Faculty of Science and Engineering

Evaluating Business Process Compliance Management Frameworks

by
Mustafa Hashmi
B.Comp. Sci., MSc. (IT)

A dissertation submitted for the degree of
IF49 Doctor of Philosophy

Principal Supervisor: Prof. Guido Governatori
Associate Supervisor: Dr. Moe Thandar Kyaw Wynn

Business Process Management Discipline
Information Systems School
Queensland University of Technology (QUT)
GPO Box 2434, Brisbane QLD 4001, Australia

December 7, 2015

مِن القُران الحَكيم

رَبَّنَالَاتُزِغ قُلُوبَنَا بَعدَ إِذ هَدَيتَنَا وَهَب لَنَا مِن لّدُنكَ رَحمَةً إِنكَ أَنتَ الوَهَّابُ ○

سورة: (٣:٨)

[Who say], “Our Lord (Allah), let not our hearts deviate after You have guided us and grant us from Yourself mercy. Indeed, You are the Bestower.” [1]

[1] Holy Qur’an, Juzz 3 Verse 8, Tafsir Al-Tabari Vol (5), Page: 228–229

DEDICATION

To my . . .
father (Mohammad ASLAM Hashmi)
† (1937–2010)
who has (and ALWAYS will be) my inspiration

mother (SURAYYAH Begum)
this is all due to your prayers Mom

daughter (SAFA Mustafa)
† (4–29 July, 2012)
a little angel sent to us just for 25 *days*

KEYWORDS

Business Processes; Business Process Compliance; Business Process Compliance Management; Business Compliance Management Frameworks; Compliance Frameworks; Regulatory Compliance; Obligations; Norms; Normative Requirements; Norms Classification; Norms Compliance; Norms Modeling Languages; Temporal Logic, Defeasible Logic; Semantic Evaluation;

ABSTRACT

Due to the ever-increasing pressure from regulatory authorities, the demand for organisations to *stay-compliant* has increased over the past few years. In response to these demands—and to support the organisational compliance reporting activities, a plethora of compliance management frameworks (CMFs) have been developed. These CMFs offer functionalities that address the compliance problem in a variety of ways to meet organisations' specific compliance reporting requirements. Regardless of *how good and flexible* these CMFs can be, their effectiveness largely depends on the ability of their underlying conceptual and formal models to provide faithful representations of normative requirements. A CMF based on weak conceptual and formal models might not be suitable for providing any certification of compliance that is acceptable to the certifying bodies.

Given the breadth of the business process compliance domain and the existence of large number of CMFs, determining the suitability of a CMF is a difficult task. Despite that there are no methodologies that can be used to evaluate the abilities of a CMF. This thesis proposes a formal framework to evaluate *whether a CMF correctly represents* the normative requirements that a system has to comply with.

The proposed framework provides the following contributions: (i) a classification model and formal semantics for normative requirements giving a rich and improved ontology of various types of norms, (ii) systematic conceptual and formal evaluations of underlying conceptual and formal models of existing CMF that determine their abilities and shortcomings, and (iii) a deontic extension to Event-Calculus (EC), a value added contribution.

The framework has been formally defined and validated through the evaluations of existing CMFs. An example of these evaluations is presented at the end of the thesis. The developed framework is independent of any specific formalism, and can fit into any other formal language.

ACKNOWLEDGEMENT

After thanking Almighty ALLAH for all the strength, compassion and mercy, I would like to quote Nelson Mandela who once rightly said: *It always seems impossible until it's done*. Finally, this journey seems to be coming to an end. The last few years have been interesting, intriguing, yet challenging ones. Without any doubt, they have provided me with invaluable opportunities to explore the fathoms of knowledge, which I have always sought, and have challenged me to extend my own knowledge. Coming this far, however, would not have been possible without the consistent guidance, support, and encouragement of so many.

I would like to extend my heartiest gratitude to my supervisors Prof. Guido Governatori and Dr. Moe Thandar Kyaw Wynn for their excellent support during my research. Thank you! Guido for your continuous guidance, encouragement, support, and profound understanding in the difficult times that I faced in (or outside of) this research. You have always been ready to discuss any questions, and have guided me to solve the problems that I would face in the best possible way. Thank you! Dr. Moe for your kindness, encouragement, right guidance, fruitful discussions, and your patience in correcting my English grammar. You are true mentors.

Also, I would like to extend special gratitude to Prof. Arthur ter Hofstede for his guidance from the very beginning in choosing the appropriate direction for my research. My special thanks go to Dr. Ho.-Pun Lam (Brian) with whom I have had countless and fruitful discussions on defeasible logic and process compliance; these discussions gave me many valuable ideas and insights on various topics. Brian, you are truly a friend.

I gratefully acknowledge the generous support of NICTA and QUT in providing the funding to support the research presented in this thesis, which would not have been generated without their financial support. Also, my heartiest thanks go to my colleagues—and friends at NICTA-QRL: Prof. Simon Kaplan, Storm Griffin, Sarah

Sims, Liz Armstrong, Marie Dwyer, and many others; and at QUT: Raffael Conforti, Anastasiia Pika, and Chathura Ekanayake.

I would like to extend my heartiest thanks to International Student Serivces (ISS) for their profound academic writing support for improving my academic writing skills. I would also like to thank Ms. Denise Scott for her nonpareil contribution to professionally editing my PhD thesis and improving the clarity.

On a more personal note, I would like to thankfully acknowledge the encouragement of my brother Hashim Zia Ullah who, after the departure of my father, has always been my support in difficult times. I also acknowledge the prayers and support of my Mother and other family members. I am profoundly indebted to my beloved wife Samina Mustafa for her prayers, unconditional love, support, understanding and patience, especially when I worked till the wee hours. Also, I am grateful for the beautiful smile of my daughter Iffah Mustafa who brings true happiness to my life.

Last, but not least, I would like to gratefully thank all those who have directly or indirectly supported me during this whole period of study.

Statement of Authorship

“The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made."

Signature: [illegible]

Date: 07/12/2015

CONTENTS

LIST OF FIGURES

List of Tables

List of Abbreviations

BPC	Business Process Compliance.
BPM	Business Process Management.
BPMN	Business Process Modelling Notations.
CMF	Compliance Management Frameworks.
CTL	Computational Tree Logic.
DT-CM	Design-Time Compliance Management.
EC	Event-Calculus.
FCL	Formal Contract Language.
LTL	Linear Temporal Logic.
PENELOPE	Process Entailment from Elicitation of Obligations and Permissions.
RCM	Regulatory Compliance Management.
SOX	Sarbanes-Oxley.
TCPC	Australian Telcos Consumer Protection Code.
WF-net	Workflow-Nets.

Part I

Background

CHAPTER 1

INTRODUCTION

1.1 Problem Area

The fall of corporate giants such as Enron, WorldCom, American International Group (AIG) in the US and Parmalat, and Société Générale in Europe caused a severe depression in the world's financial markets. The apparent deterioration in the reliability of information as the result of bad management, the declaration of insufficient and false reporting of an organisation's values, and uncontrolled transfer of money are some reasons given for this depression. Fongon and Grillo (2004) observed that bad governance, unreliable information and non–conformance to regulatory laws resulted in a USD 5 trillion loss to organisations between March 2000 and September 2002 alone.

The collapse and ultimate closures of large companies resulted in the urgent need to design and implement new regulatory laws to control that how businesses should conduct their operations in the future. Thus, several laws—such as Sarbanes–Oxley (SOX) Act (US-Government, 2002); BASEL (series of) Acts (BCBS, 2013; SCBS, 2004); Health Insurance Portability and Accountability Act (HIPAA, 1996)—and Anti–Money Laundering regulations and monetary *de–facto* standards (such as the International Financial Reporting Standard [IFRS] (IFRS, 2014)) emerged. These had a direct impact on the operations of an ogranisation. Failure to comply with these regulatory laws can damage investors confidence, and result in financial penalties or (even) criminal prosecutions. Hence, adherence to the regulatory laws, internal controls, and other sources of compliance requirements has become a *must–to–do* activity for every organisation in the interests of transparency and more efficient operations of their daily business (Abdullah et al., 2010).

Compliance is an act of establishing and enforcing internal controls through which adherence to regulatory laws and standards is secured. Compliance requirements can stem from different sources and can be categorised into different classes—for example, regulatory compliance and standardisation compliance (as shown in Figure 1.1). Regulatory laws such as SOX, MIFID, and EuroSOX aim to impose conditions on businesses requiring them to run their operations as stipulated in these laws. In contrast, standardisation compliance rules are implemented to achieve and maintain the required levels of compatibility, interchangeability, or commonality in the operational and administrative areas of an organisation.

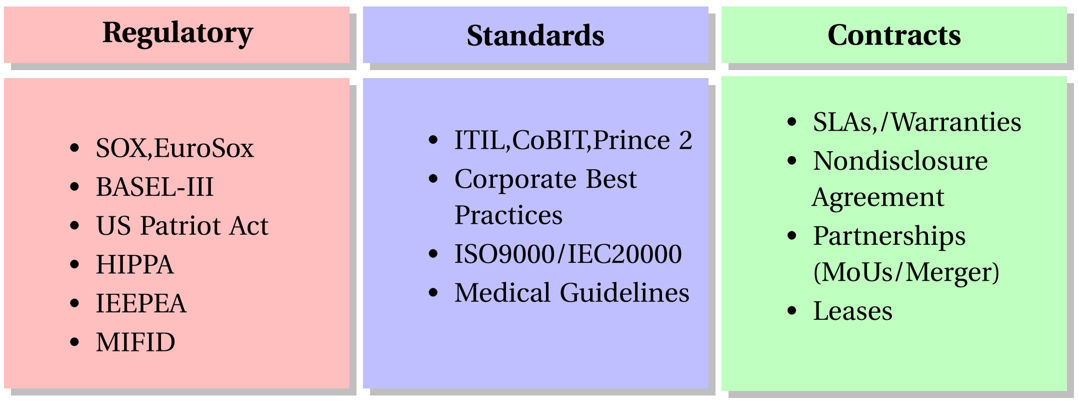


Figure 1.1: Sources of Compliance

Generally, organisations adopt standardisation compliance voluntarily to provide improved services, or to adopt procedures required by authorities for certification purposes. Business contracts, on the other hand, are agreements between organisations to achieve mutual objectives. In the framework of contracts, organisations have to fulfil responsibilities outlined in the agreed contracts. Such contracts can be in the form of customer contracts, such as warranty, service level agreement, and business partnerships (for example, memorandum of understandings or mergers). The objectives of compliance management are to identify pertinent regulatory laws; to assess the risk of derailing daily work practices because of applicable regulations; to establish the internal controls to prevent violations; and to maintain the effectiveness of these controls within an organisation (Awad and Weske, 2009; Governatori and Sadiq, 2009). Government policies alone are not the source of compliance requirements but organisations—for their own benefit, might want to implement policies and controls for better management of their business operations. Such internal control objectives limit the way an organisation is allowed to operate.

Organisations develop process models to document and automate their operational activities. These process models provide them with a high-level view of how to achieve business objectives, align IT infrastructure, estimate internal/external resources, and implement policies governing these processes. As organisations can view their activities through these models, the process models can also be used to verify the effectiveness of their internal/external regulations and governing policies (Karagiannis, 2008). This implies that business processes will be faced with correctness criteria imposed by compliance requirements (Awad and Weske, 2009). Unlike conventional correctness criteria such as different concepts of soundness (Dehnert and Rittgen, 2001; van der Aalst, 1997), and correct data flow (Sadiq et al., 2004), compliance requirements are considered as semantic constraints on process performance.

Compliance requirements vary from business to business and may be influenced by sudden changes in existing regulations or internal policy change within an organisation. These changes can significantly impact on the performance of the business process or even on the whole lifecycle of each process. This not only increases the complexity of the compliance management task, but also makes the need to develop and/or adopt an effective compliance management strategy even stronger.

1.2 Business Process Compliance Management

The term *compliance* in its literal meanings is the *ability of an object to yield elastically when a (preferably external) force is applied.* In other words, given the presence of an external force, the object has to respond flexibly without repelling the force being applied. Generally, compliance is concession to an external force (usually a government or governing body, authority, or a person), where the external force can be external to group complying, without being external to governing body (Ward, 1995). From a business process compliance perspective, McIntyre (2008) defines compliance as:

> *"A desired outcome, with regard to law and regulations, internal policies and procedures, and commitment to stakeholders that can be consistently achieved through managed investment of time and resources. The compliance management includes the legal and tactical activities in day–to–day business processes".*

The term *compliance* connects two distinct domains: the business process domain and the legal domain, as illustrated in Figure 1.2. Each of these domains has its own specificities, which are designed to achieve specific goals. The business process domain is more *prescriptive* detailing *how* the business activities should be performed. A business process can be seen as a self-contained temporal, and logical order in which a set of activities (called *tasks*) are performed to achieve a specific business objective. The business process domain is a well-researched and established domain, where researchers predominantly focus on developing and improving business process modelling approaches and languages to achieve flexibility in the business process execution (Becker and Laue, 2012; Johansson et al., 2012; Lu and Sadiq, 2007; Mili et al., 2010). In contrast, the legal (that is, regulatory) domain is *descriptive* in nature; it ascribes the legal boundaries by imposing conditions that detail which actions can be considered legal and which actions must be avoided during the execution of business process to stay compliant. Accordingly, compliance aims to gain more understanding *how* organisations *should* operate in a more sustainable way to continue providing their services while strictly observing all the applicable regulations that can significantly affect their business operations (Olivieri, 2014).

Business process and legal domains are two separate worlds, with different constructs and different goals. There is the possibility of colliding synergies between the two domains. Thus, a careful study of inter–dependencies of the domains is required (Governatori and Sadiq, 2009). For this reason, the compliance domain has received unprecedented attention from industry and academia. This attention is also motivated by recent regulations such as Sarbanes-Oxley, which requires the

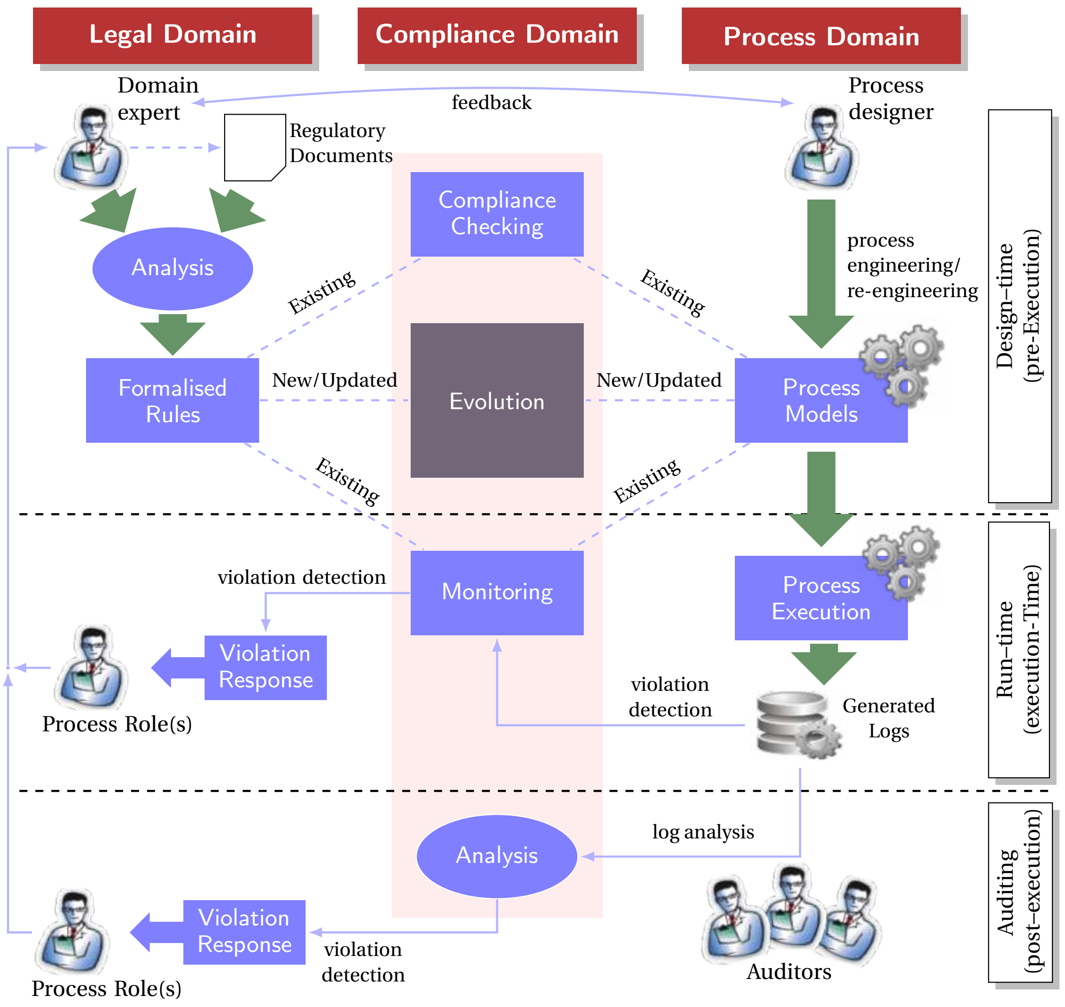


Figure 1.2: Compliance Management Strategies adapted from (Governatori and Sadiq, 2009)

establishment of stronger and more enforceable strategies to meet the organisation's compliance reporting requirements.

An organisation's process can be affected by a number of regulations or by its own internal policies. Identification of the relevant regulations causes frustration, as regulations are mostly ambiguous and require a great deal of efforts to understand. Thus, organisations are paying less attention to compliance, even while regulatory bodies require them to observe strict regulations and recommend severe penalties (or even criminal prosecution) for non–compliance (Abdullah et al., 2010; Pershkow, 2002). To avoid the problems with the regulatory bodies, organisations are putting more efforts into compliance related activities, and employ various compliance management strategies namely: *pre–execution–time, execution–time* and *post-execution time* compliance management as depicted in Figure 1.2.

Post-execution (or auditing) is a strategy by which specialised compliance consultants manually analyse the logs generated by the processes to detect possible violations. The main drawback of this strategy is the use of manual checks, which require a great deal of time and resources, and are thus a costly venture.

The increased pressure and threat of possible criminal prosecutions, however, makes it a less attractive compliance reporting approach. *Execution–time* (or run–time) compliance checking, in contrast, is a strategy by which organisations use specialised software products that produce audit reports while the processes are being executed. This approach also has limited scope because in many instances it can perform compliance checks only on a specific piece of legislation (Governatori and Sadiq, 2009). Also, it requires human intervention to manually rectify the detected problems. *Pre–execution time* (or design–time), on the other hand, is a more preventive compliance management strategy where business processes are assessed for any non–compliant patterns at the very early stages of the process design. As such, in this approach, the compliance requirements are captured through a logic–based requirements modelling framework, and propagated into business processes. Any non–compliant issues can be detected in the very early stages, thus saving an organisation efforts, time, and financial resources.

1.3 Problem Statement

In today's highly process–oriented business environment, business processes are the core of any organisation. They provide with them an abstract view of the state–of–the–affairs in relation to the achievemt of business objectives. As earlier mentioned, *to stay compliant*, organisations employ various strategies; accordingly, we list the several reasons, including the compliance reporting demands of the regulatory authorities making compliance an important activity for organisations.

Given the significance of the compliance problem, and in response to these demands, the business process management, compliance management, and computer science research community have shown a wider interest in the topic. As a result, over the last few years, a plethora of Compliance Management Frameworks (hereafter CMFs) that provide automated compliance checking has emerged (see

Chapter 2 for existing CMFs reported in business process compliance literature). These CMFs bear specific (often customisable) functionalities that address the compliance problem in a variety of ways to meet an organisation's specific compliance requirements. For example, the CMFs by (Milosevic, 2005) and (El Kharbili and Stein, 2008) focus on policy–based business process compliance, while (Schleicher et al., 2010) proposes the compliance checking approach via re–usable process fragments. A few have proposed CMFs based on defeasible and deontic logic of obligations and violation, focusing on design–time compliance management (Governatori et al., 2011; Governatori and Rotolo, 2010a; Governatori and Sadiq, 2009; Sadiq et al., 2007). Some frameworks (Cabanillas et al., 2010; Cabannilas et al., 2010), on the other hand, are data–centric, addressing the compliance problem from the data aspect of the business processes.

Regardless of the nature, types, and how good and flexible these CMFs can be, their effectiveness largely depends on their underlying conceptual and formal models that provide representations of the normative requirements. In other words the underlying formal model of a CMF needs to be sound enough to provide reasoning support for all types of normative requirements. A CMF based on weak conceptual and formal models might not be suitable to provide any certification of compliance acceptable to accredited certifying organisations. The literature on business process compliance (see, Chapter 2) shows that existing CMFs are grounded on various formal models using different formal languages to reason about the normative requirements. Given the extensibility of the business process compliance domain and the existence of a large number of CMFs, determining the suitability of a CMF for effective compliance reporting is a difficult task that requires special tools and methodologies to evaluate the abilities of the CMF. To the best of our knowledge, the business process compliance management domain currently lacks accepted tools and methodologies that researchers can rely on to evaluate the abilities of a CMF; in particular, it lacks tools and methodologies to evaluate the effectiveness of its conceptual and formal models in offering the reasoning support for various types of normative requirements. Given the lack of tools and methodologies the question is: *how can the abilities of a CMF be evaluated* to determine whether it can model all types of normative requirements, and therefore be relied upon to issue a certification of compliance. Hence, to fill this gap, this thesis proposes a formal framework to evaluate whether a CMF correctly represents the normative requirements that a system has to comply with.

1.4 Research Questions

To achieve the main goal of developing a formal framework to evaluate the abilities of a CMF, this thesis addresses the following specific research questions:

Q1. *How to evaluate existing compliance management framework, systems approaches and languages?*

The strength of a CMF largely depends on sound conceptual and formal foundations of the compliance checking approach upon which it is based. Existing CMFs address the compliance problem from a variety of perspectives (for example, from control flow, data aspects etc.) using techniques grounded on different conceptual and formal models. These models might have their own strengths and complexities, and be suitable for a specific domain or applicable under specific situations. A weak or inappropriate conceptual model can severely hinder the effectiveness of the compliance checking approach proposed in the CMF.

As normative requirements are written in natural language, different people understand and interpret them differently. For automated compliance checking, the main task is the formal representation of compliance requirements in a format that machines can understand. Generally, compliance requirements are represented using a particular formal language such as Temporal Logic, Event-Calculus, Deontic and Defeasible Logic. The formal representation with these languages provides the formal specifications of normative requirements. However, these formal languages can have varied degree of complexity and expressiveness. The effectiveness of the compliance checking approach proposed in a CMF particularly depends on how accurately the chosen language can provide the reasoning support to correctly represent different types of normative requirements in a conceptually sound way. Wrong interpretation or incorrect representation of the normative requirements can significantly reduce the effectiveness of the CMF.

Given the varying nature of the existing CMFs incorporating different conceptual and formal models, we investigate *how to evaluate existing CMFs, systems and languages* to determine their suitability to issues a certification of compliance. The business process compliance domain currently lacks tools and methods that can be used to evaluate the abilities of CMFs. In particular, we investigate *how to evaluate* *the conceptual and formal models of existing CMFs*, and *whether a CMF is able to provide compliance management support in a conceptually sound way*.

Accordingly, in order to find the answers to these questions, we first need to identify the generic classes of normative requirements for which a CMF must be able to provide full reasoning and proper modelling support. Hence, the second question of this research is:

Q2. *What are the classes of normative requirements and how can they be formally modelled?*

Compliance is about legal norms (known as *normative requirements*). Norms aim to control the behaviour of their subject by defining what is legal and what is not. Generally, norms are prescriptive and define the conditions under which they are applicable and the effects they produce when applied. The structure and properties of norms have been subject to extensive research in Artificial Intelligence and Law and Legal Reasoning with respect to various property aspects such as reification, rules semantics, normative effects, contraposition, and rules validity, to name but a few.

From a business process compliance perspective, norms prescribe conditions (otherwise 'compliance rules') relating to *how activities should be carried out and impose penalties for any divergent behaviour*. Generally, normative requirements are written in natural language and must be translated into a machine–readable format, giving the formal specifications of norms for automated compliance checking. Norms can have different structure and properties (depending upon the conditions of applicability and circumstances under which they are applicable); they can be classified according to their relevant properties. Given the different types of norms' properties, from the specifications of normative requirements for an automated compliance checking, we investigate *which are the generic classes of normative requirements and whether they can be further classified according to the relevant property?*

The main idea of business process compliance is to determine whether the constraints (normative requirements) imposed by a regulatory framework are met by IT systems. Thus, it is particularly important that a CMF offers a faithful representation of normative requirements and is able to appropriately reason with the normative requirements. A non–faithful representation of, and inappropriate reasoning with the norms, can have a significant impact on the effectiveness of a compliance checking approach in the CMF. Thus, we investigate ways *how to formally represent different classes of normative requirements as per their semantic definitions* in a proper and conceptually sound way. Currently, most approaches largely neglect the aspect whether the method they propose suitably represents the normative requirements.

1.5 Research Approach

The main objective of this research is to develop a formal framework to evaluate the abilities of existing CMFs, in particular, CMFs with a design-time compliance focus. The term *design–time compliance* refers to the compliance management approach (Governatori and Sadiq, 2009) that allows the process designers to take corrective measures at the very early stages of the process design; in this way, potential violations of the business rules can be prevented. Figure 1.3 illustrates our research approach to designing the framework.

