	85.84%	32.14%	100.00%	99.74%
Cases not checked	954	0	0	0	0	0	9073	0	0

otherwise. Results show that each of the three rules is checked in each case. Rule 3 is completely applied in all traces. Rule 2 is violated 64 times and Rule 3 is violated 21 times. According to the process manager, Rule1 should be checked pairwise. To explain the idea we will use the following trace as an example: ('Create PO', 'Sign', 'Release', 'Sign', 'Release', 'GR', ..). The originators of the first 'Sign' and the second 'Release' is the same so that its fitness degree against Rule1 is 0. However, according to the process analyst the result is 0 just in case the originators of the 'Sign' and its direct follow 'Release' is the same. Checking the 64 cases (at the cells level) reveals that none of them is a violation [121]. The result we got shows the need for a new SOD function in which the segregation of duties is checked pairwise. As for Rule 4, extra analysis reveals that there are four persons responsible for the 21 violating cases. For further information readers may refer to [121]. Recall here that the aim is to locate the deviation rather than interpreting it.

Rules 5, 6 and 8 are examples of rules that are checked exactly once in each case. The results of the three rules is either 0 or 1. No other possibilities. According to Rule 5, the case is compliant just in case it contains a 'Release' activity. The rule is violated only once. Rule 6 is the only temporal rule. The rule is defined to control the throughput time. It is checked by calculating the difference between the timestamps of the first and last activity. If the difference is within 60 days then the case is compliant. Result show that this is the second most violated rule after rule 7. Over 10% of the cases took more than 60 days to complete. To get more insight, we run the 'Replay a log on petri net for performance/conformance analysis' under prom. Results, see figure 6.2, show that some cases took around a year before it is completed. These cases need to be checked by the process manager. Rule 8 is defined to ensure that the last event in each case is of type 'Pay'. The rule is completely applied in all cases.

Rule 7 is the most violated and the least checked rule (less than 10% of the

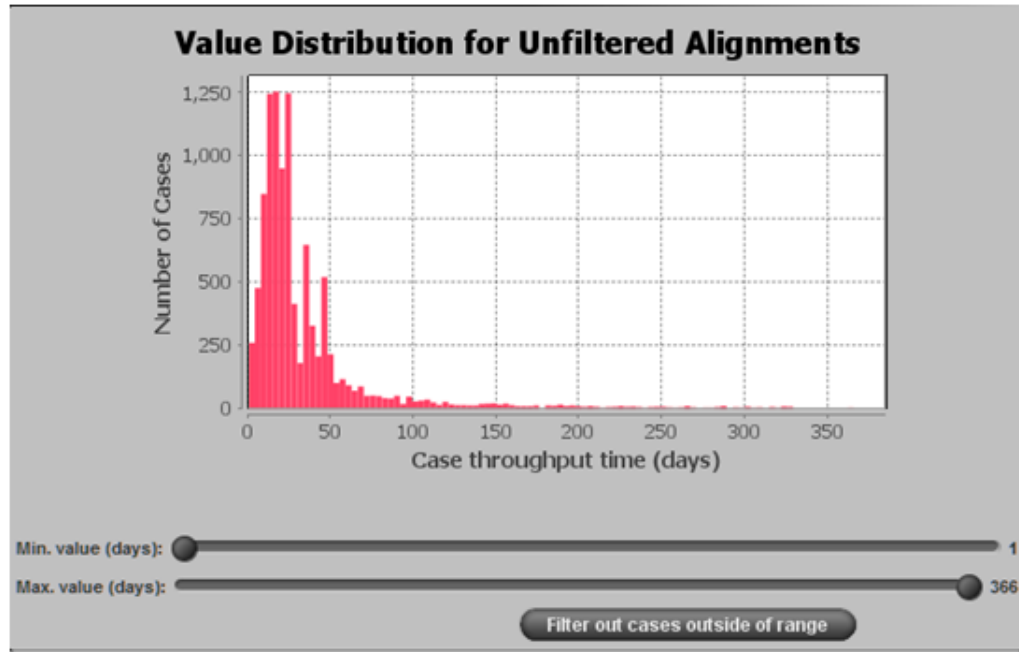


Figure 6.2: The distribution of cases throughput time

cases). The rule is a good example to show the importance of considering a dash in the result of compliance checking technique. Recall here that a dash indicates that the rule could not be checked. In the log file there are 9073 cases in which the result is a dash for some reasons. Let us assume that these cases are considered compliant. Then the new fitness degree using the AM operator would be 92.80% instead of 22.37%. On the other hand, if these cases are considered non-compliant then the results would be only 2.07%! For a process manager, each result can lead to a different decision. Hence, we believe that cases in which the rule is not checked should never be included when calculating the fitness degree.

Finally, Rule 9, is an example of a compound rule in which a sequencing requirement and a data requirement are merged together in one rule. According to the process manager recommendation, the rule is checked according to the appearance of the 'Pay' event. The aim is to ensure that each payment activity has a corresponding invoice received. This is the most important rule since it is assigned the highest weight (20%) among all other rules which are assigned 10% for each. Checking the results at the cells level we found out that the rule is violated in 16 cases with a total of 71 times. The process manager needs to check these cases thoroughly.

In what follows we compare the results of the same input using different operators. The results of rules 3 and 8 are discarded in the discussion because the results are equal no matter which operator is used.

Comparing the results of AM and WAM especially for Rules 6 and 7, the same input produces two different outputs just because the trace length is considered. In general, the results of AM and WAM will be close in two cases. First, when the length of the different traces is close to one value. Second, when the majority of traces have equal fitness degrees against this specific rule. The first case, i.e. similar lengths, is not applied here. In the log file there are traces of only four events, on the other hand, there are traces of 202 events. However, the second case is applied. The results of AM and WAM are equal for some rules, such as Rules 4 and 5, because the aggregated values are equal. In contrast, there is a relatively big difference in the results of Rule 7. It seems like long traces are less compliant against Rule 6, whereas they are more compliant against Rule 7.

With respect to the results we got, the main question is when to use each of the six operators? And how they can support the different perspectives of process analysts?

The Min operator can be used when the rule should be followed strictly. This is the best choice just in the following case: the rule is considered 100% applied if it is never violated in the entire log.

The AM and WAM are used when the process analyst aims at checking to which extent a rule is applied. A fraction in the result indicates that the rule is partially applied. Choosing AM or WAM depends on different factors such as the nature of the process under investigation and the business rules defined. If the process analyst believes that long traces are expected to have more deviations then he should use the WAM operator. Notice here that this could be the case when the rule can be checked more than once such as Rule 1 in our example. However, it is not possible for rules that are checked exactly once such as rules: 5 and 8. Rule 6 is an exception here since it is defined to check the trace length.

To test the relation between the number of occurrences and the results of AM and WAM, we run the tool again using the WAM as an aggregation operator at the cells level. Results show a very small deviation in the result. This needs more investigation in the future.

6.3 Case level fitness degree

A log file which is 98% compliant indicates that there are some non-compliant cases. Checking the fitness degree at this level, i.e case level, can help process managers get insights and check the fitness degree of each case individually. A non-compliant case could be a fraudulent one. In this work, we propose three metrics to check the compliance of one case: Min, AM and WAM. The measure to use is selected according to the process manager perspective. Using the WAM operator, rules are assigned weights according to some input from the process manager.

The implemented tool is used to compute the fitness degrees of the 10,000 cases. The AM operator is used to compute the fitness degree at the cells level. A general look at the results shows that each case is checked against at least one rule. No dashes in the result. Almost, 85% of the 10000 cases score a fitness degree = 1. The results of the AM and WAM operators are very similar. The weights defined by the process manager are in order: (10%, 10%, 10%, 10%, 10%, 10%, 10%, 10%, 10%, 20%). Recall here that the AM operator can be seen as a special case of WAM in which weights are equal. In addition, when there is a very small deviation in the input, such as our case study here, the output is not affected heavily by the operator used. Notice here that cases in which the rule could not be checked are not considered in the calculation.

6.4 Cell level fitness degree

The cell fitness degree represents the fitness degree between a specific case and a business rule. This value is affected by the number of occurrences. Three aggregation operators are used for this purpose: Min, AM and WAM. The weights in the WAM operator are assigned automatically according to the position of occurrence. In our case study, we compare the results of AM and WAM for Rule 9 to study the effect of the occurrence location. We found out that the deviation between the results of AM and WAM operators is at most 0.16. Rule 9 is selected because of the number of occurrences. Most of the other rules are checked at most once. Thus, the output of the four operators are not comparable.

Normally, process analysts start their analysis from an abstract level. Hence, this metric is not used unless a deviation at the higher levels is detected. That being the case, this metric can be used to support the analyst with further diagnostic

information.

6.5 Conclusion

In this chapter we have used the *Flexible Compliance Auditing Framework* to check the compliance of a real data set. The cooperative organization is an international financial services provider. The log file contains information about 10,000 cases extracted from the procurement process operational databases. The process manager provides a process model of nine business rules. The process model is presented by means of the BRCA language presented in chapter 3. The implemented tool is run to measure the fitness degree at the different levels of abstraction. Results show the importance of providing different aggregation operator to measure the fitness degree. Process managers can select the suitable operator according to the semantic they have.

Chapter 7

Clustering Based Compliance Checking Approach

In compliance auditing literature, existing techniques provide an overall fitness degree measuring the fitness degree of the entire log file. The computed degree cannot distinguish between cases that are more compliant with the designed process and those which are less compliant. By investigating the included cases, we may find that there are some common characteristics between the less compliant cases. For example, in the procurement process, we may find that most of the less compliant cases are created by a single person or that they are purchased by the same group. Revealing such relations would be a good start for process managers to investigate these cases in depth. According to the investigation results, managers can take some actions to improve the process in the future. To this end, we propose a clustering based compliance checking approach in which the log file is clustered before the process compliance is checked.

We believe that clustering the log file before compliance checking can reveal important diagnostic information [122, 123]. Hence, we introduce a clustering based compliance checking approach in which the log file is divided into a set of sub-logs before the process compliance is checked. Next, the fitness degree of each generated cluster is measured separately and compared with the fitness degree of the entire log. Generated clusters are profiled according to some features to find the common characteristics. We assume that clusters with lower fitness degree with respect to the overall fitness degree have some common characteristics which leads to, relatively, less compliant cases. The proposed approach is applied

to the procurement process discussed in chapter 6.

This chapter is organized as follows: first, we discuss clustering techniques in the process mining field and made a decision about which is the most suitable clustering technique for this research. Next, we use the multi-perspective compliance checking technique proposed in Chapter 6 to measure the fitness degree of each cluster separately. Fitness degrees are calculated for the entire cluster and per rule. Later, the generated clusters are profiled according the process manager recommendation. Finally, the results are discussed and some recommendations are made.