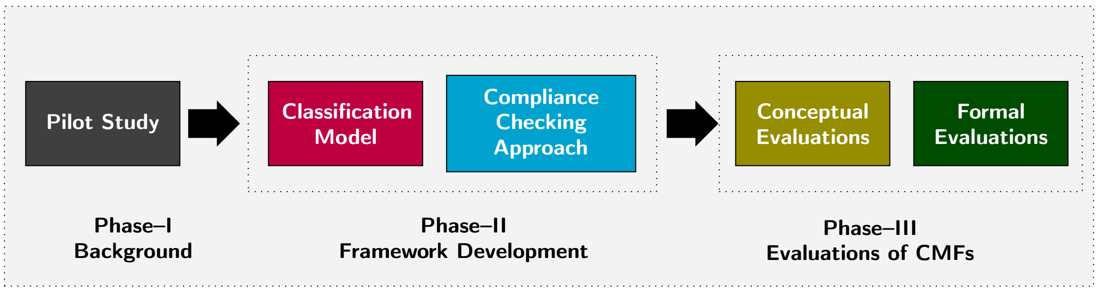


Figure 1.3: Research Approach

As a first step, we conducted a requirement analysis to identify various types of normative requirements and the functional, non–functional, and operational capabilities of CMFs. This analysis laid the foundations for a classification model that gives various classes of normative requirements using a well-known *divide and conquer* strategy. The classification model provides an exhaustive ontology of norms along temporal dimensions. The temporal criteria were selected from other properties of normative requirements; this is because generally norms have a life span, and produce effects when applied. We then provided formal semantics for each class of the classification model without restricting ourselves to any specific formalism. Normative requirements were chosen as the main criteria to the design the evaluation framework because, for any CMF, to be effective, it must be able to support all types of norms. This, in turn, is because legal documents can prescribe conditions that might be applicable under various situations and could produce legal effects when applied.

Several compliance approaches were then analysed to find a suitable model to design our compliance checking approach. We chose to use formal and case study methods. With the formal methods, formal specifications of business processes and that of normative requirements were provided. We adopted the idea of providing an intermediary mechanism to integrate these specifications from Sadiq et al. (2007), and used case study method to further elaborate on the compliance checking approach. The developed classification model and the compliance checking approach provide the foundations to evaluate *whether a CMF gives a conceptually sound representation of normative requirements.*

Finally we carried-out several evaluations to examine the conceptual and formal foundations of the CMFs whether they offer the reasoning and modelling support for various types of normative requirements. For the conceptual evaluations, we used the classification model (presented in Chapter 3) to determine the coverage of the concepts by several representative CMFs (as exemplified in Chapter 5). The representative frameworks were selected based on the expert discussions and available in literature using the methodology from (Bandara et al., 2011). The formal evaluations—built on the methods developed in Chapter 4, were carried out to establish the mappings between the language and semantics of a CMF and the semantics and definition of compliance as provided in Chapter 4. Examples of such mappings for Event-Calculus (EC) and Linear Temporal Logic (LTL) based CMFs are provided in Chapter 6.

1.6 Research Contributions

The outcomes of this research provide the following key contributions:

(1). A *classification model and formal semantics* of normative requirements that provides a rich ontology of the various types of norms every CMF should be able to provide the reasoning support for. The formal semantics defines the normative requirements and provides the foundations for representing different classes of the norms. The formal semantics is independent of any specific formalism and provides the basis for the evaluations of several CMFs.

(2). *Systematic conceptual and formal evaluations* that examine the conceptual and formal models of existing CMFs to investigate their abilities and shortcomings in terms of the formal representations of normative requirements

(3). A *deontic extension to Event-Calculus* (EC) that underlines the formal language of PENELOPE, and addresses the issues with EC predicates that prevent PENELOPE from providing full reasoning support for all types of norms. The proposed extension provides the insights into ways *to rectify the problems with a formal language*, thus enabling it to faithfully represent all types of normative requirement in a conceptually sound way.

1.7 Publications

The research conducted for this thesis resulted in the following publications:

1 Governatori, G. and Hashmi, M. (2015b). Permissions in Deontic Event-Calculus. In *Proceedings of the 28th International Conference on Legal Knowledge and Information Systems (Jurix' 15), Braga Portugal.* [Short Paper]

2 Hashmi, M., Governatori, G., and Wynn, M. (2015a). Normative Requirements for Regulatory Compliance: An Abstract Formal Framework. *Information Systems Frontiers*, pages 1–27. [Online First]

3 Governatori, G. and Hashmi, M. (2015a). No Time for Compliance. In *Proceedings of 19th IEEE the Enterprise Computing Conference (EDOC'15), Adelaide Australia.*

4 Hashmi, M. (2015). A Methodolgy for Extracting Legal Norms from Regulatory Documents. In *Proceedings of 8th International Workshop on Evolutionary Business Processes (EVL-BP 2-15), co-located with EDOC 2015, Adelaide Australia.*

5 Hashmi, M., Governatori, G., and Wynn, M. T. (2014). Modeling Obligations with Event-Calculus. In *Proceedings of 8th International Symposium, RuleML 2014, Prague, Czech Republic*, pages 296–310

6 Hashmi, M. and Governatori, G. (2013). A Methodological Evaluation of Business Process Compliance Management Frameworks. In Song, M., Wynn, M. T., and Liu, J., editors, *Asia Pacific Business Process Management*, volume 159 of *LNBIP*, pages 106–115. Springer, Switzerland

7 Hashmi, M., Governatori, G., and Wynn, M. T. (2013). Normative Requirements for Business Process Compliance. In *Proceedings of 3rd Symposium (ASSRI'13) on Service Research and Innovation, Sydney, Australia*, pages 100–116

8 Hashmi, M., Governatori, G., and Wynn, M. T. (2012). Business Process Data Compliance. In *Proceedings of 6th International Symposium, RuleML 2012, Montpellier, France*, pages 32–46

The following papers are under review:

9. Hashmi, M., Governatori, G., and Wynn, M. T. (2015b). Norms Modelling Constructs of Business Process Compliance Management Frameworks: A Conceptual Evaluation. *Enterprise Information Systems Journal.* [Submitted]

1.8 Structure of the Thesis

To conclude this chapter, we describe the structure of the thesis. The thesis is divided into three parts namely: *background, framework development* and *evaluation of CMFs*—each having dependencies between various chapters.

Part–I: Background

The first part consists of two chapters. Chapter 1 is dedicated to a discussion on business process compliance management (see, Section 1.2). Following the pilot study in Phase 1, we formulated the research problem to develop a formal framework in Section 1.3. Chapter 2 presents a detailed discussion on the state–of–the–art in the business process compliance domain structured around the compliance management strategies, and lists generic compliance management requirements.

Part–II: Framework Development

The second part consists of two chapters. Chapter 3 proposes a classification model of normative requirements based upon the temporal validity and the effects of norms giving a rich ontology of the norm classes, see Section 3.3. A formal semantics defining (and required to model) different classes of norms is discussed in Section 3.4. Chapter 4, then provides a compliance checking approach detailing the steps required to properly model the legal component of compliance. The developed framework comprising the ontology, the formal semantics of norms, and the compliance checking approach addresses the problem formulated in Chapter 1, and provides the foundations for conceptual and formal evaluations in Part 3 of the thesis.

Part–III: Evaluations of CMFs

The third part also consists of two chapters. Chapter 5 reports on detailed methodological evaluations of seven CMFs selected based on a pre-defined set of evaluation criteria to examine their conceptual foundations to deal with the normative requirements. Chapter 6, then presents the formal evaluations of Linear Temporal Logic (LTL), and Event-Calculus (EC) based CMFs. Specifically, we investigate whether they are able to provide sound reasoning support for the classes of the classification model and formal semantics proposed in Chapter 3 and Chapter 4 respectively. Also, it proposes a deontic extension to EC as a value added contribution.

Finally, Chapter 7 concludes the thesis by summarising its contributions and limitations, and sheds some light on the avenues for future work.

CHAPTER 2

STATE-OF-THE-ART

Apart from many challenges that come from different sources of compliance requirements, the most prevalent challenges when considering *how technology* might help enterprises to deal with the compliance problem are namely:

1. how to exhibit compliance of business processes against the governing rules,
2. how to handle ever-changing regulatory requirements, and
3. how to maintain business agility in dynamic business environments governed by the regulations.

To deal with these challenges—and to meet their compliance reporting requirements, enterprises employ different compliance management strategies. Governatori and Sadiq (2009) provides a very useful discussion on such compliance management strategies namely: (i) preventative compliance detection (*that is, design-time or before-the-fact*); (ii) run-time automated detection; and (iii) retrospective or backward compliance checking (that is, *after-the-fact*). These strategies lead the emergence of several CMFs, methods, approaches, and systems. Table 2.1[1] categorises some of the existing CMFs that are reported in business process compliance (BPC) literature. These frameworks address the compliance problem in a variety of ways and offer different functionalities. However, in the BPC literature, there is no single study that systematically evaluates the *legal*

[1]This is not an exhaustive list of all representative frameworks in their respective category.

Table 2.1: Business Process Compliance Research Canvas

Compliance Frameworks	Liu et al. (2007), El Kharbili et al. (2008), Karagiannis et al. (2007), Bonazzi and Pigneur (2009), Ly et al. (2012), Yip et al. (2007), Namiri and Stojanovic (2008a), Sadiq et al. (2007), Governatori and Milosevic (2005), Namiri and Stojanovic (2007a), Namiri and Stojanovic (2007b), Hoffmann et al. (2009), Schumm et al. (2010), Jiang et al. (2014)
Design-Time Compliance Management	
Logic Based Approaches	Governatori and Milosevic (2005), Governatori et al. (2006a), Milosevic et al. (2006a), Milosevic et al. (2006b), Goedertier and Vanthienen (2006c), Governatori and Rotolo (2010b), Governatori and Rotolo (2006) Governatori and Rotolo (2010a), Governatori et al. (2011), Letia and Groza (2013), Lomuscio et al. (2008)
Patterns Based Approaches	Han et al. (2007), Yu et al. (2008),Schmidt et al. (2007), Yu et al. (2006), Namiri and Stojanovic (2007a), Förster et al. (2005, 2006), Wang et al. (2014)
Run-Time Compliance Management	Keller and Ludwig (2002), Milosevic et al. (2002), Kabilan et al. (2003a), Kabilan et al. (2003b), Leitner et al. (2009) Leitner et al. (2010), Giblin et al. (2005), Alberti et al. (2007), Governatori and Rotolo (2008a), Bai et al. (2009), Gilliot and Accorsi (2009), de Moura Araujo et al. (2010), Birukou et al. (2010)
Compliance Auditing Approaches	van der Aalst et al. (2005), de Medeiros and van der Aalst (2005), Doganata and Curbera (2009), van der Aalst et al. (2010), Arya et al. (2010), Agrawal et al. (2006), Johnson and Grandison (2007)
Hybrid Approaches	Ghanavati et al. (2007), Sapkota et al. (2011), Cunningham et al. (2001), Rifaut and Dubois (2008), Kähmer et al. (2008)

requirements to properly model the legal component of business process compliance. In this chapter, we review the existing literature on business process compliance management, and divide the presented state-of-the-art literature into two parts: (i) *the literature on business process compliance management frameworks*, and (ii) *the literature on studies that evaluate existing compliance management frameworks and highlight their shortcomings.*

This chapter is organised as follows: Section 2.1 discusses the CMFs from an organisational perspective, and highlights the strengths and weaknesses of different techniques they use in these CMFs followed by a detailed discussion of design-time compliance CMFs in Section 2.2. Run-time approaches categorised into run-time monitoring and run-time detection approaches are discussed in Section 2.3. Process mining and database technology-based compliance auditing approaches are discussed in Section 2.4. Hybrid approaches incorporating a mixer of different techniques are discussed in Section 2.5. Then, a detailed discussion of existing studies that evaluate different features of CMFs and highlight the shortcomings of these evaluations is given in Section 2.6. This chapter concludes with some remarks on the shortcomings of the existing works, thus positioning the scope of the research presented in this current study in Section 2.7.

2.1 Compliance Management Frameworks

Generally non-compliance with regulatory laws or internal control policies is categorised as a risk factor. Non-compliance with regulatory laws can lead to severe financial penalties, criminal prosecutions, or even business closures (Abdullah et al., 2010). Hence, it is inevitable for businesses to track a risk type, record it, and develop policy control that guarantees the compliance. Thus, compliance management becomes a mandatory activity for every enterprise. Many dedicated research efforts have proposed compliance management frameworks that address the issues of tracking and documenting the compliance requirements.

2.1.1 Organisational Compliance Requirements Management

More and more enterprises are both venturing globally and even incorporating new technologies to provide a wide range of, and better services to their customers. On the one hand, such new business ventures and the use of new technologies increase the customer base of an enterprise; on the other, they bring new challenges from an internal management and regulatory perspective because of the increased role of compliance in their processes. Several research efforts have focused on the ever-increasing compliance requirements of enterprises. The COSO standard (COSO, 1994) provides the guidelines for establishing business objectives, and for integrating compliance requirements into business processes for effective operations. However, the standard neither proposes a compliance model nor it describes any compliance controls.

The OCEG's[2] governance and risk compliance (GRC) (OCEG, 2012) and CoBIT (COBIT, 2007) initiatives provide governance models for enterprises operating in specific domains. For example, the CoBIT standard provides the governance models for establishing, refining and concreting the control objectives for effective and efficient management of IT resources and operations in large enterprises. However, these initiatives are not meant to suggest ways in which *to define and correlate the compliance concepts, and to integrate them into their business processes* (see Elgammal, 2012); they simply provide guidelines for managing and refining the compliance requirements.

The major threat of non-compliance is the risk of financial loss and/or loss of trust, which can lead to drastic consequences for enterprises. Effective risk management is one of the key determinants of compliance, and minimising the operational risks has been highly emphasised in the COSO framework. Ashby (2008) suggests that the adoption of a '*process–based–comprehensive*' approach to effective risk management. However, Ashby also suggests that the thoughful management of risk does not only encompass the adoption of a comprehensiveness of a process-based approach—a carefully selected GRC framework is also inevitable to ensure that all business processes are fully integrated in order to manage the risk effectively and efficiently. Evans (2014) discusses the adoption of an end-to-end process-based approach to the management of risk. To improve business agility Evans also signifies the importance of choosing the right GRC framework for the successful management of risk. The study lists eight determinants from the OCEG's capability model—such as *context, organisation, assessment of threats* and *opportunities* etc., for aligning the business strategy with the business processes for effective management of compliance related risks at various levels of an enterprise.

[2]OCEG: Open Compliance Ethics Group, available at `http://www.oceg.org/` retrieved 23rd January 2014

Vicente and Mira da Silva (2011) proposed a rather similar compliance model based on the GRC framework.

A few studies conceptualise the risks and business processes, and proposed conceptual models for managing and connecting the compliance controls into business processes; for example conceptual models proposed by Strecker et al. (2011), Namiri and Stojanovic (2007a), Rosemann and zur Muehlen (2005), and Namiri and Stojanovic (2008b) to name but a few. These studies identify several business artefacts that represent the varying segments of business operations such as processes, accounts, control objectives, and risks—and their relationship to business process compliance. Namiri and Stojanovic (2008b) listed a set of properties of the internal control systems of an enterprise, contend that these properties can minimise (or even remove) the risk of non-compliance, thus increasing the likelihood of promoting, rather than inhibit the business. Accordingly, Rifaut and Dubois (2008) use a real business case study using BASEL-II (SCBS, 2004) from the financial sector for managing the operational risks, and proposed a goal-oriented approach to assessing the regulatory requirements for business processes.

2.1.2 Static Compliance Checking Frameworks

Static compliance checking is concerned with the techniques associated with a thorough analysis of the behavioural properties of a system to investigate whether a property satisfies applicable requirements is performed. For static compliance checking it is not necessary for the system to be fully functional and running. This also implies that such techniques can be applied to the properties, that are in an intermediate (potentially incomplete) state. The static compliance checking method provides several benefits over its counterpart, traditional dynamic (or run-time) compliance checking. This is because static techniques can frequently produce counter-examples from the violations, and allow the posing of '*What—If*' question (James and Jonathan, 2011). This, in turn, facilitates a greater understanding of behavioural properties, and a detailed analysis to rectify the potential problems. Model checking and the design-time compliance checking approaches fall into the category of static checking methods.

Liu et al. (2007) proposed a static compliance checking framework that uses a static method to check business process models against business rules. They employ a classical model checking approach and used high-level specifications languages such as BPEL and BPSL. Their approach enables the formalization of a business process model with π–calculus and transforms them into a finite state machine (FSM) representation. In the case where process modeller discovers a non–compliant process, the counter–examples are automatically created at process design level. This makes the compliance checking process rather easier and less error prone, thus reducing the risk of non-compliant operations. This framework provides effective support for compliance checking, as the process designers can immediately react to any non-compliant behavior; however, this support is limited to run–time only which limits the scope of this framework. In case of violations, furthermore, the report need to be in a more meaningful and readable format, even for non–expert users. However, it is unclear how transparent the compliance checking is. Similarly, it is also unclear if π -calculus accurately represents the mapping between the process models and compliance rules, and the subsequent transformation into FSM representation.

Nishizaki and Ohata (2013) propose a rather similar approach for checking the compliance of business processes for information systems, using the UPPAAL (Pattersson and Larson, 2000) model checker. The business processes are defined as timed automata, and the regulatory rules are translated into computational tree logic (CTL) specifications, which are then fed to the model checker. The model checker automatically searches for all execution paths to verify the compliance of the rules. Where some rules prove to be non–compliant, the model checker provides a counter–example against the violated regulations as transitions traces. The use of the timed automata in this approach allows the specification and verification of queries using a real-time clock variable to represent the timed constraints. This makes the model checker suitable for verifying the compliance of real–time systems. This approach differs from Liu et al. (2007) in terms of the underlying formalism used to model the regulatory rules. Liu et al. employ BPSL for the specifications of rules, while Nishizaki and Ohata (2013) use timed automata and timed CTL. Despite the fact that use of real–time automata allows for a description of the real-time properties of the system, this latter approach is less practical because of its limited capabilities in defining and verifying the compliance issues. It does have the advantage, however the model checker generates counter-examples to make corrections in the model; this is not possible with Liu et al. (2007) framework.

2.1.3 Policy-Based Frameworks

In Chapter 1 we discussed the fact that regulatory documents alone are not the only source of compliance requirements. Organisations can also implement their own policies for transparency and effective management of their business operations. Namiri and Stojanovic (2008b) provides a taxonomy of properties that organisations can use to verify their internal control systems. An organisation's internal control systems are generally responsible for implementing the external compliance requirements. If these internal control systems are compliant, it is relatively easy for enterprises to satisfy the external compliance requirements. Some key policy and internally–based compliance management approaches are El Kharbili et al. (2008); Governatori and Milosevic (2005); Namiri and Stojanovic (2008a); Sadiq et al. (2007).

The framework reported by El Kharbili et al. (2008) defines and integrates compliance requirements by means of policies within an enterprise. Due to the vertical nature of the compliance problem, the authors define the semantics of several enterprise models and enriched them with compliance requirements modelled as elements of the policy ontology. As enterprise models and compliance management models are two distinct notions, a synergistic relation between these two notions is hence mandatory to achieve compliance. The El Kharbili et al.'s framework proposes the integration of compliance requirements into the enterprise goals and strategies to provide a better understanding of compliance at different levels; for example, operational processes and business objectives levels. The multi–layered approach introduces mandatory transformations of the different components defined in each layer. The use of business rules as a source to realize and monitor the compliance requirements on a process model has also been proposed. In contrast, the work of Karagiannis et al. (2007) sees the compliance as more of an enterprise-wide problem than a project based issue. They propose a method to link regulatory laws to business processes supported by the ADONIS platform and the SOX portal. The usability of proposed method was verified by implementing the regulatory rules from SOX Act, and claimed that a significant degree of compliance at run-time was achieved.

Bonazzi and Pigneur (2009) used a holistic layered approach to deal with the compliance management problem to achieve agreement among all related regulations, policy controls and stakeholders. The layered approach first identifies all pertinent regulations, conflicts between the involved parties and then the level of the parties' compliance with the rules requirements. The authors used IT solutions and proposed a model to track down the compliance requirements. All business units are required to divulge compliance based on the established relationship in the previous step. The proposed model establishes control among all involved activities to achieve compliance. This framework provides an effective solution to the compliance problem by defining the relationship between all stakeholders at top management level and the relevant regulations; however, the framework does not appear to cover the whole business process model of an enterprise. In addition, no compliance requirements have been specifically generated. This gives the rise to question: How will the processes of each business unit comply with the regulatory laws?

2.1.4 Internal Controls-based Frameworks

Namiri and Stojanovic (2008a) presented a formal framework to define and relate an enterprise's internal controls to ensure that business operations are in alignment with the regulatory requirements. The proposed framework first identifies internal policies and controls and validates their inter-dependency against governing rules. These controls are then formally defined using a semantic approach. The framework provides the support necessary to verify whether a system implements the required set of rules, and establishes the relationship among the business processes, and remains consistent during its evolution. The formal model introduced in this framework provides a rich formal representation of risks, involved entities and their semantic relationship with business processes and controls; however, it only captures the entities involved in the process, not the internal syntax and semantics of each entity. Furthermore, the capturing of the interdependence and contradiction of semantic relations is not possible; that is the formal model is not capable of automatically detecting any contradicting and interdependent controls. This identification of contradicting controls with respect to every entity is highly desirable. It is necessary for the gathering of all information relating to the semantic relationship between entities and/or processes in order to determine whether a process is compliant with a set of rules. Another problem with this framework is that it does not provide a fully automated solution to the compliance problem as some tasks are carried out manually while defining relationship between processes and internal controls. There is also no evidence to suggest that this framework identifies and proposes remedial actions when a rule is violated, and this restricts its scope.

Since compliance aims to align the business process specifications and business rules specifications which are two distinct worlds, the idea of maintaining a separate controls directory in order to align the business practices with controls objectives was coined in Sadiq et al. (2007). The proposed framework allows a formal representation of control objectives in formal compliance language FCL (cf. Governatori and Milosevic, 2005), and links these control objectives to processes in the form of control tags. These control tags, which can be derived from the FCL to analyse and visually annotate graph–based process models. The analysis of the process models enriched with these controls tags, allows redesign of compliant business processes. The control objectives are concerned with the data related to the entities involved in a process, and impose constraints on the data. A limitation of this approach is that there is *no evidence to show where the contents of the control tags will come from* (especially with respect to the data for the data tags); nor is there any evidence of the way in which the data constraints can be implemented on a business process. The same is true for other control tags for resources, temporal and control–flow tags. In addition, Sadiq et al. (2007) primarily focuses on the preventive compliance measures to check controls–related violations at design–time. However, in some situations, not all details related to a process might be available thus compliance measures can be checked only at run-time; however, no such support is provided.

Rather, similar works by Namiri and Stojanovic (2007a,b) use a pattern–based approach to managing the control directory of different actors in the compliance management process, and present in detail the relationship between a business process and control objectives. Their approach suggests remedial recovery actions that react to the violation of a control objective, and can be linked to each control objective. For the most part, these run–time compliance monitoring approaches, monitor the control objectives when processes are running, and no mechanism is provided for design–time compliance checking.

Hoffmann et al. (2009) presents a formal framework for annotated process models, and introduced the notion of clausal compliance constraints. They devised a lower–order polynomial time *I–algorithm* to check the completeness of compliance constraints as *partly exact*, *partly approximate*, or *guaranteeing* only. The proposed approach has a number of issues: (i) the *I-algorithm* does not seem to work in the presence of conflicts between the obligations, because the algorithm operates in polynomial time and can only be used for checking the constraints on basic processes (that is, the processes that have no loops); (ii) from a constraint–modelling perspective, the formalism used in their work lacks expressiveness for modelling the compliance requirements (For example, modelling preference–based norms such as '*if you cannot do P, then do at least Q*', is an example of permission-based [or CTD] requirements, and such requirements cannot be modelled); and (iii) from a business process modelling perspective, the proposed framework suffers from several difficulties as data contents and temporal aspects of the behaviour of activities cannot be modelled using I-algorithm. Similarly, it is not possible to annotate the predicates that represent the qualitative properties of the data. Accordingly, the support for the temporal behaviour is limited only to *what is encoded in the control–flow* of the process. Hence, it is not possible to quantitatively measure that how long an activity takes to complete, and it cannot be formally expressed.

Schumm et al. (2010) use the idea of re–using business processes, and introduced the notion of compliance fragments to embed them into a business process at design-time. They combined the formalism of compliance requirements and automated verifications of a given process in a template that can be reused for another processes. This template–based approach is less advantageous, given the varying nature of compliance requirements, and the need to add, remove, and update these requirements. For example, in the process model, when a new task (or a sub–process) is introduced and has its own requirements, then the previously stored compliant fragment has to be concretised (or even) fully re–customised because previously stored requirements might not be captured in the template fragment. In addition to that, only those specific requirements those relevant to the control-flow aspect of a business process can be handled with these compliant fragments.

2.1.5 Ontology and Semantics-based Frameworks

In the context of SeaFlows[3], Ly et al. (2012) report a compliance management framework to address the challenge of the semantic constraints condition on

[3]Semantic Constraints in Process Management Systems: *http://www.uni-ulm.de/en/in/institute-of-databases-and-information-systems/research/projects/seaflows.html*

business processes to comply with regulatory laws. The framework incorporates a graphical modelling language to capture process–related compliance rules which provides primitives to capture complex compliance rules in the form of directed graphs. In addition, it indicates the need for an independent compliance requirements repository that is maintained separately from the business process repository. The framework does not only simply provide YES/NO type answer to show compliance with a process; rather, it is capable of validating semantic constraints, and of checking compliance and the violations of compliance rules, both at design-time and run-time. The compliance support is provided in textual description of violations (log files), and is enriched with compliance rules violations and compensation activities that can be used as input to process analysis and evaluations.

While it is claimed that the Ly et al. (2012) framework provides so–called, *"life–time compliance"*, it does have its drawbacks. For example, there is no indication of how well the semantic constraints can be represented in a process model, or how the implicit constraints are derived. Moreover, the semantic constraints can be often redundant and conflicting. Furthermore, there is no indication of how the redundancies and conflicts among the semantic constraints hare handled. Theoretically, a process model, or an instance, that violates semantic constraints, might still be syntactically correct; however, this is not applicable in real situations because it is semantically incorrect. Hence, it is essential to have implicitly derived constraints free from any conflicts and redundancies for effective compliance and the challenging task of balancing the semantic constraints. Another issue with this framework arises from the validation of the consistency of semantic constraints and compliance rules across different processes. Ly et al. offer no explicit solution or technique to address this issue. Moreover, no solution for establishing and verifying the relationship between the compliance rules and a business process to achieve full compliance has been proposed.

Yip et al. (2007) discussed an ontology-based framework in the scope of an intelligent compliance management (iCMP) project to explore the application of semantic web rules and *OWL*[4] ontology to represent business domain and compliance knowledge. They, first extracted the compliance requirements and documented them semi–automatically, and then check the business rule constraints.

[4]Web Ontology Language:`http://www.w3.org/TR/owl-features/`

This approach facilitates the extraction of compliance requirements from source documents; it also deals with data incompleteness, which is a major deficiency of semantic web technologies. Ideally, the orientation of the Yip et al. (2007) framework is diverse as it offers solution to defining model compliance data; for example, policies and requirements in formal and clearly defined data structures. However, the provided support is limited to the extraction of compliance requirements from data only, as the semantic constraints imposed on other business process aspects such as control–flow, time and resources have not been supported. Moreover, the complexity of ontological mapping of compliance requirements and reasoning about these mappings, is especially challenging work, as compliance rules are expanded, removed or even updated. This can increase the compliance rules repositories quite significantly, and handling huge repositories is quite difficult task. Accordingly, with added rules, computation complexity might also increase. There is no indication of how this complexity of compliance requirements can be handled, as no support is provided for the management of the changes in the compliance rules repositories can be managed. These factors limit the effectiveness of the Yip et al.'s framework.

2.2 Design-time Compliance Management

The *design-time* compliance management (DT-CM) approaches are efficient ways of verifying the compliance in the early stages of a process design and fall into the category of 'static compliance checking' methods. Design–time approaches aim to check the compliant behaviour of business processes against all the applicable rules, thus preventing the actual execution of non-compliant processes (Kharbili et al., 2008). The design–time approaches can be divided into two sub-categories namely: (i) *design–time compliance checking*, and (ii) *design–time compliance verification*. *Design-time compliance checking* targets the implementation and checking of regulatory rules while a process is being modelled. This allows the process designers to take corrective measures in the very early stages of the process design thus completely preventing potential violations of the business rules. However, because of their rigid nature, the design-time compliance checking approaches cannot be fully automated. Thus, they are more suitable for cases such as business contracts where business processes are derived from the defined specifications (Awad and

Weske, 2009).

A number of design–time compliance checking approaches have been reported in literature, and these can be categorised based on their underlying technique(s), such as *logic-based approaches, object lifecycle–based approaches, pattern–based, and query–based approaches.* In contrast *design-time verification* is used to verify whether a designed process conforms to the policies before actual execution. Unlike design–time checking, design–time verification approaches are rather flexible in nature and allow a higher degree of automation. The rest of this section gives a comprehensive view of the reported literature related to both these sub–categories of design-time compliance management, and discusses their languages, tools, systems, and formal approaches in these categories.

2.2.1 Logic–based Approaches

Governatori and Milosevic (2005) present a formal system for describing contracts in terms of deontic concepts of obligations, permission, and prohibitions. The formalism supports reasoning about violations of the obligations of business contracts. The proposed formalism lays the foundation for contract specification language known as business contract language (BCL). Later, the same authors extended their formalism and proposed FCL (Governatori et al., 2006a), a new business contracts modelling language to check the compliance of business processes and business contracts. They used logic-based formalism for the expression of contracts and their violations, coupled with new semantics specifically developed for compliance checking. These semantics can help in determining the current state of affairs; that is *'ideal', 'sub-ideal'*, and *'non–ideal'*, when comparing business processes and contract conditions. However, these semantics support relatively simple normative expressions in which deontic constraints are expressed as single events; the support for rather complex events relationships is very limited. In addition, the handling of deadlines in FCL obligations modalities is poorly expressed.

Milosevic et al. (2006a,b) use FCL and achieved compliance in a progressive manner. Initially, collaborative interaction or contract framing behaviour among all involving parties is identified; then, internal process compliance and the contract behaviour for each party are determined. Different heuristics are applied at this point to reflect different contract conditions, and to specify a set of actions to be taken when a violation occurs. The likelihood of contract violations is then checked at supplementary stages of a process design.

Goedertier and Vanthienen (2006c) achieve design-time process compliance using a rules set of permissions and obligations (deontic logic). They proposed PENELOPE (process entailment from the elicitation of obligations and permission), a declarative language to elicit business rules imposed either by internal policies or external regulations in the form of temporal deontic expressions. These expressions are used to generate compliant processes covering control–flow and temporal constraints among activities in a business process. Aiming to achieve compliance at design–time, PENELOPE focuses on the verification and validation of a process model at design-time, and does not intend to apply deontic rules at run–time. The proposed language has limitations, however. A major issue arises from the underlying formalism the Event–Calculus (EC) used for modelling the obligations and permissions. Furthermore, the language can only model a subset of obligations types. Governatori and Rotolo (2010b) proposed Process Compliance Logic (PCL), an extension of FCL (Governatori et al., 2006a), for capturing various types of normative requirements. The proposed logic is based on defeasible logic (cf. Nute, 2003), and deontic logic of violations (cf. Governatori and Rotolo, 2006), which transforms the deontic obligations subject to a business process into normal forms, and represents them as a PCL expression. These PCL constraint expressions for deontic systems define a behavioural and state space to identify the differences between the process execution paths and the PCL constraints. To test the effectiveness of the PCL, the authors used a three-step compliance–checking algorithm that they previously proposed (Governatori and Rotolo, 2010a). In Governatori et al. (2011), also present a formal approach, using defeasible logic to integrate the business policy constraints and the organisational goals in such a way that allows a business process to simultaneously fulfill the policy constraints and its organisational goals. As is the case with Governatori and Rotolo (2010b) study primarily deals only with the control-flow aspects of a business process, the data, resources, and temporal aspects are not addressed.

Letia and Groza (2013) report a logic–based model–checking approach for compliance verification of the integrated business process models. The proposed approach extends the norm temporal logic of Ågotnes et al. (2007), and introduces obligation and permission operators into the temporal logic to model the various compliance requirements from HACCP standard[5] in the food safety domain. The compliance checking is performed by a four-step mechanism where, in the first step the domain knowledge—that is, the normative requirements—is translated into Norms Temporal Logic and Attribute Language with Complement (NTL–ALC) logic. Then a WF–net using a Kripke structure is generated with states that are labelled with all normative requirements, specified in the form of normative formula f pertaining to the state. Each formula f in the state in the WF–net is verified if the formula f representing the norm holds in the state. If f does not hold, the state violating the norm is added to the set of breached states. The proposed approach allows the integration of subsumption–based reasoning, with the possibility of checking the compliance of various types of norms. By the virtue of the extended logic NTL–ALC, the proposed approach allows the integration of the abstract and concrete business processes, thus making it a more explicit in representation of the compliance requirements for business process models.

Governatori et al. (2006a) addressed the problem of compatibility checking between business processes and business contracts. The compliance checking approach involves the use of logic–based formalism to express business contracts and check their violations. The authors develop a semantics approach to determine ideal, sub–ideal and non-ideal scenarios for the comparison of business process execution paths and the contract conditions. Governatori and Sadiq (2009) reported an algorithm to check the deontic modalities of a business contract against a business process. To achieve this, activities involved in the process are annotated as having certain effects. Lomuscio et al. (2008) report a rather similar approach where they use multi-agent systems to verify the contract-regulated service compositions. However, their approach only enables the checking of compliance violations and does not suggest any remedies if any violations of contract rules occur.

Governatori and Shek (2013) report a rule–based compliance checking framework–the Regorous–[6]based on compliance-by-design methodology proposed in (Governatori and Sadiq, 2009). The main aspect of the proposed methodology is to extend the business processes with formalised rules, which are then verified by a compliance checking algorithm (Governatori and Rotolo, 2008a). For this purpose

[5]The Hazard Analysis Critical Control Point System, available at `http://www.standards.org/standards/listing/haccp`, retrieved 20 Feb 2014

[6]Regorous Compliance Checker, available at `https://www.regorous.com/` Retrieved 10 October 2013.

the rules are first modelled using FCL, and are then evaluated by the above-mentioned algorithm, which uses a logic-based reasoning engine SPINdle[7] (see Lam and Governatori, 2009) to validate the rule compliance. In case any non–compliance issues are detected, the compliance checker returns the processes (along with the traces and tasks) and the rules that have been violated. The objective of the verification is to ensure that a business process complies with all pertinent regulations before the actual deployment of that process.

2.2.2 Object-Lifecycle Approaches

Küster et al. (2007) introduce the notion of object lifecycle and coverage to check whether a process model is compliant with the referenced *object lifecycle* at design–time. The proposed technique first generates a process model from one or more referenced object lifecycles. In the first step, the object lifecycle is used to generate a set of actions for the process model to identify transitions in the given object lifecycles. This ensures that invalid composite states cannot be reached in the composite object lifecycle. The order of the process model is then determined, and actions are combined with process fragments in the second and third step respectively. The process fragments are connected in the final step. This approach provides support to process designers; however, it is not fully automated as synchronization points among the process object lifecycles have to be defined manually in the case of several objects lifecycles. In addition, in some cases, the number of object life cycles can be very large, and this can increase the size of the process model. The increased size of the process models can make them difficult to handle. In this approach, no mechanism is provided to determine how the compliance checking will be affected if the size of the process models becomes relatively large; how a large number of referenced object lifecycles are taken as an input; or how compliance will be preserved if a generated process is customised. This also poses the question that of whether the dependencies between the dynamic compliance rules and alternatives be a matter of concern (that is, in achieving the correct compliance rules) when a process model is customised.

Schleicher et al. (2009) extend the work of Küster et al., to solve the issue of a synchronisation point (variability) and to preserve compliance in customised

[7]SPINdle Reasoner, available at `http://spin.nicta.org.au/spindle/`, retrieved 21 August 2013.

process models. The authors introduce an approach based on the concept of a business process template that implicitly contains compliance constraints and points of variability to prevent process designers from bothering with compliance constraints at design-time. The reported algorithm ensures that compliance constraints are not violated when a process model is customised. The problem with the algorithm is that it does not provide any mechanism to handle dependencies between the alternatives and dynamic compliance rules, as mentioned above.

2.2.3 Patterns/Graph-based Approaches

Automated compliance checking of the legal requirements of the business processes is highly desirable. These requirements are often written in natural language, and must be translated into a machine–readable format for automated verification. Generally, formal languages (such as Event–Calculus, Temporal Logic, Deontic Logic), which provide the reasoning support, are used to translate the legal requirements. However, due to their complexity, the comprehension and usability of these languages is difficult, especially for non-technical users such as process analysts and compliance experts. Thus, the usability of the formal languages is one of the main concerns for non-technical users who possess less knowledge of these languages (Elgammal, 2012). To address the usability concern of the formal languages, researchers proposed to embed the formulas in a formal language that translates the compliance requirements into easy–to–understand visual patterns or graphs. This lead to the emergence of graph/pattern based compliance verification approaches in the business process compliance domain.

Han et al. (2007) propose such a pattern–based property specification language, PROPOLS[8] for specifying temporal business rules. The PROPOLS defines a collection of properties for a service composition, with each property being a rule or a logical composition of rules that govern the ordering of the primary services within a service composition. Each rule consists of a pattern element and a scope element. Because each pattern specifies the existence behaviour of a single business activity or temporal relationship among activities, PROPOLS enables process designers to insert, delete, or rearrange processes to be compliant, based on temporal business rules. Deviations from the business rules are identified using finite state automata (FSA) to

[8]PROPOLS is an ontology-based property specification language based on PPS to specify service composition properties.

inform process designers about non–compliant behaviour. The automata are derived from a set of business rules and existing process schema. Yu et al. (2008) extend Han et al.'s work and propose a synthesis framework to generate a process models from a set of temporal business rules. The proposed approach generates a process model and a requirements model (temporal rules) to achieve intuitive specifications and *correction–by–design*. This helps the process designers to rectify design time mistakes. In addition, it also allows automated verification of semi-automatically generated process models.

Schmidt et al. (2007) discuss an ontology–based approach to representing service processes and their compliance requirements, to verify whether the designed service processes are compliant. The proposed approach employs two distinct ontologies: a process ontology defining the concepts that are needed to represent service processes and a compliance requirements ontology consisting of the concepts that represent the objectives and requirements of compliance rules. The authors report three distinct categories of compliance requirements in their model: *syntactic, semantic* and *pragmatic*. To verify the compliant process elements on the semantic requirements of service processes, a reasoner is applied. The problem with the proposed approach is that it isolates only processes whose requirements are instantiated as compliant processes; other uninstantiated processes are not included. Moreover, there is no indication of how the proposed approach deals with non-compliant processes as no remedial actions can be taken in the proposed approach.

Yu et al. (2006) introduce a compliance verification approach to BPEL schema, which employs an ontology language for property specifications. The verification process starts with a high–level description of a BPEL schema to implement in the process. Then, semantic mapping between the operations is defined in the ontology language, and a finite and deterministic *labelled transition system* (LTS) model is generated. From this LTS model, a *total and deterministic finite automata* (TDFA) is built. This includes the set of final states and error states to collect a list of all the unwanted events of each state. In the last step, verification of the compliance BPEL schema determines whether all acceptable event sequences of the BPEL schema are present in the list of acceptable sequences generated in the form of TDFA.

Förster et al. (2005, 2006) present a *pattern-driven process* approach to visually express the compliance constraints on the process behaviour. The authors use PPSL, an extension of UML activity diagrams (OMG, 2011). The activity diagrams are used to specify possible patterns that need to be applied in the business process models. This enables the process designers to have an abstract view of a possible behaviour of a business process. For example, UML activity diagram patterns that extend the edges with the stereotype $\ll after \gg$ show that it is not necessary that two activities be strictly executed sequentially. These patterns are then used to check whether business process conform, by transforming them into temporal logic systems. While, business processes are transformed into a labelled transition system defined by a semantic domain meta-model that enables the application of model–checking to ensure the conformance of the business processes to patterns. Although the proposed approach provides a flexible way for the process designers to check the quality of conformance, the approach is not free of issues. The definition of process behaviour as visual patterns at design-time is one such issue, because these patterns might depend on each other, or even might reflect contradictory behaviour. Currently, the approach does not provide any mechanism to gain (potentially prior) knowledge of the interdependencies among different patterns. Furthermore, these patterns are not able to expressively model a negation; that is, a rule might stipulate conditions that prevent some activities from ever happening, while others have already been executed. Essentially, from a business process-compliance perspective, negation is an important aspect of modelling prohibitions, however, no explicit support is provided in this framework for modelling the negations. In addition, this approach only focuses on the control flow aspect of business processes and does not provide any support for modelling and checking their compliance with the data, resources aspects of a business process.

Namiri and Stojanovic (2007a) employ a pattern-based approach to modelling an enterprise's internal controls. They build their model on the de-facto internal control standard (otherwise known as COSO[9]). In the process execution phase, a bi-directional interaction between BPM and internal control management is established. Later, all information about the current instance of the business process is enacted. In case of any violations, a recovery action (defined in the controls) is executed. The major benefit of their approach is its ability to define different controls beyond workflows, and in different environments, to reuse of the process models. However, the proposed model is not fully automated because it requires

[9]Internal Control, An Integrated Framework. The Committee of Sponsoring Organisations of the Treadway Commission COSO (1994): `http://www.coso.org/`

manual selection of a control pattern and its design on a business process corresponding to the domain specific compliance requirements. Furthermore, there is no support for handling inter-control dependencies; for example, different controls can contradict, subsume or even block the execution of other controls in a business process interaction. This signifies the need to establish a stronger correlation between processes and controls. Moreover, this approach does not support compliance verification beyond the run-time, nor does it support resource and temporal aspects of the business process.

Arbab et al. (2009) present REO Tool–kit, a channel–based coordination language for the design-time verification of business process models. The language uses modelling checking and bisimulation techniques to formally analyse the correctness of the business processes against imposed constraints. For the verification of compliant behaviour, in this approach, business process are first modelled either in BPMN (OMG, 2010) or UML (OMG, 2011) activity diagrams, which can be mapped into constraints automata. The compliance requirements are represented using Linear Temporal Logic (LTL), and then the model-checking techniques embedded into REO tool-kit are used to verify the compliance of business processes. The work reported in Schumm et al. (2010) is grounded in the REO tool–kit, where the REO is used for the automated compliance verification of business process fragments against the business constraints mapped in LTL.

Wang et al. (2014) propose a formal approach that addresses the issue of determining the compliance of the PLM[10] systems and workflow management systems by using the data of the design objects which may evolve over the various versions of the product lifecycle in the PLMs. This compliance–by–design checking approach employs the workflow nets, which are annotated by defining the version–annotated processes. In the annotated processes, the version annotations are specified with the certain tasks, as per the specifications of the access control privileges. The aim of the access control privileges is to control some operations at a particular state of the product lifecycle, which may be subject to some restrictions. Later, the semantic and syntactical properties of the annotated process are defined, and these are then used to verify the behavioural and syntactical compliance of the annotated processes by merging the version-annotated process and transformed WF–nets. A version–annotated process is considered compliant only if its

[10]Product Recycle Management (Rangan et al., 2005)

compliance properties correspond to the soundness properties of the WF–net. If the soundness properties of both nets do not match, it means that the data design object's lifecycle is not compliant. The existence of non–live tasks in the process can be one of the reasons for a non–compliant version annotated process. Remedial action(s) can be taken to correct the problem by modifying the process model or access control specification from the task. The proposed approach provides technical foundations for merging the two types of process models to create a new type of compliant WF–net. The proposed approach has fundamental issues with the annotating process. This process is semi–automated, which means that some of annotation task have to be manually performed by the domain experts. Annotating the hundreds of tasks in a process model—each task having (possibly) several related compliance rules—is a tedious task, and potentially error-prone.

2.2.4 Query-Based Approaches

Awad et al. (2008a) discuss a BPMN–Q, a *query–based* approach to compliance checking. The approach is capable of answering Yes/No questions to verify whether a process is compliant. The authors use a graph reduction technique to gain the Yes/No answer. As an execution of a query graph, the graph reduction approach splits a process graph into a set of execution paths from the first to last nodes in the graph. Then, the order of execution is determined with respect to an execution path by finding the precedence between the occurrences of nodes. In the final step, a process graph is matched to a query graph. If it satisfies all sequence flow and path edges, the BPMN–Q returns a YES to a rule representing a complaint process. In the case where BPMN–Q does not find a match, a NO is returned to convey a violation of a rule. The Awad et al. (2008a) approach provides an answer to the rule query effectively, and enables order checking between activities involved in a process. However, one problem with the graph reduction approach is that it might remove some activities that, at first glance, might not be pertinent to a query. This might include those activities which have to play a significant role in the completion of a process.

The work is later extended with the authors introducing the ways to visualise the violations of control flow ordering in the compliance rules (Awad and Weske, 2009). Again, they use structural BPMN–Q queries to express the compliance rules, which are called 'patterns'. These queries are used to find the set of process models that are subject to compliance checking in a process repository. Temporal formulas are then derived from the queries to check against the process model. In the final step, anti-patterns are derived automatically from BPMN–Q queries to report any rule violations in the process models. Because Awad et al. use a graph reduction and model checker in this later (Awad and Weske, 2009) approach, the proposed solution to derive anti-patterns queries has some limitations. The generated anti-patterns depend on the input state transition system of the process model. If the transition system is generated from a reduced process model, the resulting anti-pattern would not be usable on the original process model. Similarly, as the generated anti-patterns are given as a disproof of rule violation, it is possible that some violations are not reported by the model checker. Moreover, re–implementation of a translation software will be required in the case where some changes are made in the model checker software.

2.3 Run-time Compliance Management

Once a process model has been designed and the actual execution of the process instance is initiated, continuous monitoring of the running process instances is pivotal to detect any divergent behaviour while the processes are still running. The aim of the run–time compliance checking is to monitor the running processes continuously to check if they violate the internal controls or policies imposed on their execution. For this purpose, a process engine keeps track of the process behaviour and alerts process designers to any violations. The process designers can then take appropriate actions to rectify the detected violations. The run–time compliance checking approaches can be broadly categorised into *run–time compliance monitoring approaches* and *run–time compliance detection approaches.* These can be further classified into *monitoring–based approaches, logic–based approaches, and model–based approaches.* The rest of this section gives a short overview of some approaches from the run–time compliance–checking domain.

2.3.1 Run-Time Compliance Monitoring

Keller and Ludwig (2002) introduce an architecture to monitor service level agreements (SLAs) for dynamic electronic services, in particular, web services. In the blocked architecture, the SLA requirements are first automatically generated by the

SLA–driven system administration block, for further interaction with the web service level agreement (WSLA) monitoring environment. In the WSLA monitoring phase, the monitoring is divided into two sub-phases: (a) *measurement service*, which measures all subsets of the SLA parameters generated by system administration block; and (b) *the condition evaluation*, which obtains measured values of SLA parameters from the measurement service and verifies these parameters against guarantees specified in the SLA. During the testing, if a breach is detected, a violation trigger is invoked to alert parties involved in the SLA. This verification of the SLA parameters can be done periodically, or when a new SLA parameter is available.

Work by Milosevic et al. (2002) discusses a compliance–monitoring mechanism for electronic contracts. In their role-based architecture, the authors introduce a discretionary enforcement mediator (DEM) to measure the performance of a contract. The DEM can signal a non–conformance of a contract event if it detects any deviating behaviour of the event. The DEM maintains a separate notary block in which it collects information about each violation, and this used to endorse the execution of corrective measures. This approach provides an effective means of monitoring of the adherence to all clauses of a contract; however, the approach is not fully automated. In an extension to their work on contract management (Kabilan et al., 2003a,b) use a multi–tier contract ontology for business contracts monitoring. They deduced a contract workflow model (CWM) from the multi–tier contract ontology consisting of different types of obligations written into a contract and different spaces from which each obligation passes. These obligations are monitored for potential breaches of clauses stipulated in the contract with respect to the actual execution of identified events. While proposed model provides an automated monitoring and tracking of obligation fulfilment, some components of this model are semi–automated and so do not support fully automated compliance management at run–time. Moreover, this work focuses on control–flow compliance monitoring only; the data, resources and temporal aspects of a business process are not considered. Similar to Keller and Ludwig's (Keller and Ludwig, 2002) work there are other run–time approaches for monitoring the violations of SLAs (see Leitner et al., 2010, 2009, for more details).

2.3.2 Logic-based Formal Run-time Approaches

Giblin et al. (2005) employ a formal approach to introducing the REALM model, a model–driven compliance automation method for regulatory policy and event monitoring. The meta-model of REALM supports the expression of temporal ordering and time periods as temporal logic modalities in real–time. The domain discourse of a regulation, on the other hand, is represented by the UML model. This approach only considers the temporal aspect of a process lifecycle and neglects control-flow, data and resources aspects.

Alberti et al. (2007) introduce a declarative programming language, SCIFF, an abductive logic programming for business contracts specification and monitoring. The run–time verification of contracts is performed by means of an abductive proof procedure which supports the dynamic occurrence of events; that is, the insertion of new facts during computation, and violation monitoring. For execution time compliance checking, Governatori and Rotolo (2008a) use formal contract language (FCL) to propose their algorithm. The FCL constraints are used to define the state space and behaviour of contract policies that are used to compare the behaviour execution path of a business process. The algorithm operates in a step–wise fashion, where it first collects a set of all tasks involved in a business interaction. These tasks are then used to determine the norms triggered at run–time in the second step. Finally, compliant or non–compliant behaviour of a task is declared after comparing all tasks with normative constraints. The compliance checking reported in Governatori and Rotolo's work is an automated monitoring of the business processes to suggest remedies and/or mitigation of the control–flow deficiencies. Thus, *after–the–fact* detection does not have a preventive focus. In addition, data, resources and temporal aspects of a process lifecycle have not been considered in this work.

2.3.3 Model Checking-based Approaches

Model checking is a state–of–the–art technique where the system specifications are verified against certain properties. For a system to be compliant, all the properties must be satisfied over all possible states of the system. To verify the compliant behaviour, a model and the properties are fed into the model checker such as SPIN[11],

[11]SPIN Model Checker Available at: `http://spinroot.com/spin/whatispin.html`

NuSMV[12], UPAAL[13] etc., The model checker then thoroughly searches the model against the properties, and generates counter examples if any of the properties do not apply (Bérard et al., 2001; Mateescu and Sighireanu, 2003). Since model checking is a well–researched area, it was widely used in a multitude of domains, including the business process compliance domain. There is a wide body of proposals grounded on model checking for the verification of process models.