7.1 Clustering in process mining

To the best of our knowledge, two clustering techniques are introduced in the process mining literature: sequence clustering and trace clustering. Using sequence clustering, the log file is divided into k clusters according to the similarity of sequences. However, trace clustering techniques use a set of features to measure the similarity between the process instances. Similar cases are grouped together in one cluster.

Sequence clustering was first developed in bioinformatics to group large protein data sets into different families [124]. Later on, the idea is applied in the process mining field to study the different behaviors within a single process [125, 126]. It is useful to discover the behavior of different traces in the log file and divide them into clusters. Hence, it is applied to deal with spaghetti models [127].

Initially, the sequence clustering algorithm generates k number of clusters where each cluster is associated with a Markov chain model. After the clusters and their corresponding Markov chains are generated, each individual sequence in the input log file is assigned to one of the generated clusters. To assign a given sequence to a cluster, the probabilities of all generated clusters to produce this sequence is computed first. The sequence is assigned to the cluster with the highest probability to produce this specific sequence. In a next phase, these steps are repeated in an iterative Expectation-Maximization procedure to re-estimate the models. The sequences assigned to a cluster are used to re-estimate the Markov chain of that cluster. This produces more accurate models to be used to assign the sequences again in the next iteration. By repeating the procedure again and again, the models (Markov chains) become more stable which no longer change. The final output Markov chains are the models that represent the behavior of its

associated cluster [125].

As for trace clustering, it is a hierarchical clustering technique. First, the traces are assigned to different clusters. Next, similar clusters are combined together until it ends up with one single cluster. In process mining, trace clustering is used to support both process discovery and conformance checking. In [123], the authors propose a process discovery technique based on trace clustering. The authors in [128] present a trace clustering approach based on profiling the traces. To compute the distance, a combination of bag-of-activities and k-gram is used. Bose et al. [129] propose a generic edit distance based approach to measure the (dis-)similarity between two sequences. The edit distance (also called Levenshtein distance) between two sequences is defined as the minimum number of edit operations needed to transform a sequence into the other. An edit operation, in this context, can be an insertion, deletion, or substitution of two events. The minimum variance criteria is used to choose the two clusters to be combined.

In this work, we use a trace clustering technique to cluster the log file. Sequence clustering is discarded for two reasons. First, it does not consider any dimension other than the sequencing one. However, we aim at checking the four types of compliance requirements. Second, sequence clustering is used in a previous work [130] for the same purpose and it did not provide promising results.

7.2 Clustering based compliance checking approach

In this section, we describe the clustering based compliance checking approach. In a first step, we cluster the log file into more homogeneous subsets. Next, we used the multi-perspective compliance checking technique (See Chapter 4) to compute the fitness degree for each cluster separately. The fitness degrees are calculated at two levels, the entire log and the rules. Next, the results are compared with those of the original log file. Finally, we profile each cluster according to the process manager recommendation to find out the common characteristics. This way mutual benchmarks are created. Process analysts are provided with multiple compliance degrees and therefore gains insights in the compliance of separate groups. By analyzing the characteristics of a cluster with a higher/lower compliance degree, insights on compliance determinants are developed.

7.2.1 Clustering the log file

As mentioned earlier, trace clustering is chosen to divide the log file into k clusters. The value of (k) is predefined by the user as an input parameter to run the clustering algorithm. But, how to determine the value of (k)?

This question has no general answer in the data mining field because each data set has its own case. In practice, the clustering algorithm is run different time and then, according to the results, the data analyst determines the most suitable solution. In this research, we leave this decision to the process analyst because he/she has the highest level of understanding the process under analysis. However, we provide a recommendation, later on, which can help in making the decision.

In our case, the fitness degree of the original log file is 97.68%. The majority of cases (almost 85%) are 100% compliant regardless of the aggregation operator used. With respect to the proposed clustering based approach, these cases should be assigned to one cluster. Accordingly, the clustering result should contain one big cluster and that cluster should have a fitness degree higher than that of the original log because it contains the most fitting cases.

We started running the clustering technique with $k = 2$ and stopped at $k = 7$. The number of cases in each cluster is recorded in Table 7.1. When running the technique with $k = 7$ we noticed that the big cluster generated in the previous run, i.e. 6 Clusters, is divided into, relatively, two big clusters. This indicates that the most fitting cases are not grouped together in one cluster any more. This is an important remark to determine the value of k . In our case, the majority of cases are 100% compliant. Accordingly, we made the decision to stop incrementing value of k when the big cluster, which we assume it contains these compliant cases, is divided into two big cluster.

Notice that each time the value of k is incremented by one, one of the clusters in the previous output is divided into two clusters. For example, the 9900 cases in (3 clusters run time) is divided into two clusters one with 9815 cases and the other with the remaining 85 cases in (4 clusters run time). This is because trace clustering is a hierarchical clustering technique as mentioned earlier.

Although our results cannot be generalized because each log file has its own cases. However, we can end up with a general recommendation to help process analysts determining the value of k . Start running the clustering technique with $k = 2$ and increment the value by one each running time. Each time you run the algorithm, you need to write down the number of cases assigned to each cluster.

In addition, you need to check the fitness degree at the case level to have an idea about the compliance of recorded cases individually. It would be better if you can run the framework presented in chapter 4 to measure the fitness degree of generated clusters. However, this is not necessary. Notice that, according to the clustering based approach, fitness degree is measured after clustering the log file. Once you have the results of different run times and an idea about the fitness degree of individual cases, then you need to stop when the clustering results show that the majority of cases are divided into two clusters. In case you want to check the fitness degree of generated clusters, then you need to stop when you find a cluster with fitness degree higher than the process fitness degree.

Table 7.1: The number of clusters generated at each run time to determine the best value for k

	2 Clusters	3 Clusters	4 Clusters	5 Clusters	6 Clusters	7 Clusters
Cluster 1	51	51	51	51	12	12
Cluster 2	9949	49	49	49	39	39
Cluster 3	—	9900	85	85	49	49
Cluster 4	—	—	9815	138	85	85
Cluster 5	—	—	—	9677	138	138
Cluster 6	—	—	—	—	9677	5362
Cluster 7	—	—	—	—	—	4315

The log file is divided into six sub-logs using the guide tree miner plugin under ProM version 6.5.1. For clustering, the plugin is configured as follow: *entire trace* as a feature type, the *generic edit distance* as a distance metric and the *Min variance* as a join type of agglomerative hierarchical clustering [129].

7.2.2 Measuring compliance

After clustering the log file, the fitness degree of each cluster is checked separately using the implemented tool. The fitness degrees are checked at two levels: the entire log and the rules. The others, i.e. cases and cells, are discarded because comparing their results is not applicable. For comparison reasons, we prefer using one of the median operators to avoid any bias. Among the two median operators, i.e. AM and WAM, we choose the AM function because we wanted to treat all traces equally regardless of their length.

1. The entire log: the fitness degree of the entire log for each cluster is measured and then compared with the fitness degree of the parent log. Results

are shown in table 7.2. As mentioned earlier, the AM is used as an aggregation operator. To measure the fitness degree of the entire log, the fitness degrees at the cells level are used as input. Recall here that the fitness degree of the entire log is calculated by aggregating the fitness degrees at one of: cells, rules, or cases.

Table 7.2: Fitness degrees of the generated clusters and the original log

	Percentage of cases	The entire log fitness degree
Cluster 1	0.12%	91.16%
Cluster 2	0.39%	89.36%
Cluster 3	0.49%	83.48%
Cluster 4	0.85%	90.84%
Cluster 5	1.38%	88.37%
Cluster 6	96.77%	98.00%
The original log	100%	97.68%

Results reveal a noticeable remark. The largest cluster is the only one for which the fitness degree is higher than that of the original log file. On the other hand, the fitness degrees of all other clusters are lower than that of the parent log. Results indicate that the most fitting cases are grouped in one cluster. This goes with our assumption when clustering the log file and selecting the most suitable value for k , i.e. the number of generated clusters.

2. Rules: the rules' fitness degrees are measured for each generated cluster. Results are show in table 7.3. Notice that the results of rules 3 and 8 are not given in the table because the two rules are 100% applied in the parent log. The discussion will be narrowed to include three rules only: 6, 7 and 9. The others are discarded for the following reasons.

- (a) Rule 1 is discarded because the violating cases are divided into three clusters. Additionally, there is a slight difference in the results of these clusters and the parent log.
- (b) Rule 2 is discarded because, as mentioned earlier, the rule is not violated according to the process analyst.
- (c) Rules 3 and 8 are discarded because they are 100% compliant in the entire log.
- (d) Rules 4 and 5 are discarded because the cases in which these rules are violated are grouped in one cluster, the largest one, containing 96.77%

of the cases. Hence, the results cannot be compared.

Table 7.3: The fitness degrees of each cluster against each of the nine rules

	Rule 1	Rule 2	Rule 4	Rule 5	Rule 6	Rule 7	Rule 9
Cluster 1	100.00%	100.00%	100.00%	100.00%	33.33%	66.67%	100.00%
Cluster 2	100.00%	97.44%	100.00%	100.00%	20.51%	88.89%	97.05%
Cluster 3	99.49%	53.06%	100.00%	100.00%	26.53%	36.36%	99.32%
Cluster 4	100.00%	98.82%	100.00%	100.00%	50.59%	14.29%	90.68%
Cluster 5	99.64%	73.91%	100.00%	100.00%	32.61%	91.43%	100.00%
Cluster 6	99.99%	99.97%	99.78%	99.99%	91.45%	18.69%	99.99%
Original log	99.99%	99.36%	99.79%	99.99%	89.63%	22.37%	99.89%

Before going to the next step, we would like to consider the number of not checked cases for each of the three rules: 6, 7 and 9. Recall here that using the AM operator, the number of cases that are not checked affects the fitness degree measure. Starting with Rules 6 (the 60 days rule) and 9 (the Pay_IR rule), their results are never affected. Both rules are checked in each specific case in the entire log. In contrast, Rule 7 (the Change line rule) is strongly affected by this factor. The rule is not checked in over 90.73% of the cases recorded in the parent log. Only 927 cases out of the 10,000 cause the deviation in the results. However, these cases are divided into 6 clusters. Notice, here that since there are no dashes in Rule 7 column then each cluster has a share of these 927 cases.

After this discussion, the results table is eliminated to include the results of rules: 6, 7, and 9 in addition to the number of cases in which Rule 7 is checked in each cluster.