Bai et al. (2009) adopt a model–based approach for policy enforcement and monitoring of the dynamic behaviour of web services at run-time. Their approach defines a policy model based on the WS-policy framework. It also includes the definition of the policies and policy–sensor correlation matrix adopted from the W3C standard[14] for specifying services of policy requirements. Policy consistency support is reported in this work; however, there is no indication of how policy violations can be handled, and no remedial actions are suggested to address these violations. Gilliot and Accorsi (2009) present a light-weight *violation anticipation monitor* (VAM) architecture for a priori run–time anticipation obligation violations. Based on run-time verification (verifier module), statistical reasoning, and (linear temporal logic) LTL–based model–checking technique. VAM can answer as *'true'*, *'false'*, *'presumably true'* or *'presumably false'* to represent compliance at run–time. Remedial decisions are taken on the basis of true or false predictions (that is, where *true* means a process is compliant with all the regulations, and *false* means that the process is not compliant). However, when VAM answers *'presumably true'* or *'presumably false'*, it is up to the process owner to grant or revoke rights, or even to stop the execution of the process.

de Moura Araujo et al. (2010) present a run-time compliance checking (RTCC) technique to validate the business process with respect to the business rules. They use UML to model processes, and OCL expressions to represent the business rules. The model validation is based on the simulation of the execution of process instances based on case studies. Their simulation algorithm steps through the process model executing the actions associated with the activities with the help of the USE tool, and checking the violations of any associated business rules. Their technique can precisely detect the situations in which the compliance rules are violated, and provide feedback to the analysts about the adequacy of a business

[12]NuSMV: Symbolic Model Verification available at: `http://nusmv.fbk.eu/`

[13]UPAAL: Uppsala—Aalborg Model Checker available at:`http://www.uppaal.org/`

[14]http://www.w3.org/standards/

process with respect to the business policies. However, the evaluation criteria used in this technique do not guarantee compliance; they simply provide some assurance that the process will not fail in the most elementary situations. Another issue is that the detected errors are not corrected automatically, in the case of violation, a business analyst's intervention would be necessary.

Kazmierczak et al. (2012) introduce a state–based norms compliance model checker, the NoRMC. The proposed approach is based on the norm compliance CTL (NCCT; see Ågotnes et al., 2010), and aims to verify which agents in the process interaction have to comply with the norms of an object to hold. The normative system is modelled as Kripke structure, and the constraints are defined to verify the agent behaviour on every state during the interaction. The prohibitions, represented as forbidden transitions, are modelled as a serial relation over Kripke structure, and all the forbidden transitions are removed from the structure after its implementation. The norms compliance checker takes a model, a normative system and the CTL formulas, models the obligations, and returns the states where the formulas are satisfied, so that counter-measures can be taken to repair the violation. Currently, the norms checker's usage is only limited to modelling obligations and prohibitions.

In the context of security compliance, Rieke et al. (2014) present Predictive Security Analysis at Run–Time (PSA@R), a model-based approach for evaluating the security status of business processes at run–time. This approach integrates the formal process modelling with the simulation of process behaviour, to identify and predict violations of the security policies at run–time. The proposed modular approach operates with the control flow and security properties of the business processes as formalised views. Each view in the PSA@R system is formalised for the evaluation of security status of critical processes in the near future. For example, critical processes are formalised by a process view using asynchronous product automata (APA) (Ochsenschläger et al., 1998), and the security requirements are formalised by a security view. The compliance of the security requirements is then monitored, and potential violations (in the near future) are predicted by comparing the predicted states with the security requirements using an on–the–fly prediction method. The prediction method employs an algorithm that computes accepting states referring to the *security critical states*. These critical states are then used to check the violations of the security requirements for computed states. If any deviations from the expected behaviour are detected, an alarm is raised for a decision support or reaction. Rieke et al. (2014) have validated the effectiveness of their approach by using the security policies from the hydro–power generation domain, and this approach seems promising in terms of checking the compliance of security requirements. However, it is not clear how the security module models these security requirements or what types of security policies can be modelled. Furthermore, the proposed approach does not elaborate how the compliance of interconnected requirements can be verified. This is because the sensitive nature of the security domain means that most of the requirements have a complex interrelationship in order to ensure high degree of safety of the critical systems.

D'Aprile et al. (2010) report an annotation–based compliance verification framework for checking the compliance of business processes with legal norms. The authors extend the business processes with semantic annotations through the specification of the effects of the atomic tasks and the obligations generated from their execution. The framework borrows AI techniques for reasoning about the actions and commitments, and for the verification purpose model, checking techniques are employed using answer set programming (ASP) and Coloured Petri Nets(CPNs). For the purpose of (semi)–automated verification, the norms are translated into LTL specifications, and these specifications are then fused onto business processes. The annotated business processes are then fed into a model checker, which returns a positive answer as its output if there is no violation, or a negative answer, if a process model violates any specification. The main issue with their framework is that it provides structural compliance only; however, compliance is not about only how the activities are performed to achieve the enterprise goals but also about the tasks and the effects of the tasks on the execution of the business process. In addition, it is not clear whether the framework is able to capture all the obligation types of the norms.

Birukou et al. (2010) propose a run-time compliance governance approach in the service–oriented architecture (SOA) domain. In the first step of the approach, business process models and activities relevant to the monitoring and checking of the compliance requirements are identified by Extended Process Engine (EPE), and passed to Process Engine Output (PEO). These process models, with their unique identities, are emitted by an Apache ODE engine as input for further compliance checking. In the second step, the business level events (policies), augmented with their unique IDs, are identified and sent to the Event Process Output engine in the second step. Once the processes and the business level events have been identified, an analysis of these events and processes is conducted by the business intelligence component by creating a one–to–one mapping of events and processes for violations detection. The results of off-line compliance monitoring and compliance checking are made available to the compliance dashboard. The problem with this approach is that the framework does not provide fully automated support for compliance governance by attaching the events and generating rules for compliance monitoring. Moreover, the compliance checking is done manually. In addition, there is no indication of how the system will deal with a detected policy violation, and no remedial steps are suggested. Furthermore, only the data aspect of service processes is considered in Birku et al.'s work.

2.4 Compliance Auditing Approaches

Compliance auditing is a retrospective reporting method that enterprises use for divulging their compliance. Usually, auditing is conducted by specially hired compliance auditors, who manually audit the huge trails of system-generated log files. Auditing the large amount of log files is a time-consuming task and prone to errors. The increased pressure from the regulatory bodies and possible penalties (for non–compliance) make this approach rather less attractive. However, with detailed information about processes increasingly available in high–quality event logs, auditors no longer have to rely on a small set of samples off-line. A number of automated systems use process-mining techniques, and can scan system logs to collect evidences to determine whether business processes are executed within the given set of rules.

2.4.1 Process Mining Based Approaches

van der Aalst et al. (2005) propose a property formulation language and process mining tool that enable the verification of business process properties based on event logs. The language is based on Linear Temporal Logic (LTL), and is tailored to event logs stored in the MXML format. The format used is tool–independent of logged events and can be generated from audit trails, the transaction logs and other data sets. Currently, the language provides the support for the control-flow aspects of the business processes only, and other aspects such as are resources, data and temporal aspects are not included. At a later date one of the same authors (de Medeiros and van der Aalst, 2005) applied process-mining techniques in the security domain, and introduced an α-algorithm. In the first step, the proposed approach detects anomalous process executions in the mined WF-nets for concrete cases. Then, the process conformance is checked by comparing process fragments with the identified WF-net. The α-algorithm discovers a net that models all acceptable behaviour of a process, using a given complete event log. A token game is then played to verify the conformance of the identified WF-net. In the token game, anomalous audit trails do not correspond to the possible firing sequences of identified WF-nets. Moreover, the token game also detects the point at which the audit trails diverges from the normal behaviour that allows a real-time verification of the audit trails.

Doganata and Curbera (2009) discuss a semi-automatic auditing method for unmanaged processes. The method is based on the business provenance that sequentially records the collection of events for unmanaged processes. Similarly, van der Aalst et al. (2010) introduced an automated auditing tool *'Auditing 2.0'* to provide support for compliance auditors using process mining techniques. The auditing framework provides support for considering the running process instances, and compares them with models based on historic data or business rules. Arya et al. (2010) also use a similar approach to gain insights into the conformance of an operational process of a given process model. The authors implemented their approach in the PROM[15] Framework. The approach uses current event logs (collected in real time) that carrying information about the activities being performed, and the order in which they are performed. Later, they compared these simulated event logs against existing conformance technique based on Petri-nets.

2.4.2 Database Technology–based Formal Approaches

Agrawal et al. (2006) use database technology to assist compliance with the internal controls of SOX Act. The approach employs workflows and discovery–driven OLAP to verify compliance with internal controls and irregularities in the financial data respectively. Initially, the internal processes are first modelled as workflows containing the required control activities, and the log of each workflow is stored in

[15]Process Mining Framework available at: `http://www.processmining.org`

database tables. Policies are later enforced at run-time. This ensures that only routine transactions comply with the prescribed WFs, which serve as on-the-shelf compliant WFs. During the compliance auditing, these on-the-shelf workflows are reconstructed using correlation rules from the activity logs, and are compared with the required workflows to determine whether transactions are compliant with internal controls.

Johnson and Grandison (2007) use *Hippocratic Database:* HDB for compliance auditing of data protection laws. The approach uses an HDB active enforcement architecture that operates as a middle-ware layer on the top of the database to enforce fine–grained policies concerning the disclosure of information. In the first step, the policy creation (HDB control) center allows the creation of policies, and then negotiates the preferences based on an in/out mechanism. Once policies and preference negotiations are formally defined, all policies are stored in an HDB logging system. Upon receiving an automatic audit query, the HDB logging system performs a statistical analysis of the query logs and generates a list of suspicious transactions, which are then combined into a single audit query. To confirm compliance, the output audit query contains the user identity, time, purpose, recipient, and exact information about the policy and pertinent disclosure information.

2.5 Hybrid Approaches

Apart from the above–mentioned classification of reported approaches in compliance management, some hybrid approaches can also be found in the literature. These claim to provide a full spectrum of compliance support. Moreover, some methods, apart from the usual components of compliance checking and monitoring, incorporate new artefacts from a business strategy point of view. The rest of this section discusses some identified hybrid methods.

Ghanavati et al. (2007) introduce a framework for tracking legal compliance in the health care domain. The framework demonstrates compliance tracking by defining and maintaining the correlation between the health care information custodian's policy models and business process models using goal–oriented language (GRL) and uses case map (UCM) notations. The custodian policy models consists of a *source links* and *responsibility links*. Source links are relationship links between the legislative policy definitions and hospital UCM model elements. The responsibility links, on the other hand, establish relationships between the UCM elements and GRL elements. These links are later checked for potential differences to see whether compliance requirements have been met. Any difference between what is implemented in the business process model and what is required by the privacy legislation (policy custodian model) is reported as rules violations.

Sapkota et al. (2011) discusses semantic methodologies for automated regulatory compliance support, using semantic web technologies. The proposed framework, RegCMatic, addresses the problem to automatically extract and model regulatory information; and to generate links between the internal compliance tasks and applicable regulations. Using various document formats such as PDF and HTML, the authors first extracted the regulations so that they could be converted into a machine-readable format. The list of extracted regulatory obligations was then processed using GATE (Cunningham et al., 2001), a text engineering platform, and the executable semantic rules were generated from the regulatory ontology. The authors implemented their proposed work using an industry case study that used Eudralex EU regulations[16]. Despite the nature of the regulatory requirements used in Sapkota et al. (2011), the work seems to offer a method for addressing the compliance problem where the business rules are frequently changed. However, the extraction of regulatory information is not fully automated due to the document format used. In practice, regulatory bodies use different document formats, and extracting information from a variety of document formats is a challenging task and requires human intervention to adjust the document format as required.

Rifaut and Dubois (2008) use goal–oriented techniques to present their compliance assessment framework for quality improvement based on ISO/IEC 15504 standard. However, the framework is in its evolutionary stage; the authors report future work in methodology, and in tool support for the management of compliance requirements and their traceability to the Process Assess Model (PAM) for assurance purposes. Kähmer et al. (2008) introduce a formal technique to elicit the regulatory requirements. The proposed technique represent the context of the policy rules, with case frames to semantically verify the regulations against the requirements. The technique uses words matched with a dictionary of policy regulations to detect the regulation sentences relevant to the requirements, such as structural similarity. Any dis–resemblance in the words format is detected, and

[16]Eudralex `http://ec.europa.eu/health/documents/eudralex/index_en.htm` retrieved 25th October 2012

notified to the analyst as a violation.

The consistency of the regulatory rules is one of the issues (as reported in Awad, 2010) that cause frustration for the analysts. Inconsistency in the rules can lead to their misinterpretation, and the incorrect modelling of the regulations. Jiang et al. (2014, 2013) proposes a consistency and compliance checking framework (CCCF), using the Norms Nets (NN) and Coloured Petri Nets (CPNs). The NN are used to formalise the regulatory rules and their relationship, whereas the CPN semantics implement the compliance-checker toolbox. The CCCF framework provides information on whether a set of regulations is consistent, and whether the business processes comply with the imposed regulations. Although the Jiang et al.'s framework is able to provide a reasonable degree of automated support for verifying the compliance to regulation, the transformation of the legal rules into NNs is primarily manually interpreted. In addition, from a business process perspective, the transformation of the model event sequences that model the behaviour of the agent (that is, trace generation) is also manual; this renders the proposed framework less effective. In contrast, the compliance checker proposed in Governatori and Shek (2013) performs these tasks automatically. Another downside of this framework is that there is no mechanism for modelling the temporal constraints in CPNs; thus, the compliance to regulation with temporal modality cannot be verified.

2.6 Existing Evaluation Approaches for CMFs

In the previous sections, we discussed different CMFs, methods, and approaches that provide compliance management support for legal requirements using different techniques. Given the extensibility of the business process compliance domain, and the diversity of these CMFs, evaluating different capabilities of the CMFs is a difficult task. In this section, we discuss some existing surveys and evaluations (reported in the literature) that examine different features of existing CMFs; in particular the evaluations from the legal requirements perspective.

Turki and Bjekovic-Obradovic (2010) evaluate the practices of legal rule analysis for extracting the key information for information system engineering (ISE) and goal–oriented–based approaches, with the aim of achieving, and maintaining the regulatory compliance. The authors use a three-point criterion, namely: extraction of rights and obligations, modelling regulations, and traceability support for compliance to conduct the evaluations. The scope of this evaluation is limited to the modelling of the regulations to goal-oriented approaches only, and to their traceability of the legal requirement support for compliance. Also, the authors evaluated approaches specific to the design of compliance e–government services only, and do not evaluate other approaches such as design–time, run–time and auditing–based techniques. Another downside of their evaluations is that the authors do not evaluate *how the extracted legal requirements are modelled, and how the traceability of the compliance support is achieved.*

Otto and Anton (2007) survey the legal requirements for a large number of approaches to modelling, and using the legal texts (regulations) for systems development. They identify the strengths and weaknesses of each of the surveyed approaches, based on the policy and regulations. They extract a large set of legal requirements for the tool support for requirements engineers and compliance auditors in order to address the challenges related to the legal compliance in software systems. The survey is limited in scope as its authors study *the ways in which requirements engineers and compliance auditors from different engineering disciplines use the legal texts to devise the regulatory compliance software systems.* In addition, the authors examine the ways in which the regulatory texts can be used to specify the system requirements, and *the ways in which analysts use the legal texts to devise the policies for the software system requirements for compliance monitoring systems.* Essentially, the survey does not include the way in which the *"legal requirements"* can be properly represented to check their compliance; or nor does it show how to systematically evaluate whether a specific CMF can provide the reasoning support for all types of the normative requirements.

Elgammal et al. (2011a) compare the expressive power of three formal languages for the specification of compliance requirements, with a focus on the design-time verification of business processes. The comparative analysis is based on the comparison of the capabilities and limitations of the evaluated languages from temporal and deontic families of logic with eleven selected features (such as formality, expressive power, declarativeness, non-monotonocity and real–time support) that compliance request language should support for the specification of legal requirements. The main shortcoming of the Elgammal et al.'s comparison is its limited scope of temporal logic and deontic logic as the verification of business processes in this regard requires that the chosen modelling language is expressive enough to capture the nuances of all types of legal requirements. We argue that this is only possible when the legal requirements are properly modelled on business processes. Another downside of their comparison is that they only compare two families of logics and other formal logics (such as Event-Calculus and First-order-logic) are excluded.

El Kharbili (2012) provides a detailed comparative analysis of the functional and non-functional capabilities of Regulatory Compliance Management (RCM) solutions in the Business Process Management (BPM) domain, based on a predefined evaluation criteria. In the first category, the authors evaluate the RCM solutions from the business users; methodological and RCM architecture perspective; in the second category, nine functional areas of the RCM from a BPM perspective (such as the strategy model and business process model), and compliance dimensions (such as compliance enforce, audit, and verification) are evaluated. In the last category, on the other hand, the authors use the functional and non-functional capabilities of a CMF as the evaluation criteria. From the compliance dimensions, the authors extracted three distinct types of rules—that is, *structural, temporal* and *contractual* rules—that are supported by the modelling languages. However, they do not systematically evaluate the *"legal requirements"* from a formal reasoning perspective, to provide a proper representation of the legal norms for the compliance checking of business processes.

More recently Ly et al. (2013) present an analysis framework to compare and evaluate the compliance monitoring approaches, using a set of core functionalities from a business process and legal requirement perspective. The proposed framework is based on the ten core compliance functionalities for monitoring the capabilities of a CMF. These core functionalities are relevant to various aspects of business processes (such as time, resources, activity lifecycle and data) and legal requirements (such as compliance rules violation detection, explanation, and degree of compliance support). The authors collected these features from the compliance management literature, and from the study of five state-of-the-art compliance monitoring approaches.

From the business process perspective, Ly et al. (2013) include all four aspects of a business process—control–flow, data, time and resources—as the core functionalities in their evaluation framework. The compliance rules relevant to *control–flow* can specify the order in which activities are to be performed. A compliance rule might be concerned that some activities of the process are executed in a predefined order. For example, the anti-money laundering act that requires if a large amount of money is transferred or deposited from/into an account, the bank are obliged to report such a transaction. The compliance rules with control–based conditions can be implemented and checked at design–time. Awad (2010) argues that given a correctly designed process model, it is very unlikely that a control flow-based compliance rule can be violated. Accordingly, the execution of business processes tasks might also involve managing a large amount of *data*. For example, information stored in the databases might change, new data might be produced and tasks might need specific data to complete. This information can flow along process in the form of data objects, for example, in the form of documents (such as a form/rule document). Compliance rules might include the constraints on the data management so that data objects must also be represented in the models, and can be subject to compliance checking.

On the same note, *time* is another important aspect of the process, some compliance constraints might include constraints that a particular task in a process is completed within the *t* unit of time (Ramezani et al., 2013). Weigand et al. (2011) categorise the temporal rules into qualitative and temporal constraints. Where the qualitative constraints aim to determine how temporal entities are related to each other, quantitative time specifies the difference in time between the entities. Generally, temporal constraints come from contracts such as a service level agreement (SLA) which might impose the requirement to keep a record of customers' products for several years. The temporal constraints are usually monitored for compliance at run–time. Similarly, tasks within an organisation are performed either by machines or by human *resources* called *agents*. In some situations, compliance rules (or internal policies) might be concerned with the agent specifying *"who will execute the task"*. For example, a rule statement might impose the condition that the task to be performed by two or more persons (for example, a segregation of duty [SoD] condition) to ensure that no unauthorised person executes sensitive transactions. Hence, it is particularly important that a CMF is able to model the constraints on the human agent at the time of process modelling. However, it is largely argued that correct modelling of the resources constraints is not sufficient because of the human factor in situations where human monitoring is required for compliance checking at run–time (Wolter et al., 2009).

From the legal requirements perspective, the downsides of the framework presented in Ly et al. (2013) is that it includes only a limited set of functionalities, and that the authors emphasise the *proactive violations detection, explanation*, and *level of compliance support* features of a CMF only. Compliance is about the legal rules, and one of the fundamental requirements for automated compliance checking is that a CMF is able to *formally represent the legal rules*. This is because rules are generally written in natural language, and generally incorporate legal jargon. Also, different people understand and interpret compliance rules differently, and this can lead to *inconsistencies and redundancies*. The inconsistencies of the compliance rules can be in the form of redundant data, or conflict between the two (or more) rules, or both. The redundancy is attributed to the appearance of the same rules or data several times, and describing the same situation. Hence, inconsistencies in the compliance rules might severely hamper the correct modelling of legal rules, and ultimately results in incorrect compliance results.

On the same note, compliance rules are modelled using logic–based formal languages, which, by the virtue of their formal semantics, are complex. This can limit the readability of the rules to technical people only. Hence, the formal specification and handling of the rule inconsistencies largely depends on the expressive power of the chosen language. Researchers argue for a careful selection of the formal language for the representation of legal rules (Governatori and Sadiq, 2009), and list several characteristics of a formal language for the representation of legal rules; for example, reasoning support, declarativeness, ability to handle inconsistencies, and readability (Elgammal et al., 2011a). The features of formal representation, and the handling of inconsistencies in the legal rules for which a CMF must be able to provide support, are not considered.

Accordingly, another important functionality that Ly et al. (2013) have overlooked is *coupling the legal requirements with business processes*. One of the desirable features of a CMF is that it is able to provide an automated support to decide which rules are applicable to various tasks of the process, and link them (Awad, 2010). Moreover, the legal requirements are frequently changed, updated, or removed because of the fast changing environments in which organisations operate. Coupling the legal requirements with the processes, allows the process designers to implement changes in the processes whenever the business rules are changed. Also, in coupling the legal rules with the business process, the CMF must be *extensible* and must not suffer as the results of the size and number of the rules it can accommodate. The feature of extensibility is not considered in Ly et al.'s analysis framework. In addition, they considered compliance monitoring frameworks only, while design-time and post-execution time frameworks are excluded.

2.7 Summary

In this chapter, we have presented a detailed review of the literature related to business process compliance (BPC). In particular, we began by analysing the existing CMFs from an organisational legal requirement compliance perspective. From this perspective, we have investigated the ways in which organisations address the compliance of the regulatory requirements and their internal controls, and have analysed policy–based, internal control–based, organisational compliance requirement frameworks. We then introduced different compliance management strategies such as design–time, run–time, and post–execution time compliance management. For each category, we analysed several of CMFs and approaches, with the focus on what they can do and what they cannot do in terms of providing reasoning and compliance management support for all types of legal requirements. Since proper modelling of the legal requirements is paramount from a business process compliance perspective, we also analysed different features for each of these CMFs; for example, features such as which formal language they use for modelling legal requirements, whether they can handle complex rules, and how they link the compliance requirements with the business processes for compliance checking.

Finally, we studied various works from the business compliance domain, surveying and evaluating different features of existing CMFs. We discovered that most studies surveyed had evaluated a very limited set of features of existing CMFs. Furthermore, none had systematically evaluated whether an existing CMF provides the reasoning support for all types of legal requirements, or whether legal requirements are properly modelled for business process compliance checking. We aim to address this key shortcoming in this current study by proposing a formal framework that systematically evaluates the features of existing CMFs, in particular, from the perspective of a proper modelling of legal requirements. To this end, as a first step (in the next chapter) we introduce a classification model of normative requirements, and provide semantics definitions of each class of the classification as the key contribution of the study. The classes of the proposed classification model, and proposed semantics provide the basis for devising the evaluation framework, and are used throughout the thesis.

Part II

Modelling Process Compliance

Chapter 3

Normative Requirements

3.1 Background

In the context of law, *norms are generally legal binding rules of conduct issued by a competent (often state) legal authority*[1] *under certain circumstances and by using certain procedures*. Essentially, depending on the nature of the applicability conditions or circumstances under which they are applicable, norms can have several features. For example, Artificial Intelligence (AI) and Law and Legal Reasoning largely admit that norms have the generic feature of conditional structure of the form (Kelsen, 1991; Sartor, 2005):

$$\text{IF } \{A_1, A_2, \ldots, A_n\} \text{ THEN } \{\text{B}\} \tag{1}$$

where $(A_1, A_2, \ldots, A_n)$ are the application conditions prescribed by the norm, and B represents the desired effects of following the conditions of the norm[2]. The conditional structure (1) shows an immediate connection between the norms and

[1]In a social context, as defined in a law dictionary, a legal authority is a government or non–government organisation (NGO), or an individual invested with power to create legal norms, to assume legal obligation, to sue and be sued in their own right and to be held accountable.
`http://thelawdictionary.org/legal-entity/`

[2]Note that it is possible that norms might not have any associated conditions; that is, their effects do not rely on any preconditions for desired effects. For example, one universal norm is that *everyone has the right to live in freedom.* This is a very generic norm and intends to achieve effects without imposing any conditions; however, norms often come with conditions.

rules[3]. A rule can be understood as a set of explicit regulations governing conduct or procedures within a particular domain of activity (Abate and Jewell, 2001). For example—in social context, speed limit rules, tenancy occupancy rules or more general—in logic, rules of inference or implication. On the other hand, a norm[4] is a set of standard rules and laws laid down by the legal system (or an authority) against which the appropriateness or inappropriateness of entity's behaviour is judged.

Generally, rules specify *how to behave* and can be classified into: (a) *determinative (constitutive) rules,* which specify the activities that cannot exist without such rules; (b) *technical rules,* which state what should be done to achieve a particular outcome; and (c) *prescription rules,* which control action by specifying *what is obligatory, permitted, or what must not be performed* (Gordon et al., 2009; von Wright, 1963).