Table 7.4: Fitness degrees of each cluster against rules 6, 7, 9 and the original log file

	Rule 6	Rule 7	Rule 9	entire log
Cluster 1	33.33%	66.67%	100.00%	91.60%
Cluster 2	20.51%	88.89%	97.05%	89.36%
Cluster 3	26.53%	36.36%	99.32%	83.48%
Cluster 4	50.59%	14.29%	90.68%	90.84%
Cluster 5	32.61%	91.43%	100.00%	88.37%
Cluster 6	91.45%	18.69%	99.99%	98.00%
The original log	89.63%	22.37%	99.89%	97.68%

By comparing the results in the table we could prove our point of view that the overall fitness degree is not representative enough to measure the process

compliance. The fitness degrees against individual rules could reveal much more information about violating cases. For instance, although Cluster 6 is 98% compliant and Cluster 5 is only 88.37% compliant, however, the result of the two clusters against Rule 7 tells something totally different. Cluster 6, which is the most fitting one, records the second lowest fitness degree (18.69%) among all other clusters. Whereas, Cluster 5 records the highest fitness degree against the same rule.

Similarly, the results of rules 2 and 4 reveal that the overall fitness degree is not enough to measure the conformity degree. Although the two clusters have a relatively close fitness degrees (89.36% and 90.84%), however, their fitness degrees against the individual rules are not close. Cluster 2 has the lowest fitness degree against Rule 6. However, it has the second highest fitness degree against Rule 7. In contrary, Cluster 4 has the second highest fitness degree against Rule 6 whereas it records the lowest fitness degree against Rule 7. Notice that the number of checked cases is considered when comparing the results of Rule 7 to make sure that it did not affect the fitness degrees. As for Rule 9, there is also a relatively high difference between the fitness degree of the two clusters against this rule.

Notice that by clustering the log file, the most fitting/violating cases against one specific rule are grouped together in one cluster. This way, the procedure of analyzing these cases becomes much easier. In the next step, we profile the six clusters to find out the common characteristics between clusters with relative high/low fitness degree to get insights.

7.2.3 Profiling the clusters

The output of the previous two steps is a set of clusters with their corresponding compliance degrees. In this step, we profile the clusters by means of case information to have a closer look at the characteristics of each cluster. The selected information depends on the case under study. However, some recommendations from the process managers would be a good start. The recommendations in this case will be a set of attributes that the process manager thinks might affect the compliance degree whether positively or negatively. The output of this step will be a set of relations between the compliance degree and the selected attributes' values such as the cluster where the majority of cases created by the originator X have a lower compliance degree.

In our case study, four attributes are analyzed according to the process manager recommendation. These are: the document type, the purchasing group (PG),

the purchase order (PO) value and the purchase order (PO) creator.

1. Document type

Starting with the document type, there are mainly five types of documents. Each case is related to only one type. Originally there were nine types but the remaining five are discarded because of having very low frequencies. The idea behind discarding the less frequent values of document types is that the number of cases for these types is not enough to do the comparison so that including them does not add a value to the results.

Figure 7.1 shows the distribution of document types over the six clusters. The values are normalized over the document types count to overcome the large difference in the number of each type in the six clusters and therefore to better represent the results. The normalization was done by dividing the number of each document type in one cluster by the total number of this type in the entire log file. Moreover, the values related to cluster 6 are divided by 10 to better visualize the values in other clusters. This was necessary because a relatively high number of cases fall under cluster 6. The normalization described here is applied to all other attributes for the same reasons.

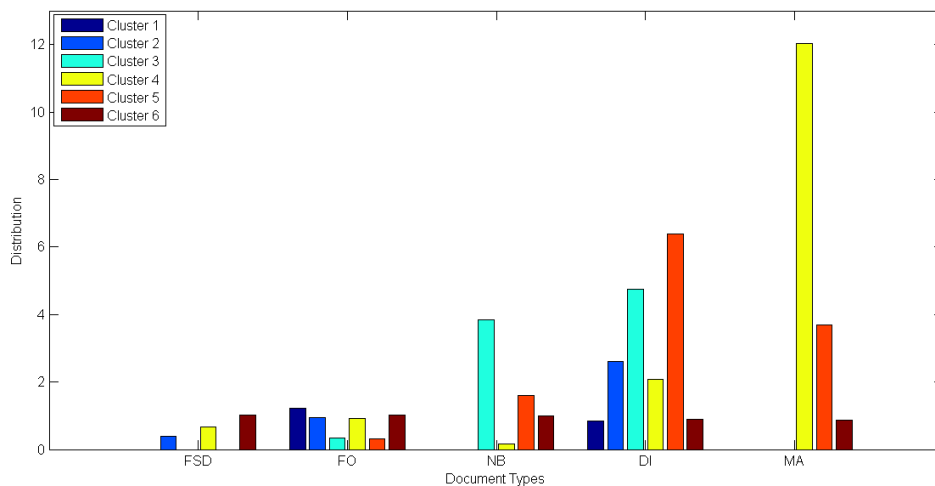


Figure 7.1: The distribution of document types over the six clusters

The distribution shows that cluster 1 is dominated by document type FO and that document types: MA, NB, FSD does not appear in this cluster. Cluster 2 is dominated by DI and does not contain any case of MA or NB. As for cluster

3, the both DI and NB appears with close distribution. However, FSD and MA do not appear in the cluster. Cluster 4 contains all types of documents. The most frequent one is MA and then DI. Similarly, Cluster 5 is dominated by MA and DI. However, unlike Cluster 4, the percentage of DI is higher than that of MA.

FO is the most frequent one. It appears in more than 75% of the cases in the original log. On the other hand, MA is the least frequent one among these five types. Recall here that we discarded four types because of having very low frequencies.

2. Purchasing group

There are mainly 11 purchasing groups registered in the original log file. Only seven groups are considered in the analysis. Again, the remaining four are discarded because of their low frequencies.

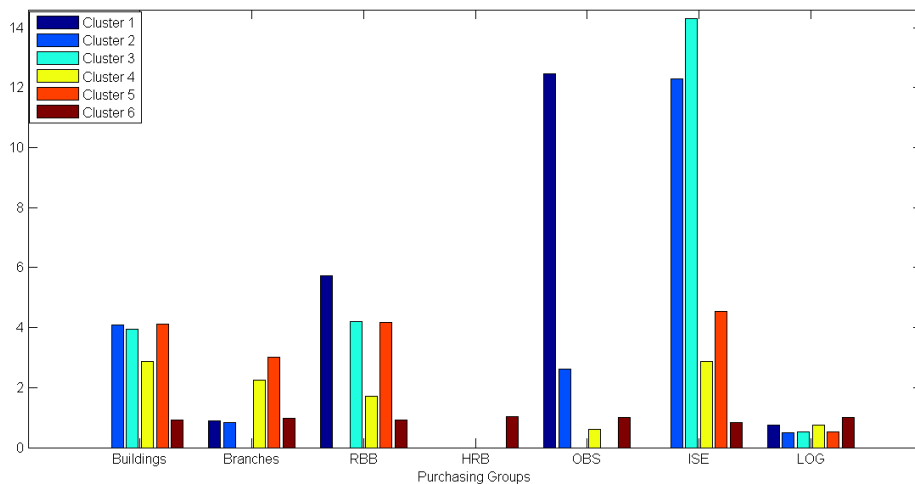


Figure 7.2: The distribution of purchasing groups over the six clusters

The PG LOG is the most frequent one (more than 78% of the cases). Results show that Cluster 1 is dominated by OBS. Both clusters 2 and 3 are dominated by ISE.

3. PO value

According to the process manager recommendation, the purchase order values are categorized into 4 using the thresholds 1,000, 1,250 and 12,500. We

profiled the four clusters according to the values of these four attributes.

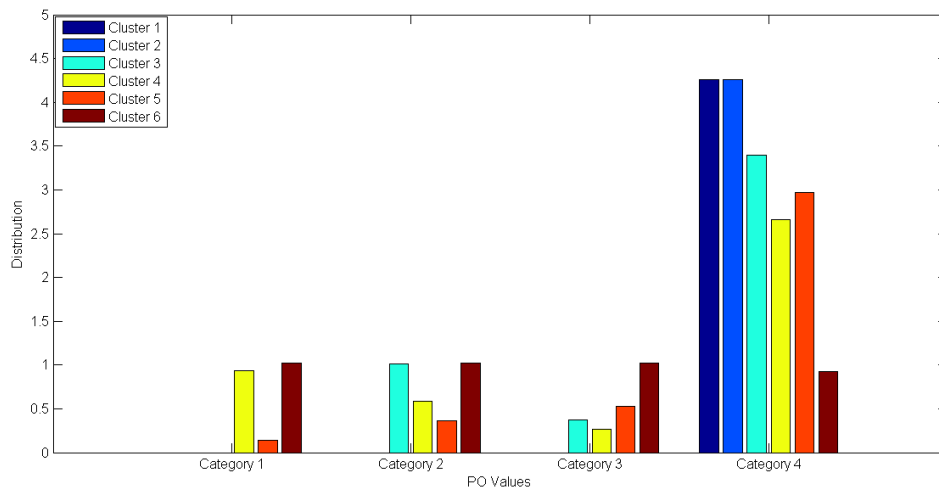


Figure 7.3: The distribution of PO values over the six clusters

Figure 7.3 shows the distribution of each category with respect to the original log. Results show that the cases in clusters 1 and 2 are all under category 4. The cases assigned to cluster 3 are mostly under category 4.

4. **PO creator** There are 81 different PO creators, most of them are less frequent (74 creators). However, these could not be discarded because, in total, they have the probability of 17% in the original log. So, they are treated as one group called others.

Figure 7.4 shows that the 'others' PG are assigned to all clusters.

7.3 Results discussion

We summarize the results obtained by profiling the clusters in table 7.5. The table presents the main characteristic captured by each of the six clusters. The analyzed characteristics included the most represented type of attribute and the most violated rule (measured by the fitness degree of the rule) for each cluster. Using this table, we aim at explaining the correlation between the clusters, the attributes, and the fitness rules. This correlation will point out the clustering role in identifying certain combinations of attributes that will affect certain fitness rules and will lead to a lower fitness values. Notice here that the fitness degrees of the entire

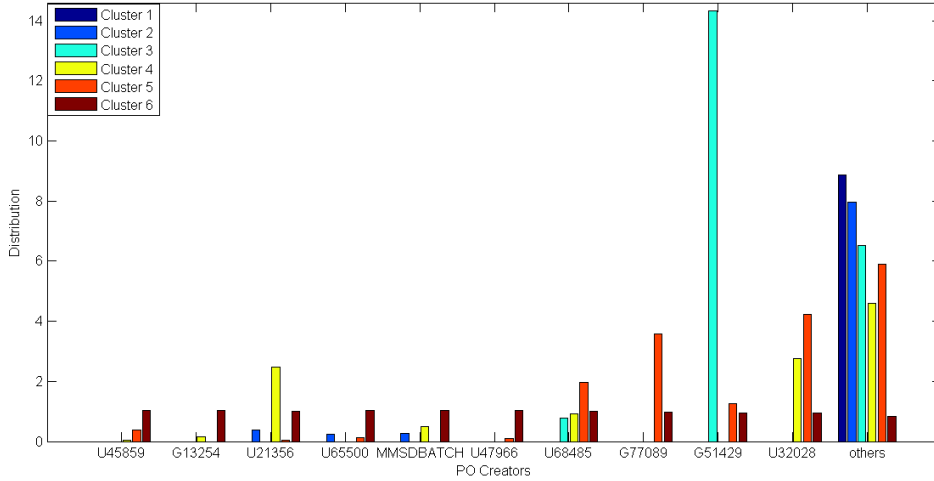


Figure 7.4: The distribution of PO creators over the six clusters

log is not considered since the rules results are more representative as discussed earlier in the chapter.