In the legal domain, prescription norms regulate the behaviour of their subject by specifying *what should be done, by whom, and under what circumstances.* The structure and properties of these norms has been subject to extensive research in AI, Law and Legal Reasoning with respect to various property aspects; for example, reification (Gordon, 1993), rules semantics, defeasibility (Gordon, 1993; Prakken and Sartor, 1996; Sartor, 2005), contraposition (Prakken and Sartor, 1996), rules validity (Governatori and Rotolo, 2008b), isomorphism (Bench-Capon and Gordon, 2009) and normative effects (Rubino et al., 2006) (see Gordon et al., 2009, for a detailed list of properties of norms).

From a business process compliance perspective, *norms* aim to control the behaviour of a business process by imposing constraints (that is, compliance rules) on *how activities should be carried out,* and applying penalties for any divergent behaviour. These constraints might be relevant to one or more aspects of a business process such as control-flow, data, or resources etc.

Generally, compliance rules (or normative requirements) that constraint the behaviour of business processes are written in natural language (c.f. those found in legal or policy documents). For automated compliance checking of business processes, normative requirements should be translated into a format that machines can understand. To this end, Sadiq and Governatori (2010) argue that business process compliance is the "alignment" of the formal specifications of a business process and the formal specifications of the relevant normative requirements. As

[3]Notice that in this thesis we use the terms norms and rules interchangeably in the regulatory sense unless stated otherwise

[4]As defined in Black's Law Dictionary, available at: `http://thelawdictionary.org/norm/`

discussed above, normative requirements might have different structures and properties depending upon the conditions of applicability, and the circumstances under which they are applicable. Thus, from the perspective of the formal specifications of normative requirements, the question is: *which properties of normative requirements are relevant* for business process compliance checking, and *can they be further classified according to the relevant properties*.

To address this question, this chapter presents, a classification model of normative requirements based on the temporal, normative, and persistent effects of norms for business process compliance checking. The rationale behind the use of these properties to classify the normative requirements is that usually norms have a particular lifespan. In other words, a norm is only applicable for a certain period, and does not hold indefinitely. Accordingly, a norm ceases to hold once its objective has been achieved, or other conditions begin to apply.

This chapter is structured as follows: Section 3.2 discusses norms in the context of business process compliance from the temporal aspect of the validity of the norms. Then, as a major contribution of this study, a classification model of normative requirements, based on the temporal validity of obligations, and the effects of violations on obligation, is presented in Section 3.3. This is followed (in Section 3.4) by the formal semantics based on the concept when an obligation enters into force, until when it remains into force, the effects of the violation on an obligation. Section 3.5 positions and discusses this study's proposed classification in the context of the related work on exisitng classifications. Finally, concluding remarks in Section 3.6 highlight the contributions of the chapter.

3.2 Norms, Time, and Compliance

Time plays an integral role in norms, in legal reasoning, and in areas governed by norms. For example, many of the normative requirements in the area of business process compliance concern the temporal aspects of norms. Suppose you have a contract specifying that one party has thirty days to pay for an invoice, and that goods cannot be delivered without payment. Thus, you have an obligation to pay after receiving an invoice; this in turn, requires that the payment *must be made before* the time of delivery.

Receiving the invoice triggers (enforces) the obligation to make a payment to complete the transaction. Accordingly, we have conditions that must be fulfilled in a determined time interval or by a given deadline, and other conditions that must be met before or after specific events. Moreover, some obligations might include conditions that *must persist over an interval of time*; for example, the continuous monitoring of a patient's blood pressure and ECG during a surgical operation. Regardless of the type, validity and nature of the legal effect(s) of an obligation, the temporal aspect of an obligation revolves around its following generic aspects (Palmirani et al., 2011):

1. *the time when an obligation is in force*,
2. *the time when an obligation is fulfilled*, and
3. *the time of application*.

Accordingly, when a business process is subject to norms, it is particularly important that the process complies with the obligations imposed by the norms for the whole duration of its validity; that it meets the deadlines, and that it follows the constraints for maintaining and delaying actions. Figure 3.1 illustrates the generic temporal validity aspects of obligations.

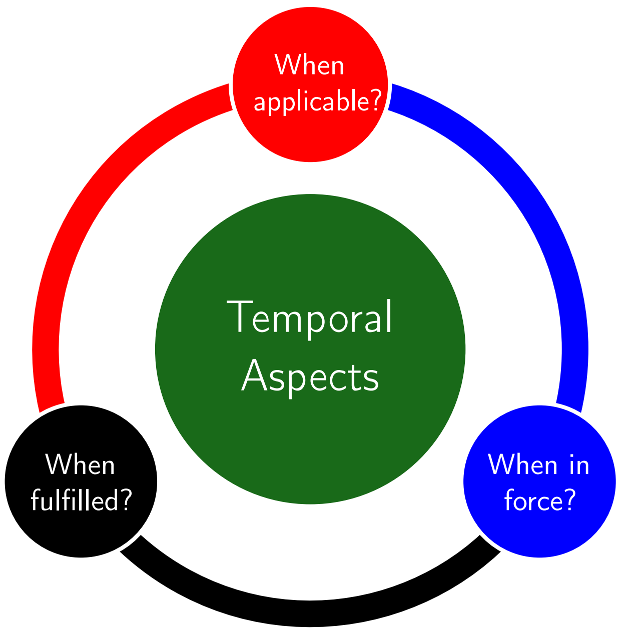


Figure 3.1: Temporal Model

Capturing the real meanings of norms is paramount for the modelling of, and reasoning about compliance checking of business processes, and, in general, for legal reasoning. It is also important that the chosen language supports the highest degree of abstraction in order to model the real meaning of the norms and the obligations they define (Awad, 2010). In other words, *the chosen language should be able to model states of affairs, and actions, as well as the (temporal) relationships among activities.*

Many studies have been conducted to model obligations, and various classifications of obligations where *time* is the key concept, have been identified in these studies, in particular, in the context of business process compliance; for example Hilty et al. (2005) and Governatori et al. (2007b, 2005), to name but a few (see Section 3.5 for details). Most of the existing classifications study norms from various perspectives of business processes, and can be used for specific purposes only. Furthermore, the existing classifications do not encompass various types of obligations based on the time, the effects of an obligation on other obligations and obligations arising from the violations. In the next section, we discuss a classification of obligations along temporal dimensions (Hashmi et al., 2013). The key aspects of this classification are:

- *temporal validity aspects of obligations and persistence effects of norms,*
- *what constitutes the violation in terms of the temporal validity of a norm,* and
- *whether violated norms can be compensated for.*

In the classification, along the temporal dimensions, for each type of obligations we specify when an obligation enters into force and the time until it remains in force, or whether it is violated at a particular point in time. Unlike other classifications, our proposed classification encompasses the (above) generic temporal model related to the validity and persistence effects of obligations after violations.

3.3 Classification of Normative Requirements

As mentioned earlier the scope of norms is to regulate the behaviour of their subjects, and to define what is legal and what is not. Typically, norms describe the conditions under which they are applicable and the normative effects they produce when applied. Gordon et al. (2009) provide a comprehensive list of normative effects.

From a compliance perspective, the normative effects of importance are the *deontic effects*. The basic deontic effects are: *obligation*, *prohibition* and *permission*.[5]

Let us start by considering the basic definitions of such concepts:[6]

Obligation: A situation, an act, or a course of action(s) to which a bearer is legally bound, and if it is not achieved or performed results in a violation.

Prohibition: A situation, an act, or a course of action(s) which a bearer should avoid, and if it is achieved results in a violation.

Permission: Something is permitted if the obligation or the prohibition to the contrary does not hold.

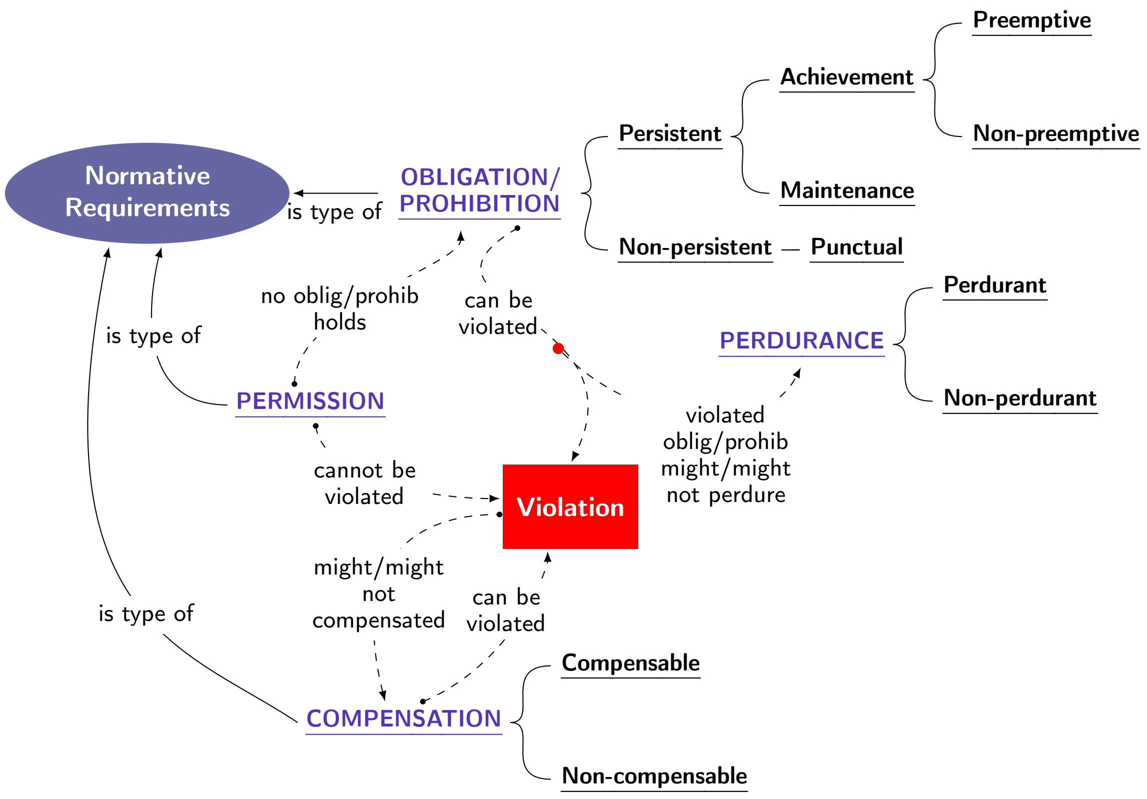


Figure 3.2: Normative Requirements: Classes and Relationships

Figure 3.2 illustrates the classification of the three basic deontic effects and the relationship between such effects and the notions of compensation and violation. The classification provided here has been obtained through a systematic and

[5]There are other deontic effects, but these can be derived from the basic ones, see (Sartor, 2005).

[6]The definitions of above concepts considered here are given by the OASIS LegalRuleML working group. The OASIS LegalRuleML glossary is available at `http://www.oasis-open.org/apps/org/workgroup/legalruleml/download.php/48435/Glossary.doc`.

exhaustive way where one considers the aspects of the validity of obligations (or prohibitions), and the effects of violations on them; namely: *whether a violation can be compensated for*, and *whether an obligation persists after being violated*.

Obligations and prohibitions are constraints that limit the behaviour of processes. The difference between obligations and prohibitions and other types of constraints is that they can be violated. On the other hand, *permissions* are constraints that cannot be violated and thus, they do not play a direct role in compliance. Rather, they can be used to determine that whether there are any obligations or prohibitions to the contrary, or to derive other deontic effects (see Makinson and van der Torre, 2003, for a detailed discussion on permissions).

Legal reasoning and legal theory typically assume a strong relationship between obligations and prohibitions: the prohibition of A is the obligation of $\neg A$ (the opposite of A), then, if A is obligatory, then $\neg A$ is forbidden (Sartor, 2005). In this chapter, we will subscribe to this position, given that our focus here is not on how to determine what is prescribed by a set of norms and how to derive it.; accordingly, we can restrict our analysis to the notion of an *obligation*.

Compliance means to identify whether a process violates a set of obligations or not. Thus, the first step is to determine *whether and when* an obligation is in force. Hence, an important aspect of the study of obligations is to understand the lifespan of an obligation and its implications for the activities undertaken in a process. As we have alluded to above, norms provide the conditions for the applicability of obligations. The next question is, then: *How long does an obligation hold for?* A norm can specify that an obligation is in force at a particular point in time only or, more often, a norm indicates when an obligation enters into force. An obligation is considered to remain in force until it is terminated or removed. Accordingly, in the first case, we will speak of *non–persistent obligations*, and *persistent obligations* in the second.

If a *persistent obligation* needs to be obeyed for all time instances within the interval in which it is in force, it is categorised as a *maintenance obligation*. If achieving the contents of the obligation at least once is enough to fulfil for that obligation, then it is considered an *achievement obligation*. Another aspect of an *achievement obligation* to consider is whether the obligation could be fulfilled even before the obligation is actually in force. If this is allowed, then we have a *preemptive obligation*; if not the obligation is a *non–preemptive obligation*. In contrast, a *non–persistent obligation* needs to be obeyed for the instance it is in force, and is categorised as a *punctual obligation*. If the contents of *punctual obligations* are not immediately achieved, a violation is triggered.

An obligation of any type can be violated. A *violation* does not always imply the consequent termination of (or the inability to continue) a business process. Certain violations can be compensated for, and processes with compensated violations are still compliant (Governatori and Milosevic, 2005; Governatori and Sadiq, 2009). For example, contracts typically contain compensatory clauses that specify penalties and other sanctions triggered by breaches of contract clauses (Governatori, 2005). However, not all violations are compensated for, and uncompensated violations mean that a process is not compliant. The effects of a violation on the obligation that has been violated also need to be considered. If the obligation persists after being violated, it is considered a *perdurant obligation*; if not, it is a *non–perdurant obligation*.

Accordingly, the violation of an obligation would not necessarily mean the termination of interaction between the tasks of a process. A violated obligation might be further *compensable*; that is, after the violation of an obligation, a new obligation might take effects and amend the violation of the violated obligation. The obligation taking effects after the violation creates an intermediate layer of protection between the violation and the penalty (Wyner, 2008), thus aiming to achieve a sub-ideal situation where a business process is still compliant even if some of the obligations have been violated (Governatori et al., 2008b).

Note that the compensation of a violated obligation depends on the obligation's violation conditions; that is, conditions stating whether it can be compensated for. If a violated obligation is compensated for—we speak of a *compensable obligation* otherwise the violation obligation is a *non–compensable obligation*.

In this section, the basic intuition of the various classes of the above classification model has been discussed. These notions give us a sense of the different kinds of obligations that might appear in the norms and, depending on their implementation, of what effects they might produce. In the next section, the formal semantics—that provide the formal definitions that are required to model these concepts based on the *temporal validity of the norms over a timeline*—are presented.

3.4 Formal Semantics

In this section, the formal definitions of the above–discussed concepts and concrete examples from the regulatory frameworks are discussed. These definitions are independent of any formalism, and are based on the concept that when an obligation enters into force and until when it remains in force and the effects of the violation on the obligations. In presenting these formal definitions, all that needed is the notion of *time-line*—that is, a possibly totally ordered discrete set of time points. In addition to that, the time-line has a minimum. Notice that, an infinite time-line is isomorphic to the set of natural numbers. We can restrict our analysis to a finite set of natural numbers in the case of finite time-line. In what follows, the existence of a suitable logical language $\mathfrak{L}$ (which can be a set of atomic propositions) on which the logic formulas are written to model the obligations and representation of the environment.

Definition 1 (State)**.** *Given a time-line, we define a function State:* $\mathbb{N} \mapsto 2^{\mathfrak{L}}$

The meaning of the function *State* is to identify what formulas that are evaluated as true at the n-th time instant of a time-line.

Definition 2 (Obligation in Force)**.** *Given a time-line, we define a function Force*: $\mathbb{N} \mapsto 2^{\mathfrak{L}}$.

The meaning of the function *Force* is to identify the obligations in force at the n-th instant of time in a given time-line.

Remark 1. *In formally defining these obligations, we are not interested in the mechanisms that establish which obligations are in force, and when. This is within the scope of specific compliance applications and implementations as we will show in the following chapters.*

Definition 3 (Punctual Obligation)**.** *Given a timeline, an obligation o is a punctual obligation if and only if:*

$$\exists n \in \mathbb{N} : o \notin Force(n-1), o \notin Force(n+1), o \in Force(n)$$

A punctual obligation is violated if and only if $o \notin State(n)$.

Figure 3.3 illustrates the nature of a punctual obligation. The conditions of a *punctual obligation* must be fulfilled immediately; if not, we have a violation. That

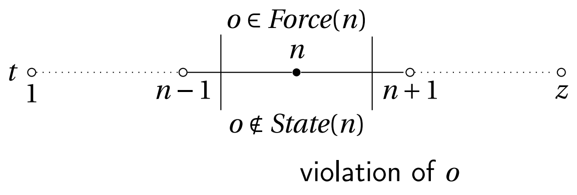


Figure 3.3: Punctual Obligation

is, o is violated at time n if o is not true at n (or at the n-th instant of time in the time-line).

Definition 4 (Persistent Obligation)**.** *Given a timeline, an obligation o is a persistent obligation in t if and only if*

$\exists n, m \in \mathbb{N} : n < m, o \notin Force(n-1), o \notin Force(m+1), \forall k : n \leq k \leq m, o \in Force(k)$

The obligation o is in force between n and m.

A persistent obligation is an obligation in force in an interval of time. The Figure 3.4 illustrates the definition when a persistent obligation o is in force between n and m. *Persistent obligations* can be further classified as:

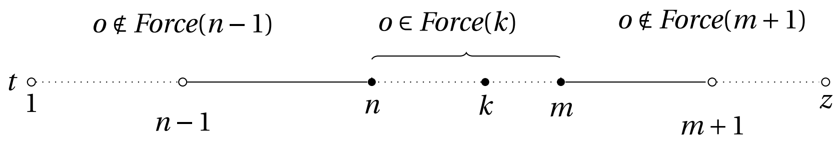


Figure 3.4: Persistent Obligation

(a) *achievement* obligations, and

(b) *maintenance* obligations.

The violation conditions for a persistent obligation can be derived from the violation conditions of these subclasses.

Definition 5 (Achievement Obligation)**.** *Given a timeline, an obligation o is an achievement obligation if and only if* $\exists n, m \in \mathbb{N}, n < m$ *such that o is a persistent obligation in force between n and m.*

An achievement obligation o in force between n and m is violated if and only if:

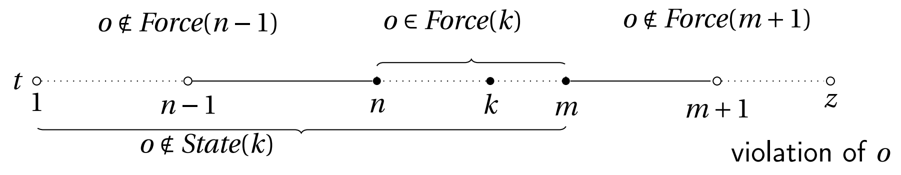


Figure 3.5: Preemptive Obligation

- *o is preemptive and* $\forall k : k \leq m, o \notin State(k)$;
- *o is non–preemptive and* $\forall k : n \leq k \leq m, o \notin State(k)$.

An *achievement obligation* is in force in an interval in the time-line, and can be further classified as: *preemptive* and *non–preemptive*. A *preemptive achievement obligation o* is an obligation that can be fulfilled even before the obligation is actually comes into force. In contrast, a *non–preemptive achievement obligation* can be discharged only after it comes into force. The violation of an achievement obligation depends on whether we have a preemptive or non–preemptive obligation.
For a *preemptive obligation o*, we have a violation if no state before the last state in which o is in force, the obligation o is in force (see, Figure 3.5 for details).

For the violation of a *non–preemptive* obligation o, in the set of states one has to consider for determining whether the obligation has been violated, only those defined by the interval in which the obligation o is in force need to be considered (see, the pictorial representation of the non–preemptive case in Figure 3.6).

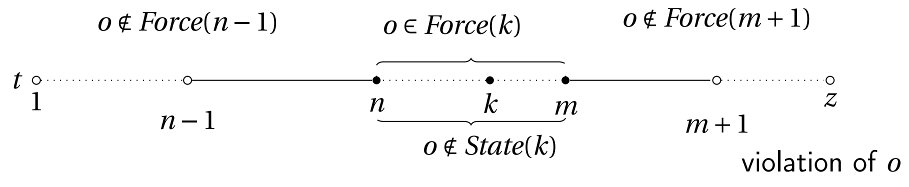


Figure 3.6: Non–Preemptive Obligation

Example 1. *Australian Telecommunications Consumers Protection Code 2012 (TCPC 2012); Article 8.2.1.*
A Supplier must take the following actions to enable this outcome:

(a) ***Demonstrate fairness, courtesy, objectivity and efficiency:*** *Suppliers must demonstrate, fairness and courtesy, objectivity, and efficiency by:*

  (i) Acknowledging a Complaint:

    A. immediately where the Complaint is made in person or by telephone;

    B. within 2 Working Days of receipt where the Complaint is made by email;....

The obligation to acknowledge a complaint made in person or by phone (8.2.1.a.i.A) is a *punctual obligation*, since it has to be done '*immediately*' while receiving it (thus, it can be one of the activities done in the task 'receive complaint'). On the other hand, 8.2.1.a.i.B is an *achievement obligation* since the clause provides a deadline for fulfilling the obligation. It is also a non–preemptive obligation; that is, it is not possible to acknowledge a complaint before receiving it.

Example 2. *Anti-Money Laundering and Counter-Terrorism Financing Act 2006; Clause 54 (Timing of reports about physical currency movements).*

(1) A report under Section 53 must be given:

  (a) if the movement of the physical currency is to be effected by a person bringing the physical currency into Australia with the person—at the time worked out under subsection (2); or

[...]

  (d) in any other case—at any time before the movement of the physical currency takes place.

Example 3. *Australian National Consumer Credit Protection Act 2009; Schedule 1, Part 2, Section 20: Copy of contract for debtor.*

(1) If a contract document is to be signed by the debtor and returned to the credit provider, the credit provider must give the debtor a copy to keep.

(2) A credit provider must, no later than 14 days after a credit contract is made, give a copy of the contract, in the form in which it was made, to the debtor.

(3) Subsection (2) does not apply if the credit provider has previously given the debtor a copy of the contract document to keep.

Clauses (d) and (3) of Examples 2 and 3 respectively illustrate a *preemptive obligation.* For clause (d), this obligation is in force when a financial transaction occurs, and the clause explicitly requires the report to be submitted to the relevant authority *before* the transaction actually occurs (it might be the case that the transaction never occurred). Clause (3), on the other hand, prescribes preemptive obligation in the sense that it requires a copy of the contract document be given to the debtor; however, the obligation is not applicable if the creditor has earlier provided a copy of the contract document (under clause [2] of the section).

Definition 6 (Maintenance Obligation)**.** *Given a time-line, an obligation o is a maintenance obligation if and only if* $\exists n, m \in \mathbb{N}, n < m$ *such that o is a persistent obligation in force between n and m.*

A maintenance obligation o in force between n and m is violated if and only if

$$\exists k : n \leq k \leq m, o \notin State(k).$$

Unlike an achievement obligation, a *maintenance obligation* must be complied with for all the instances of the interval; if it is not, we have a violation. Also, no deadline is required for a maintenance obligation, insofar as we do not need it to detect a violation. The deadline signals the instant that the obligation is no longer in force. Furthermore, it is possible to define maintenance obligation without a deadline; in other words, after it comes into force, an obligation remains in force indefinitely; in this case, one has to drop the reference to instance *m* in the above definition. The pictorial representation in Figure 3.7 illustrates the notion of a maintenance obligation.

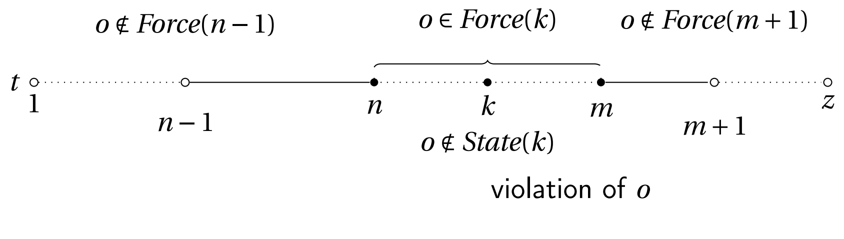


Figure 3.7: Maintenance Obligation

Example 4. *TCPC 2012. Article 8.2.1.*
A Supplier must take the following actions to enable this outcome:

(v) not taking Credit Management action in relation to a specified disputed amount that is the subject of an unresolved Complaint in circumstances where the Supplier is aware that the Complaint has not been Resolved to the satisfaction of the Consumer and is being investigated by the Supplier, the TIO or a relevant recognised third party.

In this example, as it is often the case, a maintenance obligation implements a prohibition. Specifically, it describes the prohibition against initiating a particular type of activity until either a particular event takes place, or a state is reached. As in the above example, Telcos operators are prohibited from taking credit management actions until a resolution of the complaint is to the satisfaction of the customer. The state, where a credit management action does not occur, must be maintained for all situations described by the norms until a resolution occurs.

The next three definitions are meant to capture the notion of compensation for a violation (see Figure 3.8). The idea is that a compensation is a set of penalties or sanctions imposed on the violator, and fulfilling them makes amend for the violation. The first step is to define what a 'compenstation' is. *A compensation is a set of obligations in force after a violation of an obligation* (Definitions 7 and 8).

Definition 7 (Compensation)**.** *A* compensation *is a function Comp*: $\mathfrak{L} \mapsto 2^{\mathfrak{L}}$.

The intuition behind the function *Comp* is that it associates a set of formulas to each formula; that is, if a formula corresponds to an obligation, and the obligation is violated, then the violation is compensated (or excused) by the formulas associated with the obligation. This is formalised by the following definition.

Definition 8 (Compensable)**.** *Given a time-line, an obligation o is compensable if and only if* $Comp(o) \neq \emptyset$ *and* $\forall o' \in Comp(o), \exists n \in \mathbb{N} : o' \in Force(n)$.

Notice that we have following requirements for an obligation to be compensable:

(i) *there are ways to make amends i.e.,* $Comp \neq \emptyset$;
(ii) *the actions that compensate are recognised as such (they are obligations in force), or they are not forbidden*; and
(iii) in most general form, there are no temporal requirements on when the compensation happens[7].

[7]In the vast majority of cases, it is expected that the compensatory obligations are in force after the violation. However, the definition above does not exclude retroactive compensations.

Figure 3.8 depicts the notions of compensation and recursive compensation for the violation of a compensation obligation.

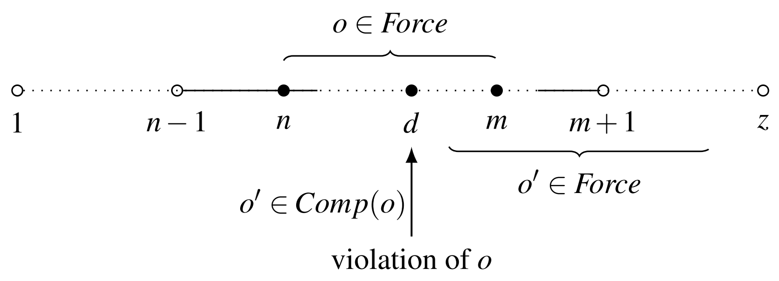


Figure 3.8: Compensation Obligation

Since the compensations are themselves obligations, they can also be violated, and compensable; thus, a recursive definition for the notion of compensated obligation is required.

Definition 9 (Compensated Obligation)**.** *Given a time-line, an obligation o is compensated if and only if it is violated and for every $o' \in Comp(o)$ either:*

1. *o' is not violated;*
2. *o' is compensated.*

In many cases, not all violations of the obligations are compensable, and the violation of an obligation might result in the direct imposition of penalties associated with that violation. Hence, for a stricter notion, if a compensated compensation does not amend the violation, it was meant to compensate, the recursive call can simply be removed. Thus, removing condition 2 from Definition 9[8] will capture the intuition of stricter non–compensable obligations.

In their most generic usage, compensation obligations can be used for the following two purposes:

- *to specify alternatives (that it, less ideal outcomes)* or
- *to capture sanctions or penalties.*

In any situation, an ideal outcome of the imposed obligations is highly desired; however, if the obligations conditions are not met because of violations, then

[8]Notice in defining the semantics for compensated obligations, we haven taken the most general definition without imposing any temporal requirements for the compensation; thus, the compensation could even precede the violation. Consider, for example, the natural language expression: "*I apologise in advance for ...*".

compensation allows for the achievement of a sub-ideal outcome that still renders the process compliance. In alternative cases, a compensation obligation may capture the penalties for violation.

Example 5. *TCPC 2012; Article 8.1.1.*
A Supplier must take the following actions to enable this outcome:

(a) ***Implement a process**: implement, operate and comply with a Complaint handling process that:*

  (vii) requires all complaints to be:

    A. Resolved in an objective, efficient and fair manner; and

    B. Escalated and managed under the Supplier's internal escalation process, if requested by the Consumer or a former Customer.

Example 6. *YAWL Deed of Assignment (Warranties & Indemnity); Clause 5.2*[9]

(5.1) Each Contributor warrants that:

  5.1.1 the Intellectual Property assigned to the Foundation by the Contributor under clause 2 comprises original works only, which have not been, and will not be, copied wholly or substantially from any other works,

  ...

  5.1.3 it has the right to assign and grant the rights under clause 2.

(5.2) Each Contributor indemnifies and will defend the Foundation against any claim, liability, loss, damages, cost and expense suffered or incurred by the Foundation as a result of any breach of the warranties given by the Contributor under clause 5.1.

Definition 10 (Perdurant)**.** *Given a time-line, an obligation o is a perdurant obligation with a deadline d if and only if o is in force between n and m, and $n < d < m$.*

A perdurant obligation o with a deadline d in force between n and m is violated if and only if

$$\forall j, j \leq d, o \notin State(j)$$

Figure 3.9 illustrates the notion of perdurant obligations. Notice that, all types of

[9]`http://www.yawlfoundation.org/files/YAWLDeedOfAssignmentTemplate.pdf`, retrieved on 28 March 2013.

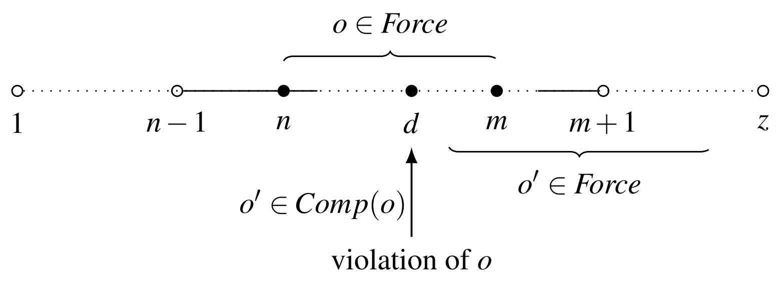


Figure 3.9: Perdurant Obligation

obligations have their own deadline (which is the end point of the interval where they are in force). The deadline can be used to indicate whether the obligation has been violated or not. Perdurant is a special case of obligation, which should have an explicit deadline which does not coincide with the end of the period when the obligation is in force.

Example 7. *Australian Telecommunications Consumers Protection Code 2012 (TCPC 2012); Article 8.2.1.*
A Supplier must take the following actions to enable this outcome:

(a) ***Demonstrate fairness, courtesy, objectivity and efficiency:*** *Suppliers must demonstrate, fairness and courtesy, objectivity, and efficiency by:*

  (i) Acknowledging a Complaint:

    A. Immediately where the Complaint is made in person or by telephone;

    B. Within 2 Working Days of receipt where the Complaint is made by email;....

Consider Example 7 (above). Clauses TCPC 8.2.1.a.i.A and 8.2.1.a.i.B state the deadlines for acknowledging a complaint; however, 8.2.1.a.i prescribes that complaints must be acknowledged. Thus, if a complaint is not acknowledged within the prescribed time then either clause A or B is violated; however, the supplier still has an obligation to acknowledge the complaint. Thus, the obligation in clause (i) is a perdurant obligation.

Remark 2. *Definition 10 only describes the perdurant obligation for pre-emptive achievement obligations. Simple adjustments can be made to model a similar notion for non–preemptive and maintenance obligations.*

3.5 Related Work

In this section, we discuss the work reported in the literature that proposes various classifications of norms, and compares this study's classification of normative requirements.

The structure and properties of norms have been extensively studied in the fields of Legal Reasoning, Artificial Intelligence, and Deontic Logic see (see Sartor, 2005), and have received comprehensive treatment from a formal and legal theoretical perspective. Accordingly, temporal reasoning has long been a topic of interest in Deontic Logic (Broersen, 2006; Frank et al., 2004; Governatori et al., 2007a), and other areas of legal reasoning, in particular, the notion to deadlines. Consequently, many classifications can be found in the literature.

Sartor (2005) classifies obligations from the legal viewpoint while, in Hilty et al. (2005), obligations are classified along the temporal structure and temporal distribution of the obligations. Unlike the classification presented in this chapter, the focus of Sartor's and Hilty et al.'s classification are the basic deontic concepts of *obligations, permissions,* and *normative conditionals* only. The former discusses the obligations and permissions, and further classifies obligations into behavioural, productive and directed obligations. The latter classifies the normative notions based on the temporal structure of obligations where the obligations can make statements about the properties of events that are observable; for example, achievement obligations. This is because as widely argued in the literature, it is apparent that *deadlines* are central to defining the various deontic notions (Wyner, 2008). The authors in (Governatori et al., 2007b, 2005) go a step further and incorporate the violations, while characterising the obligations based on the deadlines; however, no persistent effects of obligations—such as preemptive, non–preemptive, and perdurant types—have been considered in their characterisation of obligations.

Accordingly, in the context of the DECLARE (2010) framework, the authors classify obligation types as existence, choice, relation, and negative constraints. Accorsi et al. (2011) classify compliance rules from various regulatory frameworks for cloud-based compliant workflows. Spanning over nine categories, their classification comprises three main rules classes relevant to either the control-flow or the data-flow of workflow models. In contrast, a taxonomy of high-level patterns-based compliance constraints for business processes has been proposed in (Elgammal et al., 2010). This taxonomy of compliance patterns is classified into three distinct classes of constraints patterns namely: *atomic, composite,* and *timed* patterns. Weigand et al. (2011) on the other hand, provides a formal characterisation of the behavioural rules for business policy compliance for SOA (service-oriented architecture). Makinson and van der Torre (2003) study norms from a *permissions* in input/output logic perspective; they also provide a comprehensive classification of permissions classified as: *negative permissions* and *positive permissions.* Similarly, de Maat and Winkels (2007, 2010) classify the norms as *primary* and *secondary* rules from legal sources of law, based on the sentence structure prescribed in the legal documents. The sentences in the legal documents contain the semantics of legal terms, which stipulate various behavioural rules to constrain the agent's behaviour; for example, obligations, permissions and prohibitions. These classifications are mostly context-dependent and they are useful for a structural compliance checking of business processes only. The classification presented in this chapter, on the other hand, is generic and can be used in any context for business compliance checking.

3.6 Summary

This chapter addresses the question raised in the previous chapter: *What are the classes of normative requirements, and how can they be modelled.* Its resulting contribution is two-fold.

First, the chapter provides a classification of the normative requirements that are mandatory in the modelling of the normative component of the business process compliance. This classification comprises the deontic notions of obligations, permission, violations, and compensations etc. Each of these notions is further classified into sub-classes that cover a range of obligation types and the effects of each type over temporal dimensions is illustrated in Figure 3.2. Furthermore, along the temporal dimensions, the time an obligation enters into force, the time until when it remains in force or its violation at a particular point in time as specified. The presented classification has been developed in a systematic and exhaustive way using the well–known '*divide and conquer*' methodology and provides a rich ontology of the various obligation modalities along temporal validity and effects of obligations.

In addressing the *second* part of the question–*how can the classes of normative* *requirement be modelled?* The chapter makes its second contribution; that is, it provides the formal semantics for each class of normative requirements in terms of the temporal validity of an obligation, what constitutes a violation, and effects of the obligation violations. The formal semantics for each class are not restricted to any particular formalism, as they are generic in the sense that any formalism can be used to represent them, despite the fact that they are grounded with deontic logic in mind. To validate this fact, the next chapters illustrate how these semantics can be modelled with other formalisms, such as Event-Calculus (EC). In addition to the formal semantics, for each type of normative requirements, concrete examples from clauses of statutory/legislative acts that correspond to these requirements are provided. The presented classification extends the works discussed in the previous section in general, and the work of Governatori et al. (2005) in particular, where the deontic notions have been classified along the deadlines. Essentially, the proposed classification model, and the formal semantics that define the classes of the classification model, lay the foundations for this thesis.

In the next chapter, we discuss ways *how to formally represent and check the compliance* of the classes of the classifications model presented above against business processes, and present a formal foundational framework for modelling the legal component of business process compliance.

Chapter 4

Business Process Compliance

4.1 Background

Business processes provide a high-level view of how business operations can be performed to achieve a desired outcome. Hence, it is particularly important that they operate within the defined boundaries of the regulations (in the legal context) called *norms*. Aiming to control the behaviour of business processes, norms impose restrictions on *how activities should be carried out*, and impose penalties for any divergent behaviour.

Consider, for example, the procurement process of a government agency that handles the dynamic selection of contractors to place orders, which is implemented as a web service. Using such a web service, the agency can quickly place an order, and receive and evaluate the quotes from suppliers. This process is subject to certain regulations; thus, the procurement web service must be verified as compliant with the relevant regulations before it can be deployed. Hence, a process that reflects the behaviour of a web service can be used to verify the effectiveness of the regulatory and policy controls.

Governatori and Sadiq (2009) define *business process compliance* as the relationship between the formal specifications of a business process and the formal specifications of a set of normative constraints. A process is compliant if the specifications of the process do not violate the constraints that formalise the norms.

Accordingly, one has to provide:

- *a formal model for the representation of business processes*
- *a formal model for the representation of the norms and*
- *a bridging mechanism between the two representations (if they are expressed in two different formalisms)*

In the Introduction, we pointed out that a large number of CMFs exists address the issue of (regulatory) compliance in the context of business process management, service computing, and cloud computing domains (see Becker et al., 2012; Fellmann and Zasada, 2014), and offer a variety of compliance checking approaches. The general idea behind these approaches is to determine whether the constraints (that is, the norms) imposed by a regulatory framework (ranging from statutory acts to regulations, to industry standards, to best practices and internal policies) are met by IT systems.

Regardless of how good and feasible these approaches might be, to the best of our knowledge, the majority of the approaches fail to consider the aspect of whether the method they propose offers a faithful representation of the norms, and whether it is suitable to reason appropriately with the norms. A non–faithful representation of, and inappropriate reasoning with the norms, can have a significant impact on the effectiveness of an approach. By addressing the question (raised in the introduction) of *how to evaluate a compliance management framework*, this chapter presents a formal foundation framework to evaluate the abilities of a compliance framework to represent the norms with which a system need to comply.

In presenting this framework, the intention is not to provide yet another compliance checking framework, but a conceptually rich foundations for the norms for the legal component of the compliance problem. Thus, we provide a formal model for the representation of various notions of norms discussed in the previous chapter, and a formal model giving the specifications of business processes, and a mechanism integrating these two specifications. Essentially, these formal models provide formal semantics for the legal component of compliance in terms of the states that determine the temporal validity; what constitutes a violation; the effects of violations on other norms to which a business process might be subject to; possible ways in a which business can be executed; and how the behaviour of the processes can be validated against the norms.

This chapter is structured as follows: Section 4.2 of this chapter provides the formal foundations of business processes, the rationale behind using the workflow-nets (hereafter WF-nets) and enriching business processes with semantic annotations. Section 4.2.1 then provides the formal definitions, modelling various notions of norms. The formal definitions that underpin what is means to be compliant are given in Section 4.2.2. Section 4.3 provides and illustration of how the compliance checking of business processes can be carried out. A complaint-handling process (as a case study) is then discussed in Section 4.4. We formally demonstrate how our approach can be used to formally model and check the compliance of business processes against the relevant norms in Section 4.4.1. An evaluation of this approach is given in Section 4.5 is followed by a detailed discussion of related work in Section 4.6. Finally, Section 4.7 concludes this chapter with some remarks on contributions.

4.2 Formal Foundations of Business Processes

As previously discussed, business process compliance requires a formal model of the relevant business process and a formal model of the relevant norms. In this section, we provide formal definitions of processes annotated with compliance requirements, and the formal representation of the various notions of the norm classes discussed in the previous chapter. This provides both the model of the norms and the bridge between the formalisation of processes, and that of the norms. The aim is to show the evolution of the system or the environment in which a system operates, and to check that the resulting states (and intermediate states) are compatible with the norms.

In this section, we show how to start from the notion of business process model to describe the sequences of states corresponding to the execution of the process. Sequences of states are then used to provide the semantics of different classes of norms, and to provide the definitions of what it means to comply with a norm and to violate a norm. Compliance is related to the behaviour of a process; that is, whether it is possible to correctly execute a business process. Compliance is not only about the actions (that is, the tasks) undertaken during the execution of a process but also about their artefacts, and how actions change the environment in which a process is situated.

To capture this phenomenon, we adopt the idea proposed by Sadiq et al. (2007)

and enrich processes with control objectives by means of semantic annotations. Enriching processes with semantic annotations increases the understanding of the interaction between the business process specifications and the compliance controls specifications and compliance controls specifications for the involved stakeholders (that is, compliance officers and process owners).

On the one hand, embedding the compliance rules into business processes makes the compliance checking more transparent to the stakeholders; on the other, however, it also makes the handling of large compliance rules repositories a very difficult task. This is explained by the fact that compliance rules are frequently changed, removed, or updated. Some researchers (such as Ramezani et al., 2012b) propose to separate the compliance concern from business processes by capturing each aspect of a compliance requirement in a separate rule based on process vocabulary. For this, the use of a common business vocabulary–based upon the ordering-based, agent-based and values-based primitives to specify the compliance rules is proposed in (van der Aalst et al., 2011). These primitives cover a full spectrum of business processes aspects, and can be used to formulate the compliance rules. However, separating the compliance concern from business processes and the use of common process vocabulary raises the question: *How can the compliance requirements be enforced in the tasks of a business process* even if a common process vocabulary is used? Hence, it is difficult to trace the enforcement of the compliance requirements on business operations. Another question is *How feasible is it to use a common vocabulary to formulate the compliance rules* as business processes and compliance are two different concerns having different objectives and goals?

Our motivation for enriching the business processes with semantic annotations stems from the first issue as it allows for the explicit enforcement of the compliance rules on the relevant tasks of a business process, thus, making the traceability of compliance requirement enforcement rather easy. We are aware of the maintainability issue of the larger repositories and the manual work required to annotate the business processes; however, the maintainability issue is reserved for future work. Also, the reader is referred to (Hashmi et al., 2012), where a methodology to automatically annotate the business processes with the data extracted from the database schemas is proposed. Since the second issue is out of the scope of this thesis, we do not address it. Accordingly, we take an agnostic approach to the representation of annotations. All we need is that there is a language suitable for the representation of the annotations. These annotations are meant to capture the attributes, the resources and other information related to the tasks in a process (where the tasks themselves or the instances of the tasks can be captured in the language). In addition, we stipulate that the same language is used to represent both the annotations and the contents of the normative requirements. We also stipulate that the same is true for the representations of business processes.

As earlier mentioned, compliance is a relationship between the formal specifications of business processes and the formal specifications of the (relevant) normative requirements. Accordingly, we provide the formal background for the representation of business processes. A *business process* is self-contained, temporal, and logical order composed of events and activities that are executed to achieve a business goal. Generally a process model describes the order in which the activities should be performed (control-flow), by whom (agent/resources), and by using what (data). Minimally, a process might consist of a set of tasks representing (complex) business activities and connectors (for example, sequences and decisions points) that define the relationship among the tasks. Tasks and connectors collectively define the possible ways in which a process can be executed. Whereas a possible execution (called 'process trace'), on the other hand, is the way in which the tasks in the process model adheres to the order given by the connectors (see Dumas et al., 2013, for an extended representation of business processes).

In the BPM domain, a wide range of process modeling languages have been created, and new languages continue to emerge. Historically, process modeling has been mainly performed using general–purpose languages such as Activity Diagrams (AD), Unified Modelling Language (UML), Event-Driven Chains (EPC), Business Process Modelling Notations (BPMN), Yet Another Workflow Language (YAWL), Petri-Nets, Markov Chains, and Process Algebra (PA). These modelling languages, considering their expressivness in modeling business processes, can be classified as informal, semi–informal or formal (Lin, 2008; van der Aalst, 2009). The semi-formal languages (for example, AD, UML, EPC, and BPMN) have less rigid semantics and thus represent process models in a very user-friendly way. In contrast, the formal languages (for example Process Algebra, Markov Chains, and Petri Nets) describe processes more rigidly (using formal methods) and more accurately.

In this chapter, we use BPMN as the main modelling language for representing business processes; however, a fundamental problem with the BPMN is that it provides relatively precise semantics for modeling process models. The focus of this chapter is on business process compliance, which requires more information than provided by a pure BPMN process model. An automated semantics process analysis for business process compliance can be problematic. This is because of the BPMNs semantics, where the domain knowledge and structural elements are missing (Oro and Ruffolo, 2012). Thus, we opt for a formal process modelling language; in particular, we use Petri-Nets for modelling business processes, and then semantically annotate them with the effects of the norms. Another reason that we opt for a formal language is that we propose new classes of normative requirements; the way to represent these new classes of norms is a question that is addressed in this chapter. Notice, alternative formal languages such Process Algebra; Markov Chains can be used without any impact on the approach presented here.

In this chapter, we make use of *workflow-nets* (WF-nets) (van der Aalst, 2000), a subclass of Petri nets (Murata, 1989) to represent a business process. Hence, definitions 11–14 are necessary to formally define a WF-net and its behaviour. For other representations of a business process, one can directly begin with Definition 15 and the remaining definitions in this section can be easily modified for other representations of a business process.

Definition 11 (Petri net)**.** *A Petri net is a tuple $PN = (P, T, F, M_0)$*[1]*, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and $M_0 : P \to \mathbb{N}$ is an initial marking.*

A Petri net is a collection of two types of nodes: *places* and *transitions*. Arcs connect one type of node to the other. For a node $x \in (P \cup T)$, $\bullet x$ denotes the set of inputs to x, and $x\bullet$ denotes the set of outputs of x. The *state* of a Petri net is represented by a *marking* that describes the number of tokens in each place of a net.

A WF-net is defined as a subclass of Petri net with the following structural restrictions (van der Aalst, 1998, see): (i) *there is exactly one source place*; (ii) *exactly one end place*; (iii) *every node in the graph is on a direct path from the source place to the end place.*

Definition 12 (WF-net)**.** *Given a Petri net $N = (P, T, F, M)$, the net N is a WF-net if and only if:*

[1]The Definition 11 is a refinement of the definition of Petri Nets provided in van der Aalst (2000), which includes the concept of initial marking in the tuple. This refinement is also reflected in all the following definitions.

1. *there is one source place $i \in P$ such that $\bullet i = \emptyset$.*
2. *there is one sink place $o \in P$ such that $o\bullet = \emptyset$.*
3. *every node $x \in P \cup T$ is on a path from i to o.*

Definition 13 (Enabling & Firing Rules of WF-net). *Given a WF-net N=(P, T, F, M), a transition $t \in T$ and a marking M of N, t is enabled at M, denoted as $M[t\rangle$, if and only if, there is at least one token each in all $p \in \bullet t$. If $M[t\rangle$ holds and transition t is fired, a new marking M' of N is reached, which removes a token each from each $p \in \bullet t$ and puts a token in each $p \in t\bullet$. This is denoted as $M \xrightarrow{t} M'$.*

Definition 14 (Occurrence Sequence). *Given a WF-net $N = (P, T, F, M)$ and markings $M_0, M_1, \ldots, M_n$ of N, if $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ holds, then $\sigma = \langle t_1, t_2, \ldots, t_n\rangle$ is an occurrence sequence leading from M_0 to M_n.*

The initial marking of a WF-net is i, where there is one token in the source place i, and the end marking of a WF-net is o. A *trace* in a WF-net represents an occurrence sequence from the initial marking i to the end marking o.

Definition 15 (Labelled WF-net). *A labelled WF-net $N = (P, T, F, M, l)$ is a WF-net (P,T,F,M) with some labelling function $l \in T \nrightarrow \mathcal{U}_A$, where $\mathcal{U}_A$ is some universe of activity labels. Let $\sigma_v = \langle a_1, a_2, \ldots, a_n\rangle \in \mathcal{U}_A{}^*$ be a sequence of activities and M, M' be two markings of N. $M[\sigma_v \rhd M'$ if and only if there is a sequence $\sigma \in T^*$ such that $M[\sigma\rangle M'$ and $l(\sigma) = \sigma_v$.*

With this definition, we only have the *visible and labelled* transitions in the net. For a set of traces of a WF-net $\mathfrak{T}^+(N)$, $\mathfrak{T}^+ = \{\sigma_\Theta | i[\sigma_\Theta\rangle o\}$ is the set of all visible traces in the net, where $\Theta = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a set of all occurrence sequences. The idea behind the notion of a labelled WF-net is that a trace of visible transitions corresponds to a possible execution sequence of the process, where the visible transitions correspond to the tasks executed by the process. One can argue, however, that there might be some other (invisible) traces that could still affect the compliance checking of a business process model. However, invisible traces might consist of tasks representing invisible actions. These invisible actions are used for routing purposes only and might not represent any task from a business point of view (Gambini et al., 2011; Wen et al., 2010). In contrast, we use visible traces because tasks in a trace represent some activity and might have significance from a business process compliance perspective. Furthermore, some literals representing obligations might be associated with the tasks of a trace. Hence, we limit our attention to visible traces only for compliance checking.

Next, we look at *how a WF-Net* can be annotated with the compliance requirements. First, the definition of the language is provided.

Definition 16 (Literal)**.** *Let A be the set of all atomic propositions. The set of literals is* $\mathscr{L} = \{a, \neg a | a \in A\}$.

In the rest of this section, we concentrate on a consistent set of literals, which can be understood as either a (partial) interpretation (that is, an assignment of truth value), or equivalently, a (partial) description of a state.

Definition 17 (Consistent Set)**.** *A set of literals* $\mathscr{L}$ *is consistent if and only if* $\mathscr{L}$ *does not contain any pair of literals* $l, \neg l$.

The next step is to enable a process to have states attached to the tasks, depending on which trace they appear in.