Table 7.5: Summary of profiling results

Cluster	Rule with lowest fitness degree	Rule with second lowest fitness degree	Dominant doc. type	Dominant PG	Dominant PO value Category	Dominant PO creator
Cluster 1	R6	R7	FO, DI	OBS, RBB	4	others
Cluster 2	R6	R7	FO, DI	ISE, Build.	4	others
Cluster 3	R6	R7	NB, DI	ISE, Build., RBB	4, 2	G51429, others
Cluster 4	R7	R6	MA, DI	ISE, Build., Branches	4, 1	others, U21356, U32028
Cluster 5	R6	R7	MA, DI	ISE, Build., RBB	4, 3	others, G77089, U32028
Cluster 6	R7	R6	FO, DI, FSD	Log, Branches, RBB	3, 1	U47966, G13254, U65500, MMSDBATCH, U45859

The results indicate that there is a general detection of document type ‘DI’, purchasing groups ‘Buildings’ and ‘ISE’ (except Cluster 1), PO value ‘category 4’ (except Cluster 1), and PO creator ‘others’ in all clusters with lower fitness degrees. Moreover, the distribution of the PO creators in the most fitting cluster ‘Cluster 6’ is the lowest! This indicates that the appearance of PO creator ‘others’ in the other five clusters affects their fitness degrees negatively. Recall here that ‘others’ groups the less frequent PO creators in the entire log file. Thus, we assume that these creators do not have enough experience to perform the procurement process which in turn leads to lower fitness degree.

The most noticeable remark in the table is the clear representation of certain patterns in the cases included in each cluster. For example, clusters 1 and 2 contain common document types, PO value categories and PO creators. However, the two clusters record different fitness degrees. Cluster 1 is dominated by PG 'OBS' and 'RBB'. However, Cluster 2 is dominated by PG 'Buildings' and 'ISE'. This maybe the reason behind the difference between the results of the two clusters against Rule 6. The rule is only 20.5% applied in Cluster 2 whereas it is 33.3% applied in Cluster 1. Recall here that Rule 6 is the rule to check the throughput time. Thus, it could be the case that these purchasing groups consume extra time.

Cluster 3 represents the pattern where Document type 'NB', PG 'ISE' and 'RBB', PO value 'Category 2', and PO creator 'G51429' appear. This indicates that cluster 3 grouped the cases with lower fitness degree (related to rule 6 and 7) caused by this pattern.

The highest Rule 7 effect on the fitness degree was isolated in clusters 6 and then 4. A noticeable remark about the distribution of the two clusters is that these are the only clusters containing cases of PO value 'Category 1'. In cluster 4, this was associated with document type 'MA', PG 'ISE' and 'Branches', PO value 'Category 1', and PO creators 'U21356' and 'U32028'. In Cluster 6, this was associated with document type 'FO' and 'FSD', PG 'Log', 'Branches' and 'RBB', PO value category '3' and '1', and PO creators 'U47966', 'G13254', 'U65500', 'MMSDBATCH', 'U45859'.

Regarding Rule 9, cases in Cluster 4 were the most violating cases and then those in Cluster 2. Comparing this with the results of clusters 1 and 5, which are 100% compliant against the same rule, we noticed that PG 'RBB' does not appear in Cluster 2 and has a relatively small distribution in Cluster 4. We assume that the presence of this PG affects the fitness degree positively.

7.4 Conclusion

In this chapter we present a new approach for compliance auditing based on clustering. The idea is to cluster the log file and then check the compliance of generated clusters. Later on, the clusters are profiled to reveal common characteristics in the least compliant and most compliant clusters. The aim is to provide diagnostic information about the reason behind violation. This chapter can be seen as a first step towards extending this research in the future to support the process manager with diagnostic information.

Chapter 8

BRCA Tool

The BRCA tool is developed as an implementation for the Flexible Compliance Auditing Framework. The tool can be used by process analyst to check the compliance of executed processes with respect to a normative model. The log file should be in the xml format. The normative model which is a BRCA model is a file with the extension *.br. It contains the set of business rules that should be checked. The tool provides the ability to create a new model, save it, and load a predefined one. However, to measure the fitness degree the the normative model is required. The output of the tool is a csv file containing the fitness degrees at all levels of abstraction. The tool is developed using the C#.net programming language.

Figure 8.1 shows the main screen. To run the tool, the user should have the log file in xml format. Other formats are not supported for the moment. However, if the log file is an mxml file or an xes file then they can be converted to an xml file easily by changing the extension. In some cases, the structure of the mxml does not work properly this way. In such cases, we recommend converting the file into xes file first using the ProM tool [119] and then change the extension.

Once the log is located, it is parsed and then the tool is tuned according to the recorded cases. In case the file could not be parsed, then the message (Cannot parsed the selected XML file) appears to notify the user. Once the log file is parsed, the business rules can be defined. The user has two options at this stage; he/she can either build the model from scratch, or open a predefined model using the (Load) option. In the former case, the new model can be saved to be used later. The normative model is a BRCA model with the extension (*.br). To build a new BRCA model, the tool provides a set of predefined rules representing the most frequent checked requirements as shown in figure 8.2.

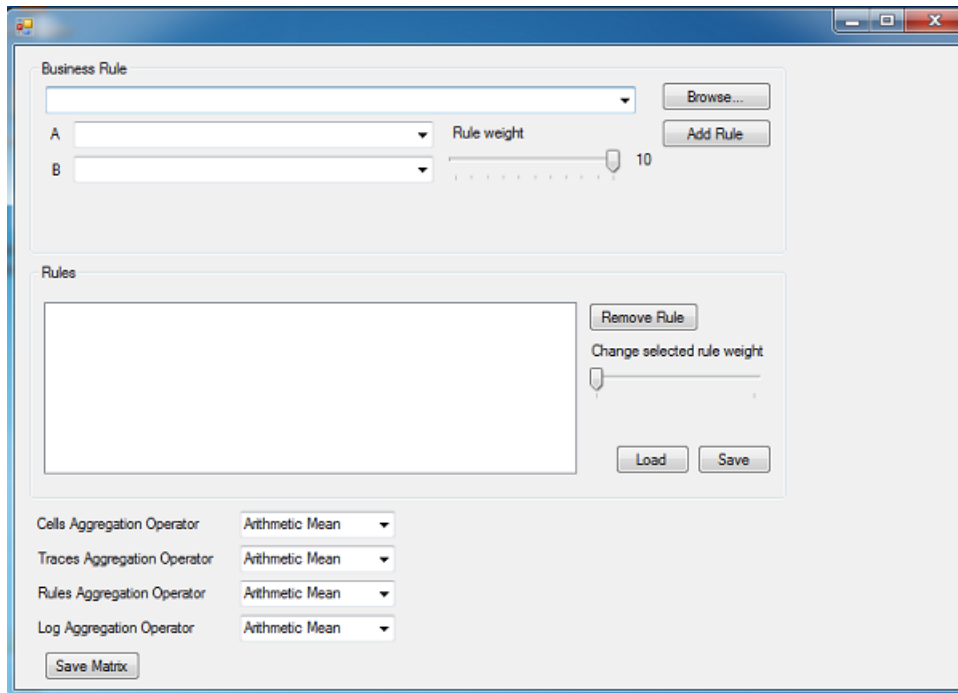


Figure 8.1: The main screen of the BRCA tool

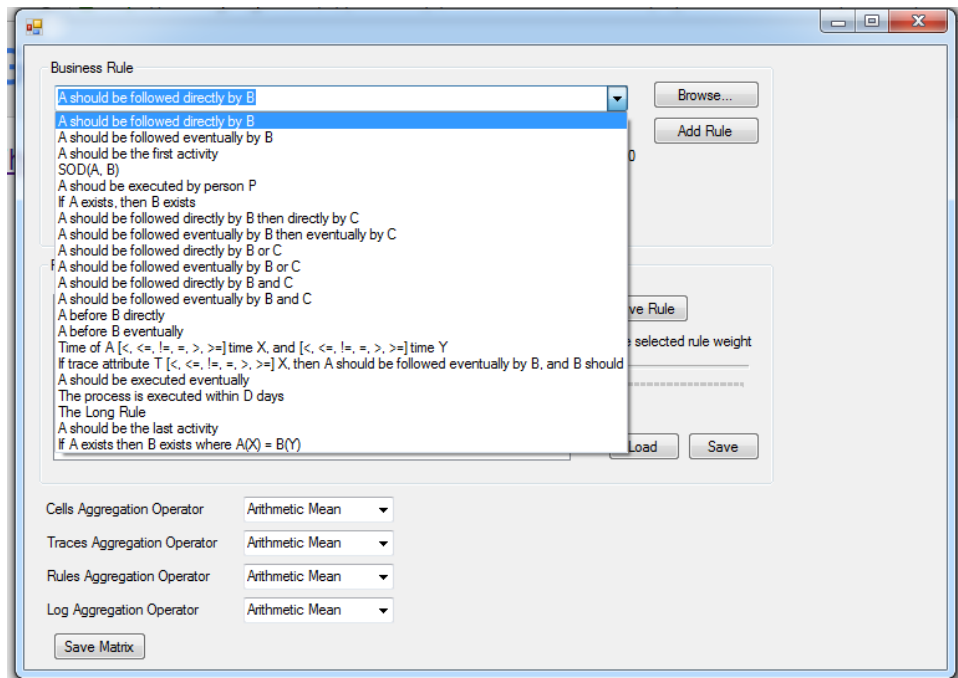


Figure 8.2: List of predefined business rules

Selecting one of the rules, the user needs to tune the parameters according to his/her needs. For instance, if the rule: *If A exists then B exists where $A(X) = B(Y)$* is selected, then the user should select the values of A, B, X, and Y among a list of values such as shown in Figure 8.3. Notice that there is no need to type the values of the parameters. Expected values are listed next to their corresponding parameters. Users can just select the values from the list.

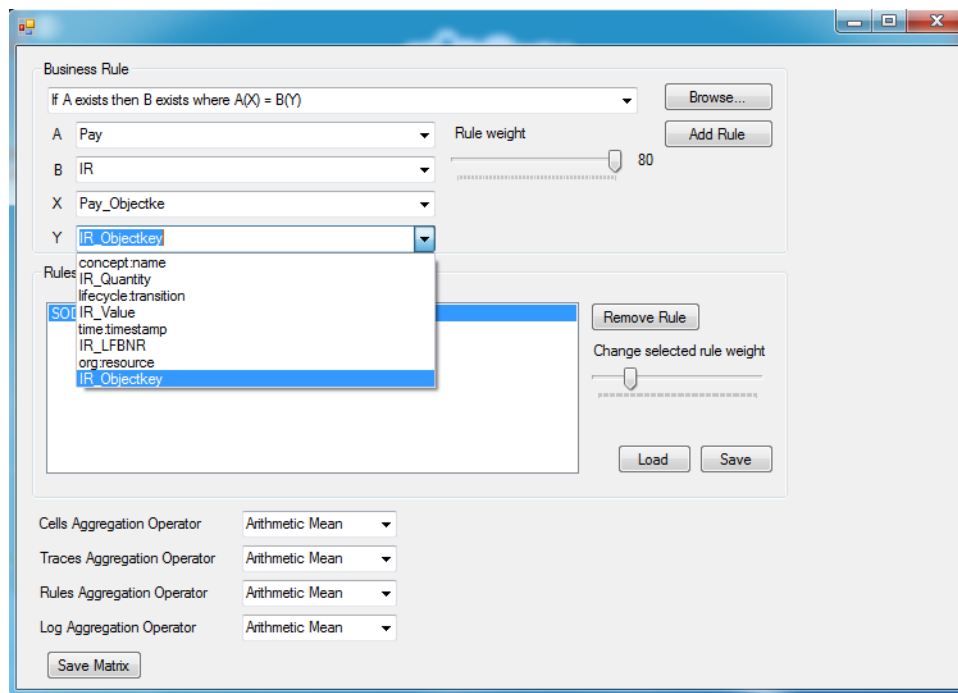


Figure 8.3: Tuning parameters according to the selected rule

Once the model is ready to use, the user needs to select a suitable aggregation operator which has a proper semantics (see figure 8.4). In case the WAM operator is selected to measure the cases fitness degree, then the user can tune the weights according to the importance of each rule.

After setting the process model, the tool can be run to measure the fitness degrees. For this purpose, the user needs to press the (Save matrix) button and choose a suitable location to save the output file. The output is an excel file contains the fitness matrix such as shown in Figure 8.6.

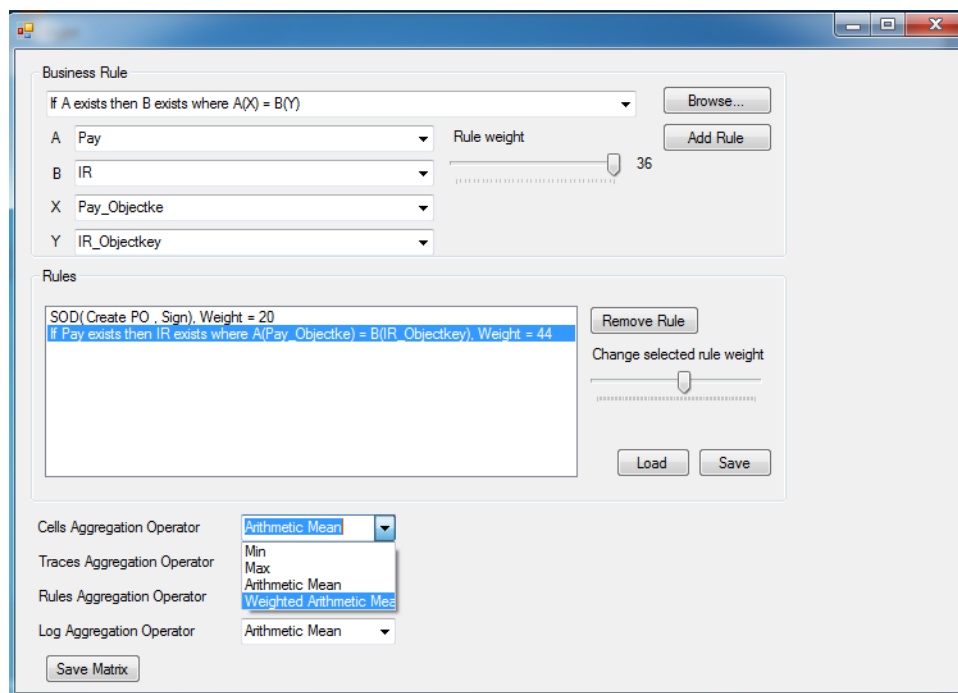


Figure 8.4: Selecting the suitable aggregation operator

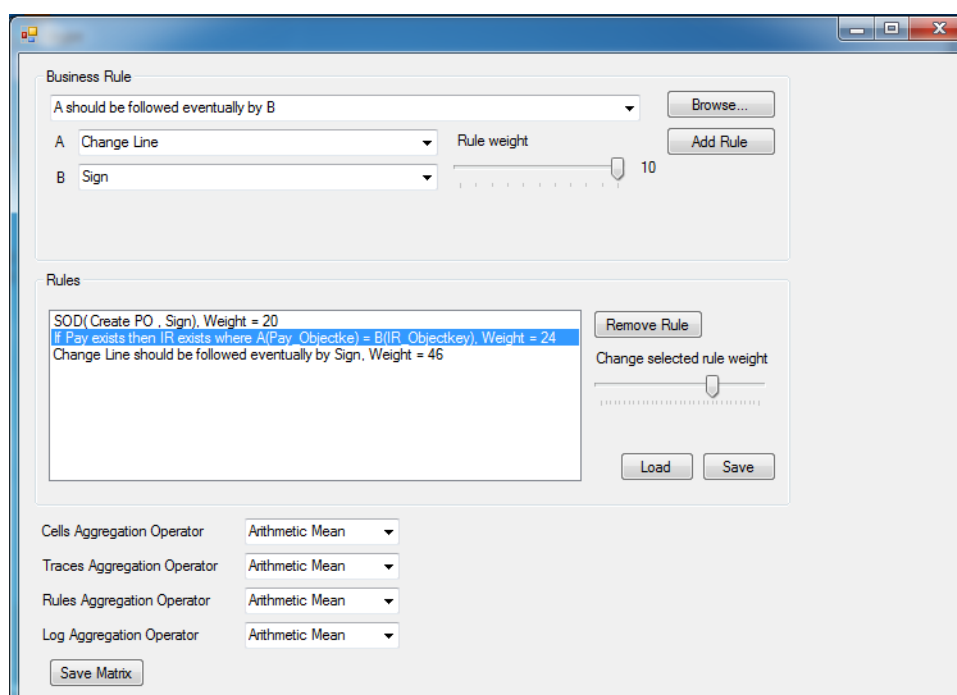


Figure 8.5: Assigning weights to WAM operator to measure the cases fitness degrees

PI ID	Rule 1	Rule2	Rule 3	Cases
45004160596630	-	1.00	1.00	1.00
450040421940	-	1.00	1.00	1.00
45004051033350	1.00	1.00	1.00	1.00
45004089553290	-	1.00	1.00	1.00
4500435563630	-	1.00	1.00	1.00
45004684822870	-	1.00	1.00	1.00
45004433724360	-	1.00	1.00	1.00
450046182710	0.00	1.00	1.00	0.67
45003974678480	-	1.00	1.00	1.00
45004240123940	-	1.00	1.00	1.00
45004238761760	-	1.00	1.00	1.00
Rules	0.60	0.98	1.00	0.93

Figure 8.6: Fitness matrix

Chapter 9

Conclusion and Future Work

Compliance auditing techniques are of high importance for process managers. They can help them analyzing the performance of their executed processes. The result is a good indicator to evaluate the organizational performance. Although existing techniques provide quantitative metrics to measure the conformity between a log file and a set of executed processes. However, they have some important limitations. The study of the literature reveals the need for a flexible approach for compliance auditing which can be customized to meet the process analyst needs. Some major factors have been underestimated in the exiting work such as the flexibility of execution, the different interpretation for a compliant process, and the ability to check the compliance at different levels of abstraction.

In this work we present our Flexible Compliance Auditing Framework to support the business world with more applicable metrics. The framework provides different semantics for a compliant process. Hence, it can be used to check the compliance taking into consideration the business context in which it is applied. The developed framework supports the business world with a fitness measurement technique which can be customized according to the process under investigation. This enables the process analysts to consider important factors affecting the procedure such as: the nature of the process under investigation, the different perspective of process analysts when interpreting the results of compliance auditing techniques, the balance between ensuring process compliance and flexible execution, and the dynamic nature of the business world.

Flexibility and compliance are conflicting terms. Keeping the balance between them is not easy. In this work, we handle the flexibility issue in two ways: firstly, we use a new declarative language called the BRCA language which we developed

to represent the compliance requirements. Hence, the process can be executed in a very flexible way. Notice here that procedural models and flexible execution never meet. Secondly, we define different functions to measure the fitness degree of executed processes with respect to the normative model. Selecting the suitable function is left to the process analysts. This way, they can customize the fitness measurement technique according to their needs.

The BRCA language is introduced to represent the compliance requirements for compliance auditing purposes. The study of the declarative languages used in the literature reveal the need for a new language. Existing languages have two main problems which make it difficult to use one of them in this research: 1) they do not provide the ability to represent cases in which a business rule cannot be checked, 2) they do not allow formulating rules that can be checked at the occurrence level. Hence, the BRCA language is introduced to overcome these limitations. Using BRCA, the process analyst can define a set of business rules. Each of which can be checked at different levels of abstraction. In addition, it provides the ability to represent when the rule is applied, violated, or cannot be checked!

Given a log file and a BRCA model, the fitness degree can be measured at four levels of abstraction: process, rule, case, and cell (one case against one business rule). These metrics are formulated using different aggregation operators. Each operator provides a different semantic. This is an important feature for process analysts. It provides them with fitness metrics that match the context in which they are applied. In addition, the proposed technique considers the different types of compliance requirements. This is another added value for using a declarative language other than flexibility.

Moreover, to the best of our knowledge, this is the first work to introduce the concept of "not checked rules". Not checked rules are those that cannot be checked in an executed process. Previously, these processes are considered either compliant or non-compliant. We believe that these cases should be treated differently. However, this concept needs more investigation. For the moment, these cases affect the result when using an averaging aggregation function.