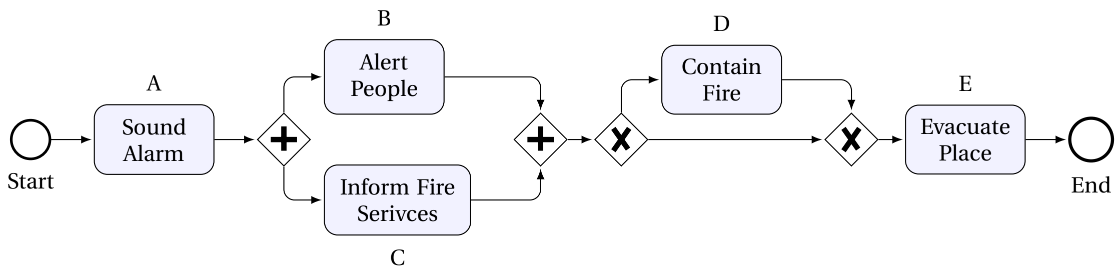
Definition 18 (Annotation)**.** *Let N be a WF-net and* $\mathfrak{T}^+$ *be the set of visible traces of* N*. An annotation Ann is a function* $Ann \colon \mathfrak{T}^+ \times \mathbb{N} \mapsto 2^{\mathscr{L}}$ *such that for every* $t \in \mathfrak{T}^+$ *and every* $n \in \mathbb{N}$, $Ann(t, n)$ *is a consistent set of literals.*

The idea of the above definition is that $Ann(t, n)$ returns the state obtained after the execution of the n-th task (visible transition) in the (visible) trace t.

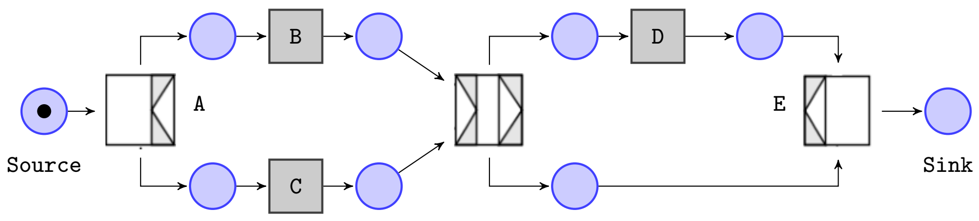
Definition 19 (Annotated WF-net)**.** *An annotated WF-net is a pair* $\langle N, Ann \rangle$*, where* $N = (P, T, F, M, l)$ *is a labelled WF-net, and Ann is an annotation function.*

Next, the concepts behind the above-mentioned definitions are explained with a small (abstract) example. As stated earlier, a process can be represented using any process modelling language (for example, Business Process Modelling Notation [BPMN], Event Process Chains [EPC]). Such a process model can be transformed into a Petri net/WF-net by making use of translation rules, as shown in (Dijkman et al., 2008; Ouyang et al., 2006, 2009). Figure 4.1 shows a simple BPMN process (with AND/XOR splits and joins) and its corresponding WF-net.

Now, consider the abstract BPMN model in Figure 4.1a as an emergency evacuation process with compliance requirements. Let us assume that Task *A* is ‘*sound alarm*’, task *B* is ‘*alert people*’, task *C* is ‘*inform fire services*’, task *D* is ‘*contain*



(a) BPMN Model



(b) WF-Net

Figure 4.1: Transformation of the BPMN Model into an equivalent WF-Net

fire', and task E is '*evacuate place*'. Assuming that semantic annotations are written in some language, we consider the annotations consisting of two propositions: p meaning 'the alarm has sounded' and q meaning 'a small fire to contain'. Four possible traces of this process are as follows:

$$t_1 : \langle A, B, C, D, E \rangle$$
$$t_2 : \langle A, C, B, D, E \rangle$$
$$t_3 : \langle A, C, B, E \rangle$$
$$t_4 : \langle A, B, C, E \rangle$$

After the execution of task A, we have the state 'alarm has sounded', which can be represented as

$$State(t_1, 1) = State(t_2, 1) = State(t_3, 1) = State(t_4, 1) = \{p\}$$

for all traces. After executing the next two tasks (B and C) also common to all traces, it is possible to have different annotations for these traces. For example, in traces t_1 and t_2, we reach:

$$State(t_1, 3) = State(t_2, 3) = \{p, q\}.$$

In contrast, we reach the following state for t_3 and t_4:

$$State(t_3, 2) = State(t_4, 3) = \{p, \neg q\}.$$

In t_1 and t_2, we check whether the fire is small enough that it can be contained (task D) before evacuating (task E); otherwise, we directly evacuate (task E) in t_3 and t_4. It can be seen that the information we have after the execution of tasks B and C varies depending on the trace being examined. For example, from trace t_1, we know that the fire is small enough and it is possible to contain the fire, represented as $State(t_1, 4) = \{p, q\}$. In contrast, trace t_3 informs us that it is not possible to contain the fire and, thus, we have to evacuate; that is, $State(t_3, 4) = \{p, \neg q\}$.

Note that different states can be obtained from different traces, even though the same tasks are being executed and the same end state can be reached from different traces. However, each visible trace uniquely determines the sequence of states obtained by executing the trace. Thus, in what follows, whenever clear from the context, we use the term trace to refer to a sequence of tasks and the corresponding sequence of states.

Remark 3. *Here we are not concerned with how the sequences of states corresponding to the execution of a process are obtained. The task of specifying how the annotation function Ann is implemented is left to specific compliance applications. However, one can use the update semantics approach (as described in Ghose and Koliadis, 2007) or by using EC to model the inertia of effects from one task to the next (as demonstrated in Goedertier and Vanthienen, 2006c), or by using the I-propagation approach for logical state representation (as described in Governatori et al., 2008a; Hoffmann et al., 2009).*

4.2.1 Modelling Obligations

In this section, we provide refined definitions of the obligation classes presented in Chapter 3. These refined definitions provide the same semantics of the norm classes in Section 3.4; they demonstrate that these semantics can be easily modelled with any formal language, and show *how these concepts* can be used to model the various notions of norms classes for business process compliance checking.

The revised definitions include the annotation function Ann and visible process traces (see Definition 14 and Definitions 18–19 respectively). Accordingly, these definitions reflect *when a particular type of obligation is in force* in the WF-net, and *when we have a violation of the obligation* if it is not in the annotation function *Ann*. We extend the *Force* function with $\mathfrak{T}^+$, a set of visible traces to redefine the obligation in force function.

Definition 20 (Obligation in force)**.** *Given a WF-net N, let $\mathfrak{T}^+$ be the set of visible traces of N. We define a function Force*: $\mathfrak{T}^+ \times \mathbb{N} \mapsto 2^{\mathscr{L}}$.

The function *Force* associates with each task in a trace a set of literals, where these literals represent the obligations in force for the combination of task and trace. These are among the obligations that the process need to fulfil to comply with a given normative framework. Given that a visible trace is a sequence of tasks, the second argument of *Force* indicates the index of a task in the visible trace given in the first argument. For example, $Force(t,3) = \{p,q\}$ specifies that p and q are obligatory in the third task of trace t. In the rest of this section, we supply definitions that specify when a process has to fulfil the various obligations (depending on their type) to be deemed compliant.

Definition 21 (Punctual Obligation)**.** *Given a WF-net N and a visible trace $t \in \mathfrak{T}^+$, an obligation o is a* punctual obligation *in t if and only if*

$$\exists n \in \mathbb{N} : o \notin Force(t, n-1),\ o \notin Force(t, n+1), o \in Force(t, n).$$

the obligation o is in Force at n in t.

A punctual obligation o in force at n in t is violated *if and only if $o \notin Ann(t,n)$.*

A punctual obligation o (represented as a literal) is in force on one task n in a trace t; that is, $o \in Force(t,n)$. Note that it might be the case that there are multiple instances in which the obligation is in force. The obligation is violated if what the obligation prescribed is not achieved in, or by, the task when the obligation enters into force. This is represented by the literal not being in the set of literals associated with the task in the trace; that is, $o \notin Ann(t,n)$.

Definition 22 (Persistent Obligation)**.** *Given a WF-net N and a visible trace $t \in \mathfrak{T}^+$, an obligation o is a* persistent obligation *in t if and only if*

$$\exists n, m \in \mathbb{N} : n < m, o \notin Force(t, n-1), o \notin Force(t, n+1), \forall k : n \leq k \leq m, o \in Force(t,k)$$

the obligation o is in Force between n and m in t.

A persistent obligation is an obligation in force in an interval (a contiguous set) of tasks in a process; that is, o is in force at $k - th$ task between n and m in the visible trace t.

Definition 23 (Achievement Obligation)**.** *Given a WF-net N and a visible trace $t \in \mathfrak{T}^+$, an obligation o is an* achievement obligation *in t if and only if $\exists n, m \in \mathbb{N}, n < m$ such that o is a persistent obligation in force between n and m in t.*

An achievement obligation o in Force between n and m in t is violated *if and only if*

(a) *o is* preemptive *and $\forall k : k \leq m,\ o \notin Ann(t,k)$ or*

(b) *o is* non–preemptive *and $\forall k : n \leq k \leq m,\ o \notin Ann(t,k)$*

As mentioned in the previous chapter (Definition 5), an achievement obligation is in force in a contiguous set of tasks in a trace. The violation depends on whether we have a preemptive or a non–preemptive obligation. For a *preemptive obligation o*, we have a violation if no state before the last task in which o is in force has o in its annotations.

A *preemptive obligation* is in force at task k in a set of contiguous tasks between n and m, where m is the task when the obligation enters in force, and m is the deadline by when the obligation has to be discharged. The obligation o is violated if o does not appear in the annotations associated with all tasks preceding m. Note that a preemptive obligation can be complied with even before the obligation is in force. Thus, one might ask why we bother with the task when the obligation enters into force. The reason is that having (or not having) an obligation at a particular time could be the trigger for other deontic effects.

For a *non–preemptive obligation*, the set of states one has to consider for determining whether the obligation has been violated is limited to those defined by the interval in which the obligation is in force.

Definition 24 (Maintenance Obligation)**.** *Given a WF-net N and a visible trace $t \in \mathfrak{T}^+(N)$, an obligation o is a* maintenance obligation *in t if and only if $\exists n, m \in \mathbb{N}, n < m$ such that o is a persistent obligation in force between n and m in t.*

A maintenance obligation o in Force between n and m in t is violated *if and only if*

$$\exists k : n \leq k \leq m, o \in Ann(t,k).$$

Similar to an achievement obligation, a *maintenance obligation* is in force in an interval. The difference is that the obligation has to be complied with for all tasks in the interval; otherwise, we have a violation. Another difference is that deadlines are not required to detect the violation of maintenance obligations; for an achievement obligation, however, violations are detected at deadlines (Hashmi et al., 2014).

Definition 25 (Compensation)**.** *A* compensation *is a function Comp*: $\mathscr{L} \mapsto 2^{\mathscr{L}}$.

Remark 4. *We remind the reader that Defintion 25, with minor adjustments, corresponds to the compensation function defined in Definition 7.*

Definition 26 (Compensable Obligation)**.** *Given a WF-net N and a visible trace* $t \in \mathfrak{T}^+(N)$*, an obligation o is* compensable *in t if and only if* $Comp(o) \neq \emptyset$ *and* $\forall o' \in Comp(o), \exists n \in \mathbb{N} : o' \in Force(t)n$.

Definition 27 (Compensated Obligation)**.** *Given a WF-net N and a visible trace* $t \in \mathfrak{T}^+(N)$*, an obligation o is compensated in t if and only if it is violated and for every* $o' \in Comp(o)$ *either:*

1. *o' is not violated in t, or*
2. *o' is compensated in t.*

Definition 28 (Perdurant Obligation)**.** *Given a WF-net N and a visible trace* $t \in \mathfrak{T}^+(N)$*, an achievement obligation o is a* perdurant obligation *in t with a deadline d if and only if o is in force between n and m and* $n < d < m$.

A perdurant obligation o with deadline d in force between n and m is violated in t if and only if

$$\forall j, j \leq d, o \notin Ann(t, j)$$

4.2.2 Business Process Compliance

The set of (visible) traces of a given business process describes the behaviour of the process insofar as it provides a description of all possible ways in which the process can be correctly executed. Accordingly, for the purpose of defining what it means for a process to be compliant, we will consider a process as the set of its (visible) traces.

Intuitively, a process is compliant with a given set of norms if it does not violate those norms. Given that, in general, it is possible to perform a business process in many ways, we can have two notions of compliance: (a) a *fully* compliant process; and (b) a *partially* compliant process.

> *A process is (fully) compliant with a normative system if it is impossible to violate the norms while executing the process.*

The intuition about the above notion is that no matter what way the process is executed, its execution does not violate the normative system. The second notion considers the case where there is an execution of the process that does not violate the norms.

> *A process is (partially) compliant with a normative system if it is possible to execute the process without violating the norms.*

Based on the above intuition, we can give the following definitions. We first define when a trace is compliant, and then extend that notion to cover a process.

Definition 29 (Compliant Trace)**.** *Given a WF-net N and a trace t in $\mathfrak{T}^+$. Let $O(t)$ be the set of obligations in force in t, i.e., $O(t) = \bigcup_{n \in \mathbb{N}} Force(t)n$.*

1. *A trace t is* strongly compliant *if and only if no obligation $o \in O(t)$ is violated in t.*
2. *A trace t is* weakly compliant *if and only if every violated obligation $o \in O(t)$ is compensated in t.*

Definition 30 (Compliant Process)**.** *Let N be a WF-net.*

1. *N is fully compliant if and only if every trace $t \in \mathfrak{T}^+$ is compliant.*
2. *N is partially compliant if and only if there exists a compliant trace $t \in \mathfrak{T}^+$.*

Note that a refinement of Definition 30 is possible. Thus, we can distinguish between strongly and weakly compliant processes. This is simply achieved by passing the strong/weak parameters to the traces. For example, a process is strongly compliant if all its visible traces are strongly compliant.

Except for Definition 30, the definitions given in this section can be used across the entire lifecycle of a process: *design-time, run-time* and *log analysis.* As was pointed out in Remarks 1 and 2, the states and obligations in force have to be determined by compliance applications and implementations. For example, the annotations associated with a task at run-time or log-analysis will be obtained from the running instance, or extracted from the log and the data sources related to the process; while at design-time, such information can be provided by business analysts or extracted from the schemas of the databases and data sources linked to the process, by using the data schema extraction methodology proposed by (Hashmi et al., 2012).

Definition 30 can be used at design time in what is called *compliance-by-design,* as proposed by (Governatori and Sadiq, 2009; Sadiq et al., 2007); that is, verifying that a process complies with regulations before deploying that process. Notice that, the definition is not suitable for checking compliance at run-time (also called 'conformance') or auditing (log analysis), since it is possible that some visible traces are never executed (run-time) or were not executed (auditing). For these two cases, one has to apply Definition 29 to the executed traces, and to the traces of instances of a process recorded in a log.

4.3 Compliance Checking Approach

Generally compliance rules are written in natural language (c.f. those found in legal documents or policy documents). To enable automatic compliance checks, these rules need to be formalised in a machine-readable format. Essentially, the formalisation of compliance rules is language dependent, and the choice of a formal language depends on the business analysts. With formalised rules, we can have the types deontic modalities; for example, obligations and permissions etc.

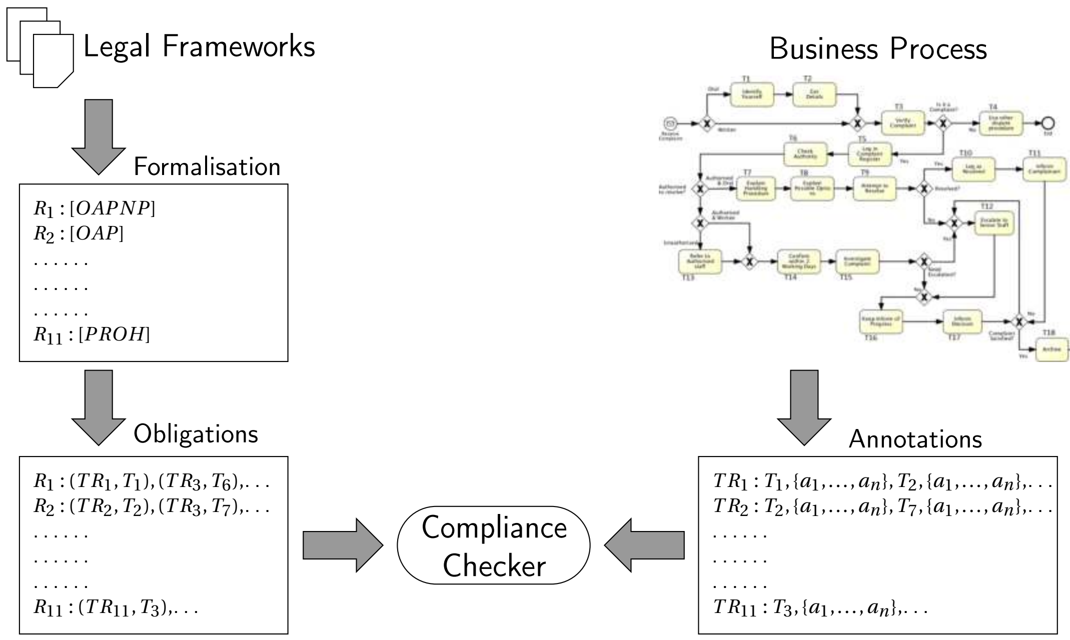

Legal Frameworks
Business Process
Formalisation
$R_1 : [OAPNP]$
$R_2 : [OAP]$
......
......
......
$R_{11} : [PROH]$
Obligations
Annotations
$R_1 : (TR_1, T_1), (TR_3, T_6), \ldots$
$R_2 : (TR_2, T_2), (TR_3, T_7), \ldots$
......
......
......
$R_{11} : (TR_{11}, T_3), \ldots$
Compliance Checker
$TR_1 : T_1, \{a_1, \ldots, a_n\}, T_2, \{a_1, \ldots, a_n\}, \ldots$
$TR_2 : T_2, \{a_1, \ldots, a_n\}, T_7, \{a_1, \ldots, a_n\}, \ldots$
......
......
......
$TR_{11} : T_3, \{a_1, \ldots, a_n\}, \ldots$

Figure 4.2: Business Process Compliance: Abstract Framework

Figure 4.2 illustrates the key concepts behind our approach for business process compliance. Rules impose conditions on the tasks to control the behaviour of processes. Business processes are annotated with rules for compliance checking purposes. These annotations are usually formalised rules and the data is parsed in the tasks at design-time. However, at design-time, very limited information is available about the real data on which a process operates on. Thus, for design-time compliance checking, business analysts provide abstract values and attributes to annotate processes. These abstract annotations can be used to verify the compliant behaviour of a business process at *design-time*.

By contrast, at *run-time*, processes are annotated with real values and attributes that are, again, provided by the business analysts. Regardless of whether compliance is checked at design-time or run-time, all that needed is visible traces consisting of annotated tasks of a process.

To sum up, given a business process and a set of norms, checking whether a business process is compliant with the set of norms amounts to the following operations:

1. *determining the deontic effects (and their type) of the set of norms*
2. *for each task in each trace of the process*:
   a) *determining what is the state corresponding to the task,*
   b) *determining what are the obligations in force for the task,* and
   c) *checking whether the obligations in force have been fulfilled, violated (and for compensable obligation, whether they have been compensated for), or whether the judgment is postponed to the next task in the trace (according to the semantics presented in Section 4.2.1).*

These steps are then illustrated in details, and we show *how the proposed compliance checking approach* can be used to check the compliance of business processes. For this purpose, the next section discusses a complaint-handling process as a motivating example to elaborate each step of the presented compliance checking approach.

4.4 Motivating Example: Complaint Handling Process

This section includes, a short description of a complaint-handling process as a motivating example. This example is then analysed in details in Section 4.3 to illustrate how the definitions given in Section 4.2.1 can be used to check whether a business process complies with a particular normative framework. In particular, the way in which the designed complaint handling process has to satisfy a number of different types of compliance requirements obtained from the internal policy document is described (see Table 4.1). The first column of Table 4.1 shows the *rule*

ID. The natural language description, the specific obligation type, and the deontic effects it might produce are given in the second column. The rule R_1, for example, is a *non–preemptive, non–perdurant achievement obligation* [OANPP], and the R_1 describes that any received complaint must be resolved at the earliest opportunity. Accordingly, the deontic effects for any received complaint for R_1 is *resolve_complaint*. Meanwhile, R_4 specifies that all the received complaints must be acknowledged, and two options are provided: (1) immediately acknowledge complaint received in person or by phone, and (2) acknowledge a written complaint within two working days. The R_4 stipulates two different obligations: a *punctual non–preemptive, perdurant obligation* [OPNPP] for (1); and a *non–preemptive, perdurant achievement obligation* [OANPP] for (2) respectively. The deontic effects that R_4 produces are to acknowledge a received complaint (see Table 4.1 for the description, types, and deontic effects of the rules related to the complaint-handling process).

Figure 4.3 depicts the overview of the procedure followed to resolve a complaint as a BPMN process model. According to the policy guidelines, the first step in the process is to determine whether the received complaint is an oral complaint or a written complaint. If it is an oral complaint, a staff member will identify him or herself, and details are gathered from the complainant before proceeding with the complaints-handling process. The staff member then verifies whether the received complaint meets the requirements of a legitimate complaint as defined in Section 9 of the policy. If the received complaint does not meet the definition of a complaint, alternative dispute procedures are adopted (These alternatives are is beyond the scope of this process).

After a complaint has been determined as a legitimate complaint, the staff member must decide whether (s)he has the appropriate authority to handle the complaint. If so, then the complaint will go through the complaints-handling process with the staff member as its handler. Otherwise, the complaint is referred to an authorised staff member and the complainant is informed. The authorised staff member explains the process and the available options, and attempts to resolve the complaint immediately if it is an oral complaint. If the complaint is resolved, then it is logged as such and the complainant is informed of the decision.

For a written complaint, an authorised staff member will confirm the process within two working days. A complaint is escalated to a senior staff member if it cannot be resolved, if the complainant is not satisfied, or if the staff member decides that it

Table 4.1: The Compliance Requirements of Complaints-handling Process

Rule ID	Policy Description (Compliance Controls/Specifications)
R_1	*Staff receiving a complaint will aim to resolve it at the earliest opportunity or at least confirm that the complaint will receive attention.*
	Type: Obligation, Achievement, Non–preemptive, Non–perdurant **Deontic Effect**: *resolve_complaint*
R_2	*Where the client is not satisfied with the initial response to the complaint, they will be given the option to progress the issues through the formal complaints-handling process outlined in the complaints-handling procedure.*
	Type: Obligation, Achievement, Preemptive, Perdurant) **Deontic Effect**: *provide_escalation_options*
R_3	*Staff will treat all complaints fairly and impartially, as is their obligation under the code of conduct.*
	Type: Obligation, Maintenance, Perdurant **Deontic Effect**: *treat_fairly*
R_4	*All complaints will be acknowledged:* *(1) immediately where complaints are made orally or by phone,* *(2) within 2 working days for written complaints.*
	Type-1: Obligation, Punctual, Non–preemptive, Perdurant **Type-2**: Obligation, Achievement, Non–preemptive, Perdurant **Deontic Effect**: *acknowledge_complaint*
R_5	*All complainants kept informed about the progress of the matter, particularly if delays occur.*
	Type: Obligation, Achievement, Non–preemptive, Non–perdurant **Deontic Effect**: *inform_progress*
R_6	*Complainants will not be subject to any form of prejudice, loss of services, or be disadvantaged in any way as a result of having complained.*
	Type: Obligation, Maintenance, Perdurant **Deontic Effect**: $\neg$*disadvantage*
R_7	*Complaints will be treated with an appropriate level of confidentiality. Information about complaints will only be shared on a need–to–know basis, both within the agency and externally.*
	Type: Obligation, Maintenance, Perdurant) **Deontic Effect**: *ensure_confidentiality*
R_8	*Reasons will be provided for decisions made in relation to complaints received.*
	Type: Obligation, Achievement, Non–preemptive, Perdurant **Deontic Effect**: *provide_reasons*
R_9	*If complaints do not meet the conditions in section 9, the department may set limits or conditions on the handling of their complaint.*
	Type: Permission **Deontic Effect**: *limit_complaint*
R_{10}	*Unauthorized staff cannot handle complaints(either oral or written).*
	Type: Prohibition, Maintenance, Perdurant **Deontic Effect**: *authorized*

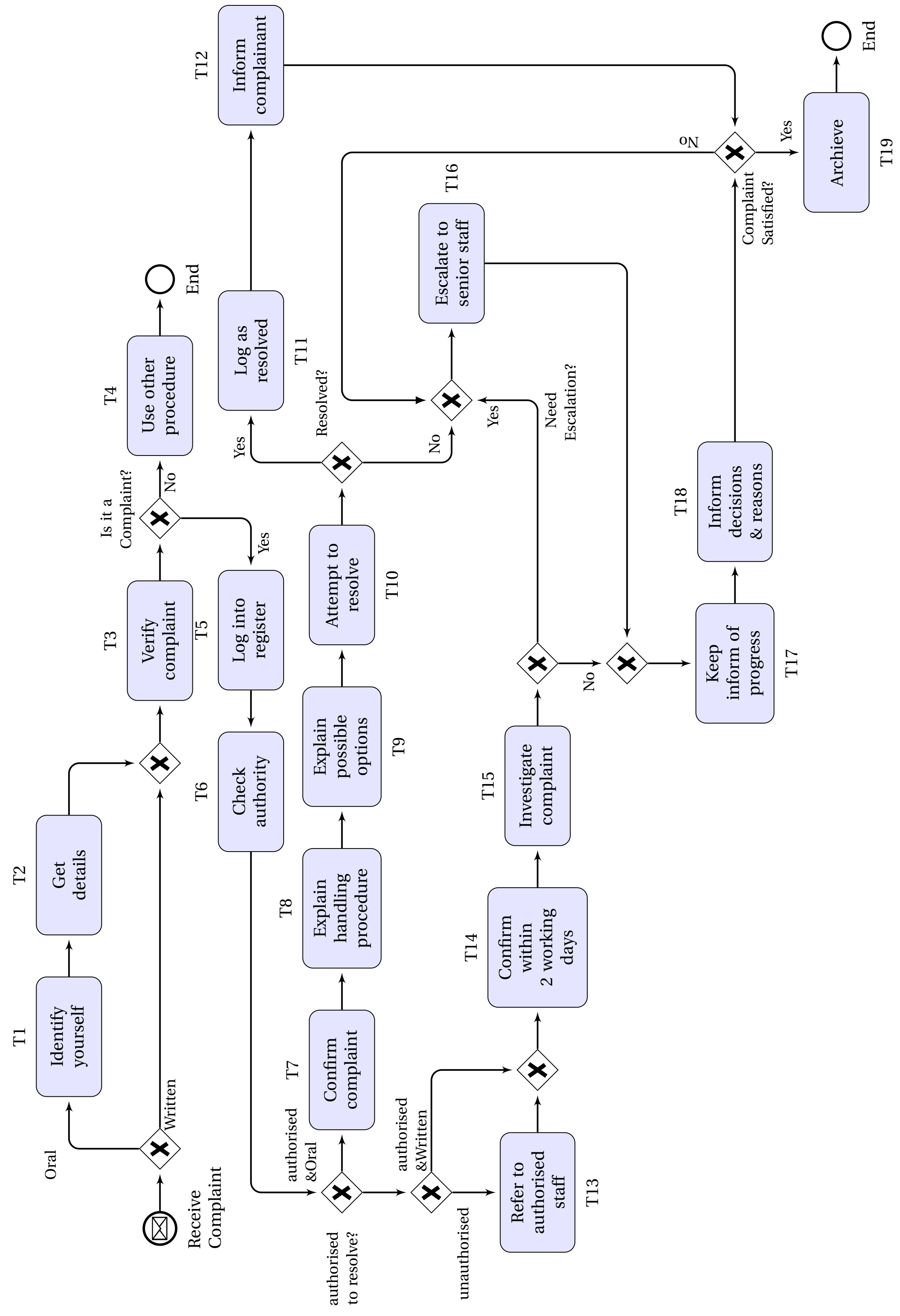


Figure 4.3: A Complaint Handling Process

needs to be escalated. While the complaint is being investigated, the complainant is being kept informed. When a decision has been reached, the complainant is informed of the decision. When the complainant is satisfied with the decision, the complaint process is closed off and the details archived.