In addition, we propose a new clustering-based approach to provide diagnostic information for compliance auditing. The idea is to cluster the log file and then check the compliance of each cluster individually and then profile the clusters to reveal common characteristics. The results provide diagnostic information about the reasons behind violations.

The developed framework is evaluated by means of a synthesized data set for

the loan approval process and a real data set for the procurement process. Results show the importance of providing different semantics to measure the fitness degree.

Future Work

Although the framework developed in this research has solved the limitations detected in the existing work, however, nothing is complete. This work can be further improved in a number of ways:

1. Extend the framework to provide diagnostic information: the framework can be extended to support the user with diagnostic information. The idea is to include the clustering based compliance auditing technique presented in chapter 7 in the developed framework. Notice that, for the moment, the two sides are not integrated. Process analysts can use each of them separately.
2. Find a neutral element which can replace the dash in the fitness matrix: currently, when a case cannot be checked against one of the rules, a dash is assigned to the corresponding cell. However, when measuring the fitness degree at higher levels, these dashes affect the result. Assume that we need to measure the fitness degree at the case level for two cases C_1 and C_2 and that the process model contains 5 business rules. Lets assume that C_1 could be checked against one of rules only and that this rule is applied in that case. C_2 could be checked against the five rules, two of them are applied and the remaining are not. That being the case, the fitness degree of C_1 and C_2 would be 1 and 0.4 respectively. Notice that the C_1 is considered completely compliant although it is applied in only one rule, whereas, C_2 is partially compliant although it is applied in two rules. We believe, as mentioned earlier, that it is important to distinguish checked cases from others. However, it should be handled in a different way when measuring the fitness degree using an averaging aggregation operator.
3. Enrich the BRCA language with the contrary-to-duty (CTD) feature. Due to the dynamic nature of the business world, violations is normal to occur. However, a violation is not always an error. Thus, CTD is used to represent a reparation chain. In this context, a reparation chain is the secondary obligation which needs to be activated as a response to violating the primary obligations.

4. Define some thresholds between 0 and 1 to categorize the measured processes accordingly. In this research we define the fitness degree to be a value between 0 and 1. Defining some thresholds can help process analysts categorizing their business processes, process instances and business rules into predefined categories according to their fitness degree. For instance, we can define the thresholds: 0.35, 0.75 to categorize cases of the procurement process. Accordingly, cases with fitness degrees below 0.35 are categorized as low compliant, cases with fitness degrees between 0.35 and 0.75 are categorized as moderate compliant and the remaining cases are categorized as high compliant. Notice that each process should have its own thresholds to keep on the flexibility feature of the work and that these thresholds should be defined taking into consideration the process analyst point of view.

Appendix: The Syntax of BRCA Language

1. Variables

1.1. Declaration

Variables are allocated on their first use and initialized with 0, there is no need to declare them explicitly.

1.2. Identifier

Variable identifier should match the following regular expression:

`[a-z]+[a-z0-9]*`

2. Expressions

2.1. Arithmetic Expression

Any arithmetic expression should satisfy the following grammar:

Arithmetic_Expression Term Expression'

Arithmetic_Expression' +Term Arithmetic_Expression' |-Term Arithmetic_Expression' |

Term Factor Term'

Term' *Factor Term' /Factor Term' |

Factor Identifier |Number |true |false |null |(Arithmetic_Expression) |(Conditional_Expression)

Number 0 |[1-9] Number'

Number' [0-9] Number' |

true and false factors evaluate to 1 and 0 respectively in arithmetic expressions.

2.2. Assignment Expression

Variables can be assigned an expression using the following grammar:

Assignment Expression Identifier = Expression

2.3. Conditional Expression

Conditional_Expression Conditional_Term Conditional_Expression'

Conditional_Expression' or Conditional_Term Conditional_Expression' |

Conditional_Term Factor Conditional_Term'

Conditional_Term' and Factor Conditional_Term' | (Conditional_Expression) |

Conditional_Term'' |

Conditional_Term'' not Factor | not (Conditional_Expression) | not (Arithmetic_Expression) |

null and number 0 are treated as false in conditional expressions, whereas any other number is treated as true.

3. Functions

3.1. Count Function

count(x in y):

x: an event type (?).

y: a trace

return: the number of occurrences of x in trace y.

3.2. Exist Function

exist(x in y):

x: an event type.

y: a trace.

return:

true: if there exists at least one event of type x in y.

false: otherwise.

3.3. First Function

first(x in y):

x: an event type.

y: a trace.

return: the first event of type x in trace y.

3.4. Follow Function

`follow(x,y):`

`x`: an event.

`y`: an event type.

return: the first event of type `y` that eventually follows the event `x`, or null if no such event.

3.5. Last Function

`last(x in y):`

`x`: an event type.

`y`: a trace.

return: the last event of type `x` in trace `y`.

3.6. Length Function

`length(x):`

`x`: a trace.

return: the number of events in trace `x`.

3.7. Next Function

`next(x):`

`x`: an event.

return: the next event in the same trace of `x`, or null if `x` is the last event.

3.8. Position Function

`position(x):`

`x`: an event.

return: the 1-based index of event `x` in the trace.

3.9. Prev Function

`prev(x):`

`x`: an event.

return: the previous event in the same trace of `x`, or null if `x` is the first event.

Bibliography

- [1] Guido Governatori and Antonino Rotolo. Norm compliance in business process modeling. In *Proceedings of the 2010 international conference on Semantic web rules, RuleML'10*, pages 194–209, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16288-6, 978-3-642-16288-6.
- [2] Wil van der Aalst, Kees van Hee, Jan Martijn van der Werf, Akhil Kumar, and Marc Verdonk. Conceptual model for online auditing. *Decision Support Systems*, 50:636–647, February 2011. ISSN 0167-9236.
- [3] Guido Governatori and Shazia Sadiq. The journey to business process compliance. In *Handbook of Research on BPM*, pages 426–454. IGI Global, 2009.
- [4] Jeremy Hope. Governance and control: Focus risk management on multiple levers of control. Technical report, IBM Corp., Canada, April 2009.
- [5] Tom Butler and Damien McGovern. Adopting it to manage compliance and risks: an institutional perspective. In Willie Golden, Tom Acton, Kieran Conboy, Hans van der Heijden, and Virpi Kristiina Tuunainen, editors, *Proceedings of the 16th European Conference on Information Systems*, pages 1034–1045, Galway, Ireland, 2008.
- [6] Patrícia Silveira, Carlos Rodríguez, Fabio Casati, Florian Daniel, Vincenzo D' Andrea, Claire Worledge, and Zouhair Taheri. On the design of compliance governance dashboards for effective compliance and audit management. In *Proceedings of the 2009 International Conference on Service-oriented computing*, pages 208–217, Berlin, Heidelberg, 2009. Springer-Verlag.
- [7] Yurdaer Doganata and Francisco Curbera. Effect of using automated auditing tools on detecting compliance failures in unmanaged processes.

- In *Proceedings of the 7th International Conference on Business Process Management, BPM '09*, pages 310–326, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03847-1.
- [8] 2015 sarbanes-oxley compliance survey: Changes abound amid drive for stability and long-term value. <http://www.protiviti.com/en-US/Pages/Sarbanes-Oxley-Compliance-Survey.aspx>, . Accessed: 2016-03-02.
- [9] Florian Daniel, Fabio Casati, Vincenzo D' Andrea, Emmanuel Mulo, Uwe Zdun, Schahram Dustdar, Steve Strauch, David Schumm, Frank Leymann, Samir Sebahi, Fabien de Marchi, and Mohand-Said Hacid. Business compliance governance in service-oriented architectures. In *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications*, pages 113–120, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3638-5.
- [10] Ahmed Awad. *A Compliance Management Framework for Business Process Models*. PhD thesis, University of Potsdam, Potsdam, Germany, May 2010.
- [11] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714, pages 149–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75182-3. URL <http://espace.library.uq.edu.au/view/UQ:34544>.
- [12] Manuel Resinas, Antonio Ruiz-Cortes, and Cristina Cabanillas. Hints on how to face business process compliance. *III Taller de Procesos de Negocio e Ingenieria de Servicios PNIS10 in JISBD10*, 4(4):26–32, 2010.
- [13] TILBURG UNIVERSITY. Compliance-driven models, languages, and architectures for services compas: D2.1, July 2008. URL http://www.compas-ict.eu/compas_results/deliverables/m06/D2.1_State-of-the-art-for-compliance-languages.pdf.
- [14] Michael Fellmann and Andrea Zasada. State-of-the-art of business process compliance approaches. In *22st European Conference on Information Systems, ECIS 2014, Tel Aviv, Israel, June 9-11, 2014*, 2014. URL <http://aisel.aisnet.org/ecis2014/proceedings/track06/8>.

- [15] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008. ISBN 978-3-540-85757-0. doi: http://dx.doi.org/10.1007/978-3-540-85758-7_24.
- [16] Alexander Forster, Gregor Engels, Tim Schattkowsky, and Ragnhild Van Der Straeten. Verification of business process quality constraints based on visual process patterns. In *Theoretical Aspects of Software Engineering*, pages 197–208, 2007.
- [17] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. Yet another event-driven process chain. In *Business Process Management*, volume 3649, pages 428–433, 2005.
- [18] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering, CAiSE '99*, pages 286–300, London, UK, 1999. Springer-Verlag.
- [19] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. On enabling data-aware compliance checking of business process models. In *ER'10*, pages 332–346, 2010.
- [20] Martijn Zoet, Richard Welke, Johan Versendaal, and Pascal Ravesteyn. Aligning risk management and compliance considerations with business process development. In *Proceedings of the 10th International Conference on E-Commerce and Web Technologies, EC-Web 2009*, pages 157–168, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03963-8. ACM ID: 1611992.
- [21] Cristina Cabanillas, Manuel Resinas, and A. Ruiz-Cortés. On the identification of data-related compliance problems in business processes. In *VI JORNADAS CIENTÍFICO-TÉCNICAS EN SERVICIOS WEB Y SOA (JSWEB'10)*, pages 89–102, 2010.
- [22] Ly Linh Thao, Stefanie Rinderle-Ma, Göser Kevin, and Dadam Peter. On enabling integrated process compliance with semantic constraints in

- process management systems. In *Information Systems Frontiers*. Springer, 2009.
- [23] Kioumars Namiri and Nenad Stojanovic. Using control patterns in business processes compliance. In Mathias Weske, Mohand-Said Hacid, and Claude Godart, editors, *WISE Workshops*, volume 4832 of *Lecture Notes in Computer Science*, pages 178–190. Springer, 2007. ISBN 978-3-540-77009-1.
- [24] Marwane El Kharbili, Sebastian Stein, Ivan Markovic, and Elke Pulvermüller. Towards a framework for semantic business process compliance management. In Shazia Sadiq, Marta Indulska, Michael zur Muehlen, Xavier Franch, Ela Hunt, and Remi Coletta, editors, *Proceedings of the 1st International Workshop on Governance, Risk and Compliance: Applications in Information Systems (GRCIS '08)*, pages 1–15, June 2008.
- [25] Stijn Goedertier and Jan Vanthienen. Designing compliant business processes with obligations and permissions. In *Business Process Management Workshops*, pages 5–14, 2006.
- [26] Zoran Milosevic, Shazia Wasim Sadiq, and Maria E. Orlowska. Towards a methodology for deriving contract-compliant business processes. In *Business Process Management*, pages 395–400, 2006. doi: 10.1007/11841760_29.
- [27] Shazia Sadiq and Guido Governatori. A methodological framework for aligning business processes and regulatory compliance. In Jan Brocke and Michael Rosemann, editors, *Handbook of business process management: 2. Strategic alignment, governance, people and culture*, pages 159–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [28] Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business processes and business contracts. In *Proceedings of the 10th IEEE Conference on Enterprise Distributed Object Computing*, pages 221–232, 2006.
- [29] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. Compliance aware business process design. In *Proceedings of the 2007 international conference on Business process management, BPM'07*, pages 120–131, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-78237-0, 978-3-540-78237-7. ACM ID: 1793730.

- [30] Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Systems Journal*, 46:335–361, April 2007. ISSN 0018-8670.
- [31] Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In Karl Aberer, Zhiyong Peng, Elke A. Rundensteiner, Yanchun Zhang, and Xuhui Li, editors, *WISE*, volume 4255 of *Lecture Notes in Computer Science*, pages 156–168. Springer, 2006. ISBN 3-540-48105-2. URL <http://dblp.uni-trier.de/db/conf/wise/wise2006.html#YuMHJHW06>.
- [32] Aditya Ghose and George Koliadis. Auditing business process compliance. In *Proceedings of the 5th international conference on Service-Oriented Computing*, ICSOC '07, pages 169–180, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74973-8.
- [33] Ahmed Awad. Bpmn-q: A language to query business processes. In Manfred Reichert, Stefan Strecker, and Klaus Turowski, editors, *Enterprise Modelling and Information Systems Architectures - Concepts and Applications*, *Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 07)*, St. Goar, Germany, October 8-9, 2007, volume P-119 of *LNI*, pages 115–128. GI, 2007. ISBN 978-3-88579-213-0.
- [34] Ahmed Awad, Sergey Smirnov, and Mathias Weske. Towards resolving compliance violations in business process models. In *Proceedings of the 2nd International Workshop on Governance, Risk and Compliance - Applications in Information Systems*, Amsterdam, The Netherlands, June 2009.
- [35] Rakesh Agrawal, Christopher Johnson, Jerry Kiernan, and Frank Leymann. Taming compliance with sarbanes-oxley internal controls using database technology. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 92–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2570-9.
- [36] Christopher Giblin, Alice Y. Liu, Samuel Müller, Birgit Pfitzmann, and Xin Zhou. Regulations expressed as logical models (realm). In *Proceeding of the 2005 conference on Legal Knowledge and Information Systems: JURIX*

- 2005: *The Eighteenth Annual Conference*, pages 37–48, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press. ISBN 1-58603-576-2.
- [37] Christopher Giblin, Samuel Müller, and Birgit Pfitzmann. From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Technical Report RZ 3662, IBM Research, 2006.
- [38] Guido Governatori and Zoran Milosevic. Dealing with contract violations: formalism and domain specific language. In *EDOC*, pages 46–57. IEEE Computer Society, 2005. ISBN 0-7695-2441-9. URL <http://dblp.uni-trier.de/db/conf/edoc/edoc2005.html#GovernatoriM05>.
- [39] Zoran Milosevic, Audun Jøsang, Theo Dimitrakos, and Mary A. Patton. Discretionary enforcement of electronic contracts. In *Sixth International Enterprise Distributed Object Computing Conference, EDOC '02*, pages 39–50, Lausanne, Switzerland, 2002. IEEE.
- [40] N Gehrke and Michael Werner. Process mining. *Das Wirtschaftswachstum*, 42 (7), pages 934–943, 2013.
- [41] Jonathan E. Cook and Alexander L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999.
- [42] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009. doi: 10.1007/s00450-009-0057-9. URL <http://dx.doi.org/10.1007/s00450-009-0057-9>.
- [43] Alessandra Agostini and Giorgio De Michelis. Improving flexibility of workflow management systems. In *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2000.
- [44] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Expressing and verifying business contracts with abductive logic programming. *International Journal of Electronic Commerce*, 12:9–38, July 2008. ISSN 1086-4415. ACM ID: 1481698.

- [45] W. M. P. van der Aalst and A. K. A. de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 121:3–21, February 2005. ISSN 1571-0661.
- [46] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, pages 82–97, 2012. doi: 10.1007/978-3-642-32885-5_6. URL http://dx.doi.org/10.1007/978-3-642-32885-5_6.
- [47] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, 47:258–277, 2015. doi: 10.1016/j.is.2013.12.005. URL <http://dx.doi.org/10.1016/j.is.2013.12.005>.
- [48] Jonathan E. Cook, Cha He, and Changjun Ma. Measuring behavioral correspondence to a timed concurrent model. In *ICSM*, pages 332–341, 2001.
- [49] Wil M. P. van der Aalst. Business alignment: Using process mining as a tool for delta analysis. In *CAiSE Workshops (2)*, pages 138–145, 2004.
- [50] A J M M Weijters, W M P Van Der Aalst, and A K Alves De Medeiros. Process mining with the heuristicsminer algorithm. *Technology*, 166:1–34, 2006. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.8288&rep=rep1&type=pdf>.
- [51] A. K. Alves De Medeiros and A. J. M. M. Weijters. Genetic process mining. In *Applications and Theory of Petri Nets 2005, volume 3536 of Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, 2005.
- [52] W. M. P. van der Aalst. Business alignment: using process mining as a tool for delta analysis and conformance testing. *Requirements Engineering*, 10:198–211, November 2005. ISSN 0947-3602.
- [53] A. Rozinat and W. M. P van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33:64–95, March 2008. ISSN 0306-4379. ACM ID: 1316257.

- [54] Internal Controls Practitioner. Sarbanes-oxley section 404: A guide for management. Technical report, The Institute of Internal Auditory, USA, Jan 2008.
- [55] Francisco Curbera, Yurdaer Doganata, Axel Martens, Nirmal K. Mukhi, and Aleksander Slominski. Business provenance: A technology to increase traceability of end-to-end operations. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems: , OTM '08*, pages 100–119, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88870-3.
- [56] W.M.P. van der Aalst A. Adriansyah, B.F. van Dongen. Cost-based conformance checking using the a* algorithm. *Bpm-11-11*, BPM Center, 2011.
- [57] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *EDOC*, pages 55–64. IEEE Computer Society, 2011. ISBN 978-1-4577-0362-1.
- [58] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. doi: 10.1002/widm.1045. URL <http://dx.doi.org/10.1002/widm.1045>.
- [59] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536, July 1985. ISSN 0004-5411.
- [60] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. Memory-efficient alignment of observed and modeled behavior. *BPMcenter.org, Tech. Rep*, 2013.
- [61] Massimiliano de Leoni, Wil M. P. van der Aalst, and Boudewijn F. van Dongen. Data- and resource-aware conformance checking of business processes. In *Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings*, pages 48–59, 2012. doi: 10.1007/978-3-642-30359-3_5. URL http://dx.doi.org/10.1007/978-3-642-30359-3_5.

- [62] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011. ISBN 3642193447, 9783642193446.
- [63] Massimiliano de Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 113–129, 2013. doi: 10.1007/978-3-642-40176-3_10. URL http://dx.doi.org/10.1007/978-3-642-40176-3_10.
- [64] Massimiliano de Leoni, Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Decomposing alignment-based conformance checking of data-aware process models. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, pages 3–20, 2014. doi: 10.1007/978-3-662-45563-0_1. URL http://dx.doi.org/10.1007/978-3-662-45563-0_1.
- [65] Natalia Sidorova, Christian Stahl, and Nikola Trcka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.*, 36(7):1026–1043, 2011. doi: 10.1016/j.is.2011.04.004. URL <http://dx.doi.org/10.1016/j.is.2011.04.004>.
- [66] Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking based on partially ordered event data. In *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pages 75–88, 2014. doi: 10.1007/978-3-319-15895-2_7. URL http://dx.doi.org/10.1007/978-3-319-15895-2_7.
- [67] Elham Ramezani Taghiabadi, Dirk Fahland, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Diagnostic information for compliance checking of temporal compliance requirements. In *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, pages 304–320, 2013. doi:

- 10.1007/978-3-642-38709-8_20. URL
http://dx.doi.org/10.1007/978-3-642-38709-8_20.
- [68] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. Where did I misbehave? diagnostic information in compliance checking. In *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, pages 262–278, 2012. doi: 10.1007/978-3-642-32885-5_21. URL
http://dx.doi.org/10.1007/978-3-642-32885-5_21.
- [69] Elham Ramezani Taghiabadi, Vladimir Gromov, Dirk Fahland, and Wil M. P. van der Aalst. Compliance checking of data-aware and resource-aware compliance requirements. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, pages 237–257, 2014. doi: 10.1007/978-3-662-45563-0_14. URL
http://dx.doi.org/10.1007/978-3-662-45563-0_14.
- [70] Vladimir Gromov. Diagnostics in compliance checking. Master's thesis, Eindhoven University for Technology, 2014.
- [71] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In Sung Y. Shin and José Carlos Maldonado, editors, *SAC*, pages 1454–1461. ACM, 2013. ISBN 978-1-4503-1656-9. URL
<http://dblp.uni-trier.de/db/conf/sac/sac2013.html#LeoniA13>.
- [72] Elham Ramezani Taghiabadi, Dirk Fahland, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Diagnostic information for compliance checking of temporal compliance requirements. In *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, pages 304–320, 2013. doi: 10.1007/978-3-642-38709-8_20. URL
http://dx.doi.org/10.1007/978-3-642-38709-8_20.
- [73] Massimiliano de Leoni, Wil M. P. van der Aalst, and Boudewijn F. van Dongen. Data- and resource-aware conformance checking of business processes. In *Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings*,

- pages 48–59, 2012. doi: 10.1007/978-3-642-30359-3_5. URL http://dx.doi.org/10.1007/978-3-642-30359-3_5.
- [74] Mahdi Alizadeh, Massimiliano de Leoni, and Nicola Zannone. History-based construction of alignments for conformance checking: Formalization and implementation. In *Data-Driven Process Discovery and Analysis - 4th International Symposium, SIMPDA 2014, Milan, Italy, November 19-21, 2014, Revised Selected Papers*, pages 58–78, 2014. doi: 10.1007/978-3-319-27243-6_3. URL http://dx.doi.org/10.1007/978-3-319-27243-6_3.
- [75] Mahdi Alizadeh, Massimiliano de Leoni, and Nicola Zannone. History-based construction of log-process alignments for conformance checking: Discovering what really went wrong. In *Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), Milan, Italy, November 19-21, 2014.*, pages 1–15, 2014. URL <http://ceur-ws.org/Vol-1293/preface.pdf>.
- [76] M. Hack. Petri net language. Technical report, Cambridge, MA, USA, 1976.
- [77] Felix Mannhardt, Massimiliano Leoni, Hajo Reijers, and Wil van der Aalst. Balanced multi-perspective checking of process conformance. *Computing (2016)*, 2016. ISSN 1436-5057.
- [78] Matthias Weidlich, Artem Polyvyanyy, Nirmal Desai, and Jan Mendling. Process compliance measurement based on behavioural profiles. In Barbara Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 499–514. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13093-9. doi: 10.1007/978-3-642-13094-6_38. URL http://dx.doi.org/10.1007/978-3-642-13094-6_38.
- [79] Kerstin Gerke, Jorge Cardoso, and Alexander Claus. Measuring the compliance of processes with reference models. In *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part I*, pages 76–93, 2009. doi: 10.1007/978-3-642-05148-7_8. URL http://dx.doi.org/10.1007/978-3-642-05148-7_8.

- [80] W. M. P. van der Aalst, H. T. Beer, and B. F. Dongen. Process mining and verification of properties: An approach based on temporal logic. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760, pages 130–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-29736-9.
- [81] Federico Chesani, Paola Mello, Marco Montali, and Sergio Storari. Testing careflow process execution conformance by translating a graphical language to computational logic. In *Proceedings of the 11th conference on Artificial Intelligence in Medicine, AIME '07*, pages 479–488, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73598-4. ACM ID: 1420822.
- [82] Diana Borrego and Irene Barba. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Syst. Appl.*, 41(11):5340–5352, 2014. doi: 10.1016/j.eswa.2014.03.010. URL <http://dx.doi.org/10.1016/j.eswa.2014.03.010>.
- [83] A. Adriansyah, N. Sidorova, and B.F. van Dongen. Cost-based Fitness in Conformance Checking. In *Proceedings of the 11th International Conference on Application of Concurrency to System Design (ACSD), 2011*, June 2011. to appear.
- [84] A K Alves De Medeiros, A J M M Weijters, and Wil M P Van Der Aalst. Using genetic algorithms to mine process models: Representation, operators and results. *Event London*, (i):1–38, 2004. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.9095&rep=rep1&type=pdf>.
- [85] A K Alves De Medeiros. *Genetic Process Mining*. PhD thesis, TU Eindhoven, Eindhoven, Nederland, 2006.
- [86] A K Alves De Medeiros and A J M M Weijters. Genetic process mining: A basic approach and its challenges. In *In Business Process Management 2005 Workshops*, pages 203–215. Springer Verlag, 2006.
- [87] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10:1305–1340, June 2009. ISSN 1532-4435.

- [88] A. Adriansyah, B. F. Dongen, and W. M. P. Aalst. Towards robust conformance checking. In Michael Muehlen and Jianwen Su, editors, *Business Process Management Workshops*, volume 66, pages 122–133. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-20510-1. URL <http://www.pubzone.org/dblp/conf/bpm/AdriansyahDA10>.
- [89] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Eindhoven University of Technology, 2008.
- [90] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *BMMDS/EMMSAD*, volume 29 of *Lecture Notes in Business Information Processing*, pages 353–366. Springer, 2009.
- [91] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.*, 152:125–142, March 2006. ISSN 1571-0661. doi: 10.1016/j.entcs.2005.10.021. URL <http://dx.doi.org/10.1016/j.entcs.2005.10.021>.
- [92] Ivo Raedts, Marija Petković, Yaroslav S. Usenko, Jan Martijn Van Der Werf, Jan Friso Groote, and Lou Somers. Transformation of bpmn models for behaviour analysis, 2007.
- [93] Lex Wedemeijer. Transformation of imperative workflows to declarative business rules. In *BMSD*, volume 173 of *Lecture Notes in Business Information Processing*, pages 106–127. Springer, 2013.
- [94] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, Cambridge, MA, USA, 2004. ISBN 0262220695.
- [95] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 383–394. Springer, 2011. ISBN 978-3-642-28107-5.

- [96] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009. ISBN 3642002838, 9783642002830.
- [97] Dirk Fahland, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of maintainability. In *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 477–488. Springer, 2009.
- [98] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *Int. J. Cooperative Inf. Syst.*, 15(4):659–685, 2006.
- [99] Guido Governatori. *Law, logic and business processes*, page 1?10. IEEE, 2010.
- [100] Guido Governatori and Antonino Rotolo. A conceptually rich model of business process compliance. In *Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling - Volume 110*, APCCM '10, pages 3–12, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.
- [101] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 1–22, London, UK, 2001. Springer-Verlag. ISBN 3-540-41865-2.
- [102] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Representation results for defeasible logic. *ACM Trans. Comput. Logic*, 2(2):255–287, April 2001. ISSN 1529-3785.
- [103] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Embedding defeasible logic into logic programming. *Theory Pract. Log. Program.*, 6(6):703–735, November 2006. ISSN 1471-0684.
- [104] Guido Governatori and Antonino Rotolo. Logic of violations: A gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215, 2006. URL http://www.philosophy.unimelb.edu.au/ajl/2006/2006_4.pdf.

- [105] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. A flexible framework for defeasible logics. *CoRR*, cs.AI/0003013, 2000.
- [106] Guido Governatori. Representing business contracts in *ruleml*. *Int. J. Cooperative Inf. Syst.*, 14(2-3):181–216, 2005.
- [107] Guido Governatori, Antonino Rotolo, and Giovanni Sartor. Temporalised normative positions in defeasible logic. In *The Tenth International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 6-11, 2005, Bologna, Italy*, pages 25–34. ACM, 2005. ISBN 1-59593-081-7.
- [108] Guido Governatori, Jörg Hoffmann, Shazia Wasim Sadiq, and Ingo Weber. Detecting regulatory compliance for business process models through semantic annotations. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 5–17. Springer, 2008. ISBN 978-3-642-00327-1.
- [109] Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In Mehmet A. Orgun and John Thornton, editors, *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, volume 4830 of *Lecture Notes in Computer Science*, pages 486–496. Springer, 2007. ISBN 978-3-540-76926-2.
- [110] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, New York, NY, USA, 2004. ISBN 052154310X.
- [111] M. Pestic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops, BPM'06*, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-38444-8, 978-3-540-38444-1. doi: 10.1007/11837862_18. URL http://dx.doi.org/10.1007/11837862_18.
- [112] Wil M. P. van der Aalst and Maja Pestic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in*

- Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
- [113] Nataliya Mulyar, Maja Pesic, Wil M. P. van der Aalst, and Mor Peleg. Declarative and procedural approaches for modelling clinical guidelines: Addressing flexibility issues. In Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2007. ISBN 978-3-540-78237-7. URL <http://dblp.uni-trier.de/db/conf/bpm/bpmw2007.html#MulyarPAP07>.
- [114] Gleb Beliakov, Ana Pradera, and Tomasa Calvo. *Aggregation Functions: A Guide for Practitioners*, volume 221 of *Studies in Fuzziness and Soft Computing*. Springer, 2007.
- [115] Radko Mesiar, Anna Kolesárová, Tomasa Calvo, and Magda Komorníková. A review of aggregation functions. In *Fuzzy Sets and Their Extensions: Representation, Aggregation and Models - Intelligent Systems from Decision Making to Data Mining, Web Intelligence and Computer Vision*, pages 121–144. 2008.
- [116] Marcin Detyniecki. Fundamentals on aggregation operators. Technical report, University of California, Berkeley, 2001.
- [117] 3tu.datacentrum website. <https://data.3tu.nl/repository/uuid:bd8fcc48-5bf3-480e-8775-d79d6c700e90>. Accessed: 2016-03-02.
- [118] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004. ISSN 1041-4347.
- [119] Prom framework. <http://www.processmining.org/prom/start>, . Accessed: 2016-03-02.
- [120] Thomas Stocker and Rafael Accorsi. Secsy: A security-oriented tool for synthesizing process event logs. In *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014)*, Eindhoven, The Netherlands,

- September 10, 2014., page 71, 2014. URL <http://ceur-ws.org/Vol-1295/paper13.pdf>.
- [121] Mieke Jans. *A Framework for Internal Fraud Risk Reduction: The IFR² Framework*. PhD thesis, Hasselt University, Diepenbeek, Belgium, 2009.
- [122] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.*, 37(2):117–141, 2012.
- [123] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18:1010–1027, August 2006. ISSN 1041-4347. doi: <http://dx.doi.org/10.1109/TKDE.2006.123>. URL <http://dx.doi.org/10.1109/TKDE.2006.123>.
- [124] Wei-bang Chen and Chengcui Zhang. A robust method for biological sequence clustering. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006: Heuristic Systems Engineering, September 16-18, 2006, Waikoloa, Hawaii, USA*, pages 286–291, 2006. doi: 10.1109/IRI.2006.252427. URL <http://dx.doi.org/10.1109/IRI.2006.252427>.
- [125] Diogo R. Ferreira. Applied sequence clustering techniques for process mining. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on Business Process Modeling*, Information Science Reference, pages 492–513. IGI Global, 2009.
- [126] Diogo Ferreira, Marielba Zacarias, Miguel Malheiros, and Pedro Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In *Proceedings of the 5th International Conference on Business Process Management, BPM'07*, pages 360–374, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75182-3, 978-3-540-75182-3. URL <http://dl.acm.org/citation.cfm?id=1793114.1793147>.
- [127] Gabriel M. Veiga and Diogo R. Ferreira. Understanding spaghetti models with sequence clustering for ProM. In *Business Process Management Workshops, BPM 2009 International Workshops, Ulm, Germany, September 7, 2009. Revised Papers*, volume 43 of *Lecture Notes in Business Information Processing*, pages 92–103. Springer, 2010.

- [128] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 109–120. Springer, 2008.
- [129] R. P. Jagadeesh Chandra Bose and Wil M. P. Van Der Aalst. Context aware trace clustering: Towards improving process mining results. In *SDM'09*, pages 401–412, 2009.
- [130] Nour Damer, Mieke J. Jans, Benoît Depaire, and Koen Vanhoof. Making compliance measures actionable: A new compliance analysis approach. In *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, pages 159–164, 2011. doi: 10.1007/978-3-642-28108-2_16. URL http://dx.doi.org/10.1007/978-3-642-28108-2_16.