4.4.1 Compliance Checking of Complaint Handling Process

Table 4.1 illustrates rules applicable to the complaint handling process. For each rule, we have also identified the obligation triggered by the rule. These rules are of different types and relevant to one or more tasks in the aforementioned process.

Compliance of every rule cannot be automatically checked for several reasons; for example, the rule might be vaguely described, or only partial information is available (see Awad, 2010). Rule R_1 in the complaint-handling process is one such type of rules that has been vaguely defined. For example, the *'earliest opportunity'* does not clearly specify by what time the obligation has to be fulfilled. However, R_1 is an achievement obligation applicable from T_3 and the obligation triggered by it remains in force until the obligation has been fulfilled. R_4 is a punctual obligation (for an oral complaint) and an achievement obligation (for a written complaint) where the received complaint has to be acknowledged within 2 working days. Rules R_3, R_6, and R_7 are maintenance obligations applicable from the beginning of the process. They must be complied with for all the instances of the complaint handling process.

To determine whether the obligation has been complied with, regardless of when an obligation it comes into force and at which task in the process, one has to consider all the traces of the process including the task from where the obligation enters into force. Thus, the first step is to consider all the traces of a process. Given that there is a loop in the process model, the number of traces is infinite. While this is not a problem for the theoretical compliance model, for practical purposes we have to consider a finite number of them. In practice, loops typically have exit conditions; accordingly, we can limit the analysis to the case where each loop is expanded once, and in the case of a nested loop, the external loop passes from the origin of the loop twice: where the internal loop is executed, and when the internal loop is skipped. In this case the second time, the effect will be annotated with the exit condition. This procedure is applied recursively for more deeply nested loops.

The complaint-handling process generates the following (finite) set of traces.

$$
\begin{aligned}
\mathfrak{T}_p^+ = \{ & t_1 = \langle T_1, T_2, T_3, T_4 \rangle, \\
& t_2 = \langle T_1, T_2, T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_3 = \langle T_1, T_2, T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_4 = \langle T_1, T_2, T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_5 = \langle T_1, T_2, T_3, T_5, T_6, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_6 = \langle T_1, T_2, T_3, T_5, T_6, T_{14}, T_{15}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_7 = \langle T_1, T_2, T_3, T_5, T_6, T_{14}, T_{15}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_8 = \langle T_1, T_2, T_3, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_9 = \langle T_1, T_2, T_3, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{19} \rangle, \\
& t_{10} = \langle T_1, T_2, T_3, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{11} = \langle T_1, T_2, T_3, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{16}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{12} = \langle T_3, T_4 \rangle, \\
& t_{13} = \langle T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{14} = \langle T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{15} = \langle T_3, T_5, T_6, T_{13}, T_{14}, T_{15}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{16} = \langle T_3, T_5, T_6, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{17} = \langle T_3, T_5, T_6, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{18} = \langle T_3, T_5, T_6, T_{14}, T_{15}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle, \\
& t_{19} = \langle T_3, T_5, T_6, T_{14}, T_{15}, T_{17}, T_{18}, T_{19} \rangle \}
\end{aligned}
$$

The next step is to determine what are the effects of the tasks in the trace, as each task is annotated with one or more effects (or sets of effects); we refer to these effects as *annotations.* Hence, we consider which literals are relevant to each task in the trace. We use the *Ann* function defined (see Definition 18) in Section 4.2. To improve readability, in the rest of this section we use the annotation function *Ann* as:

$$Ann(trace, task, integer) = \{ \textit{ set of (consistent) literals} \}$$

This means that we also include the name of the task in its signature.
We now take a significative trace, trace t_{11}, to illustrate how the function populates the states corresponding to the tasks in a trace. Trace t_{11} is as follows:

$$t_{11} = \langle T_1, T_2, T_3, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{16}, T_{17}, T_{18}, T_{16}, T_{17}, T_{18}, T_{19} \rangle$$

Based on the availbe information we populate the function *Ann* for t_{11} as follows:[2]

$$
\begin{aligned}
t_{11}: \langle & Ann(t_{11}, T_1, 1) = \{receive\_complaint, oral, identify\_yourself\} \\
& Ann(t_{11}, T_2, 2) = Ann(t_{11}, T_1, 1) \cup \{get\_details\}, \\
& Ann(t_{11}, T_3, 3) = Ann(t_{11}, T_2, 2) \cup \{verify\_complaint\}, \\
& Ann(t_{11}, T_5, 4) = Ann(t_{11}, T_3, 3) \cup \{valid\_complaint, register\_complaint\}, \\
& Ann(t_{11}, T_6, 5) = Ann(t_{11}, T_5, 4) \cup \{check\_authority\}, \\
& Ann(t_{11}, T_7, 6) = Ann(t_{11}, T_6, 5) \cup \{authorised, acknowledge\_complaint\}, \\
& Ann(t_{11}, T_8, 7) = Ann(t_{11}, T_7, 6) \cup \{explain\_handling\_procedure\}, \\
& Ann(t_{11}, T_9, 8) = Ann(t_{11}, T_8, 7) \cup \{explain\_options\}, \\
& Ann(t_{11}, T_{10}, 9) = Ann(t_{11}, T_9, 8) \cup \{attempt\_resolution\}, \\
& Ann(t_{11}, T_{16}, 10) = Ann(t_{11}, T_{10}, 9) \cup \{\neg resolve\_complaint, escalate\}, \\
& Ann(t_{11}, T_{17}, 11) = (Ann(t_{11}, T_{16}, 10) - \{escalate\}) \cup \{inform\_progress\}, \\
& Ann(t_{11}, T_{18}, 12) = Ann(t_{11}, T_{17}, 11) \cup \{inform\_decision, provide\_reasons\}, \\
& Ann(t_{11}, T_{16}, 13) = (Ann(t_{11}, T_{18}, 12) \; - \{inform\_decision, provide\_reasons\}) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{\neg satisfied, escalate\}, \\
& Ann(t_{11}, T_{17}, 14) = Ann(t_{11}, T_{16}, 13) \cup \{provide\_escalation\_options\}, \\
& Ann(t_{11}, T_{18}, 15) = Ann(t_{11}, T_{17}, 14) \cup \{inform\_decision, provide\_reasons\}, \\
& Ann(t_{11}, T_{19}, 16) = (Ann(t_{11}, T_{18}, 15) \; - \{\neg resolve\_complaint, \neg satisfied\}) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad \cup \{satisfied, archive, resolve\_complaint\}\rangle
\end{aligned}
$$

The integer and task appearing in the *State* function indicate, respectively, the step of the process and the task (to be) executed at that step. Apart from its own, each task in the trace can inherit effects from its previous tasks to determine the state corresponding to the task. These effects can be accumulated as the information grows for every subsequent task in the trace. These effects are computed based on the updated semantics where if the effects of previous tasks are in conflict with the effects of the current task, the effects of previous tasks are replaced with current ones. For example, the state reached after task T_2, namely $State(t_{11}, T_2, 2)$, accumulates the effects of its previous task T_1, and also has its own effects $\{get\_details\}$. Similarly, task T_3 accumulates the effects of tasks T_1 and T_2 producing $State(t_{11}, T_3, 3)$. In other

[2]The annotations for each task can be given by domain experts or can be extracted from databases or forms related to the tasks (see Hashmi et al., 2012, for details).

Table 4.2: Applicable Rules and Obligations in Force for Trace t_{11}

Task, Step	Rules	Obligations in Force
$T_1, 1$	R_1, R_3, R_6, R_7	$Force(t_{11}, T_1, 1) = \{$ *resolve_complaint, treat_farly, ¬disadvantage, ensure_confidentiatly*$\}$
$T_2, 2$		$Force(t_{11}, T_2, 2) = Force(t_{11}, T_1, 1)$
$T_3, 3$		$Force(t_{11}, T_3, 3) = Force(t_{11}, T_2, 2)$
$T_5, 4$	R_4, R_5	$Force(t_{11}, T_5, 4) = Force(t_{11}, T_3, 3) \cup \{$ *acknowledge_complaint, inform_progress* $\}$
$T_6, 5$	R_{10}	$Force(t_{11}, T_6, 5) = Force(t_{11}, T_5, 4) \cup \{$*authorized*$\}$
$T_7, 6$		$Force(t_{11}, T_7, 6) = Force(t_{11}, T_6, 5)$
$T_8, 7$		$Force(t_{11}, T_8, 7) = Force(t_{11}, T_7, 6)$
$T_9, 8$		$Force(t_{11}, T_9, 8) = Force(t_{11}, T_8, 7)$
$T_{10}, 9$		$Force(t_{11}, T_{10}, 9) = Force(t_{11}, T_9, 8)$
$T_{16}, 10$	R_8	$Force(t_{11}, T_{16}, 10) = Force(t_{11}, T_{10}, 9) \cup \{$*provide_reasons*$\}$
$T_{17}, 11$		$Force(t_{11}, T_{17}, 11) = Force(t_{11}, T_{16}, 10)$
$T_{18}, 12$		$Force(t_{11}, T_{18}, 12) = Force(t_{11}, T_{17}, 11) - \{$*provide_reasons*$\}$
$T_{16}, 13$	R_2, R_5, R_8	$Force(t_{11}, T_{16}, 13) = Force(t_{11}, T_{18}, 12) \cup \{$*provide_reasons, provide_escalation_options* $\}$
$T_{17}, 14$		$Force(t_{11}, T_{17}, 14) = Force(t_{11}, T_{16}, 13)$
$T_{18}, 15$		$Force(t_{11}, T_{18}, 15) = Force(t_{11}, T_{17}, 14)$
$T_{19}, 16$		$Force(t_{11}, T_{19}, 16) = Force(t_{11}, T_{18}, 15)$

cases, some effects obtained in previous tasks can be removed or their truth value can be changed. For example, the first time we pass through task t_{17} (step 11) we remove the *escalate* flag that was raised in the previous task indicating the complaint was escalated. The change of polarity of literals is exemplified at step (16) where the negative ¬*resolve_complaint* and ¬*satisfied* are removed and replaced by their positive counterparts, *resolve_complaint* and *satisfied*.

The next step is to identify which rules are applicable to trace t_{11} on which task and when in order to determine which obligations are in force. Table 4.2 illustrates when the various rules become active in trace t_{11} (when they begin to produce their deontic effects), and when the various obligations are in force.

Four rules are effective at T_1. Rule R_1, whose deontic effect is an achievement obligation, becomes active as soon as a complaint is received, and remains active until the complaint is resolved. The other three rules, that is, R_3, R_5 and R_6, are for maintenance obligations and never terminate for all instances of the process. No rules are associated with T_2, T_3, tasks T_7–T_{10} or with the tasks in the last three steps of the trace. R_4 and R_5 produces achievement obligations, and their effects enter in force at step 4 (task T_5) when the complaint has been deemed valid. Rule R_{10}

kicks in at task T_6, and its deontic effect is a maintenance obligation (where the staff is authorised to handle the complaint, or alternatively, the prohibition to handle a complaint, if not authorised).

Rule R_8 is triggered twice. The first trigger is at step 10 and the corresponding non–preemptive obligation is in force for that step and the next one, when the obligation is fulfilled. Thus, the obligation *provide_reasons* is no longer in force for step 12. The new decision in step 13, reinstates that non–preemptive obligation. The non–preemptiveness of the obligation implies that the previous discharging instance does not count for the instance of the obligation in force from step 13.

It is easy to verify that the trace is compliant for rules R_1, R_2, R_4, R_5, R_6, R_8, R_9 and R_{10}: the achievement obligations triggered by rules R_1, R_2, R_4, R_5, R_8 are fulfilled, respectively at steps: 16, 13, 5, 11, and 11 and at 15 for the two instances of R_8. The maintenance obligation of rule R_8 is maintained from step 5, when it enters in force, and remain active until the end of the process. R_9 is casually complied with since it is a permission, and it cannot result in a non–compliant situation. Finally, the maintenance obligations of R_3, R_6 and R_7 are not fulfilled. This is due to a lack of information of about what their obligations mean in term of the given process.

4.5 Evaluation

In this section, we report on an evaluation of the framework against real processes and norms. The aim of the section is to provide evidence that all types of obligations are eventually present in real life compliance scenarios.

The evaluation was carried out using Regorous.[3] Regorous is an implementation of the compliance checking methodology proposed by (Governatori and Sadiq, 2009; Sadiq et al., 2007), where the normative provisions relevant to a process are encoded in PCL (Governatori and Rotolo, 2010a,b), and the tasks of a process are annotated with sets of literals taken from the language used to model the norms.

The Regorous module for checking compliance generates the traces of the given process and cumulates the annotations attached to tasks, using an update semantics to determine the state corresponding to a task in a trace; in other words, in a case where a literal from the current task is the complementary of from a previous task, the old literal is removed and a new one is inserted. PCL offers comprehensive support

[3]Regorous Compliance Checker, available at `https://www.regorous.com`

for modelling and compliance checking of various types of obligations and, for every step in a trace, it retrieves the state corresponding to the task being examined. Based on state, PCL determines the obligations in force for the current task. Finally, it checks if the obligations have been fulfilled or violated based on the semantics discussed in the previous section (For the full details of PCL mechanisms, (see Governatori and Rotolo, 2010b).).

Regorous was tested against the novel Australian Telecommunication Consumers Protection Code 2012 (TCPC). The code specifically mandates that every Australian entity operating in the telecommunication sector must provide a certification that their day-to-day operations comply with the code.

Table 4.3: Number and types of obligations and permissions in Section 8 of TCPC

Punctual Obligation	5	(5)
Achievement Obligation	90	(110)
Preemptive	41	(46)
Non–preemptive	49	(64)
Non–perdurant	5	(7)
Maintenance Obligation	11	(13)
Prohibition	7	(9)
Non–perdurant	1	(4)
Permission	9	(16)
Compensation	2	(2)

The test was limited to Section 8 of the TCPC code concerning the management and handling of consumer complaints. The section was manually mapped to PCL. This section of the code contains approximately 100 commas, in addition to approximately 120 terms (given in the Definitions and Interpretation section of the code). The mapping resulted in 176 PCL rules, containing 223 PCL (atomic) propositions (literals). The formalisation of Section 8 of the TCPC required all types of obligations described in Section 4.2.1. Table 4.3 reports the number of distinct occurrences and, in parentheses, the total number of instances (some effects can have different conditions under which they are effective).

The evaluation was carried out in cooperation with an industry partner operating in the sector of the code. The PCL formalisation of TCPC Section 8 was reviewed and informally approved for the purpose of the exercise by the regulator. The industry partner did not have formalised business processes. Thus, we worked with their domain experts from the industry partner (who had not been previously exposed to BPM technology, but who were familiar with the industry code) to draw process models to capture the existing complaint-handling and management procedures and other related activities covered by Section 8 of TCPC code. As the result, we generated and annotated 6 process models. Five of these 6 models were limited in size and can be checked for compliance in seconds; we were thus able to identify non–compliance issues in the processes and to rectify them. In the simplest and most frequent cases, the modification required was simply to ensure that some type of information was recorded in the databases associated with the processes. Other cases needed an addition to simple activities (tasks) either after or before other tasks; for example, the need to make a customer aware of documents detailing the escalation procedure after the unsatisfactory outcome of a non–escalated complaint.

These two types of non–compliance were detected by unfulfilled achievement obligations, and they were the results of new requirements in the 2012 version of the code. Another case of non–compliance was related to ensuring that a particular activity did not occur in a particular part of the process. Finally, there were some cases where a combination of the above issue was needed (for example, a novel way to handle in-person or by-phone complaints) where totally new sub-processes were designed.

The largest process contains 41 tasks, 12 decision points, XOR splits, (11 binary, 1 ternary). The shortest path in the model has 6 tasks, while the longest consists of 33 tasks (with 2 loops); the longest path without loop is 22 tasks long. The time taken to verify compliance for this process was approximately to 40 seconds on a MacBook Pro 2.2Ghz Intel Core i7 processor with 8GB of RAM (limited to 4GB in Eclipse).

4.6 Related Work

We now consider related work in the business process compliance checking domain and compare it with work presented in this chapter.

In the recent past, a number of approaches to checking compliance of business process models have been reported in the literature (Bonazzi and Pigneur, 2009; Elgammal et al., 2011a; Kharbili and Stein, 2008; Liu et al., 2007; Ly et al., 2012). As discussed previously, the requirement of a preventive approach *compliance by design* for business process compliance. This literature can be divided into two distinct categories: compliance by design and post-design compliance checking. In the first approach, new business process models are fed with business rules as input; a process model, on the other hand, is checked against compliance requirements when a process has completed the design phase.

Lu et al. (2007) objectively show how to enforce compliance requirements to avoid the chance of potential rules violations. While, similar works reported by Goedertier and Vanthienen (2006c); Milosevic et al. (2006b) provide an effective solution to achieving design-time compliance, compliance checking will still be required if changes are made to the process model and new business rules are introduced. In addition to that, the emphasis of these approaches remains on the structural compliance of a process model, and the data aspect has been largely ignored.

Goedertier and Vanthienen (2006c) achieve design-time business process compliance using rule sets with permissions and obligations, and proposed PENELOPE, a declarative language to specify compliance rules. From these rules, ENELOPE generates a state space and a BPMN model from these rules that is compliant by design. However, this approach concentrates on acyclic processes only, and the data and data constraints aspect in the business rules is not included. An artefact-centric business process modelling approach has been recently proposed in Lohmann (2012), which exhibits how artefact-centric business processes can be canonically extended to take compliance rules into account. As these business rules can express constraints on the execution of actions, it is claimed that the data information can also be taken into account; however, it is not clear whether the model will be semantically annotated with the data, and how data constraints will be modelled. If the business process model is semantically annotated then where this data will come from (that is, the source of data for annotations).

With respect to post-design process compliance, Awad et al. (2009) discuss a temporal logic query-based approach for specification, verification, and explanation of violations of data-aware compliance rules. The approach employs extended BPMN-Q to realise the business rules, including the data aspects, to increase the expressiveness of their previously proposed language (given in Awad et al., 2008a). The authors used *past linear temporal logic* (PLTL) to formalize the rules; however, temporal logic poses a problem in that it provides structural compliance only, and does not distinguish different normative positions; that is, it does not indicate how these normative positions can be represented, or how the data is associated with the rules. Moreover, this proposed approach comes under the post design compliance checking approach. To measure the compliance distance between the process model, and a rule, an automated approach was introduced in Lu et al. (2008). The degree of compliance is checked on a scale from 0 to 1, but the data aspect has not been covered.

Ramezani et al. (2013) report a conformance checking approach based on Petri-Net patterns and alignments. They created a repository of 55 control-flow based compliance rules spanning over 15 distinct categories, including compliance rules for data, resources and organisational rules. The collected rules were formalised in terms of Petri-nets rather than logics. For conformance checking, they employed alignment techniques from van der Aalst et al. (2012) to analyse process compliance with the formalised Petri-Net patterns. If the patterns are consistent with the compliance rules, the execution behaviour is consistent. However, if any deviant behaviour is observed, a violation of the rule is reported and the alignment shows the reason(s) for the deviations. The approach is useful for checking the compliance of control-flow related rules; however, this only provides the structural compliance of the rules. In addition, conformance checking of business processes against the business rules has different specifications and properties than those in the legal domain. Thus, the proposed approach is not suitable for compliance checking of the normative requirements.

In addition to, design time compliance approaches, there are other compliance checking approaches that focus on run time (Hee et al., 2010; Maggi et al., 2011a; Ramezani et al., 2013). However, since, the focus of this thesis is on design-time compliance checking, we do not discuss such approaches here see, Chapter 2 for further details.

4.7 Summary

The main contribution of this chapter is the formal foundational framework that provides the baseline for properly modelling the legal component of business process compliance, to evaluate the ability of a CMF to represent the norms with which a system needs to comply. The presented framework comprises several formal models that provide formal specifications (definitions) of business processes and the formal specifications (definitions) of the various notions of norms, and an approach to integrate these formal models. The provision and integration of these formal models is imperative for business process compliance modelling and checking. We began with the notion of a business process model to describe the sequence of states corresponding to the execution of the process, and the use of WF-nets to model the specifications of process models. To provide the formal model of the norms, we the used these sequences of states to provide the formal semantics of different classes of the norms (discussed in Chapter 3). Finally, we provided the definitions of what it means to comply with a norm, and to violate a norm. These formal models provided the basis for the compliance checking approach, with the idea of semantically annotating business processes to validate their compliant behaviour.

To validate the effectiveness of these formal models and compliance checking approach, we used a real life complaint-handling process and showed *how we can correctly model the legal component of compliance to check the compliance of business processes annotated with the compliance rules.* For this purpose, we first determined which rules (with their types) are relevant to the complaint-handling process, and then determined the state corresponding to the tasks and the obligations that are in force for that task. We then manually attached the relevant obligations to the generated traces (states) of the processes, to determine whether the complaint-handling process is compliant with the set of the applicable norms, using the formal semantics proposed in this chapter. The main feature of the presented formal semantics that give the formal specifications of norms and business processes is their flexibility and they are independent of any specific formal language. They can be simply transformed into any other formal language without having any complexity. In addition, the presented framework is not limited to design-time compliance of norms: it can equally be used for conformance checking or auditing (that is, log analysis of business processes). The possible uses of the presented framework include, but not limited to, establish the mappings between the language and the semantics of a CMF and the semantics and definition of compliance as provided in this chapter.

The next chapter presents, the conceptual evaluations of seven frameworks selected according to pre-defined criteria, to examine what these frameworks can do in terms providing the compliance management support for modelling and reasoning with the norms, using the approaches proposed in these frameworks. In particular, these evaluations are based on the constructs existing CMFs provide to model various classes of normative requirements; in particular, the new classes and formal semantics of norms proposed in this and the previous chapter.

Part III

Evaluating BPC Frameworks

CHAPTER 5

CONCEPTUAL EVALUATION OF CMFS

5.1 Background

In the introduction, we discussed various compliance management strategies (for example, *design–time, run–time* and *post–execution time*), and pointed that a wealth of CMFs exist. These CMFs bear specific functional and operational capabilities, support specific compliance requirements for specific domains. Each of the CMFs is grounded on different concepts and incorporates different conceptual and formal models. The strength of a CMF largely depends on its conceptual and formal models. If a CMF is not conceptually sound, it might be not suitable to provide the certification of compliance that is acceptable to accredited certifying organisations.

Previously, we provided a classification model of normative requirements giving a detailed ontology of different types of normative requirements, and formal semantics defining each class of the classification model. Hence, to support the overall objectives of the study, and given the new classification model the natural question is *whether existing CMFs are conceptually sound to provide reasoning and modelling support for each type of normative requirements?* In this chapter, we evaluate the conceptual foundations of seven CMFs selected according to four–point criteria. The classification model of normative requirements (presented in Chapter 3) and the compliance checking approach (discussed in Chapter 4) provide the basis for these evaluations. Specifically, we address the following questions: *(a) how do existing CMFs address the compliance problem? (b) what are the underlying conceptual models of these CMF? (c) what constructs do they provide to represent different types of normative requirements? (d) how do they link these requirements to business processes?* Addressing these question will provide a better understanding on various functionalities of CMFs especially the norms modelling constructs provided in these CMFs. Also, it will identify issues related to modelling normative requirements as it is generally acknowledged observation that no modelling language can ever be perfect to represent the legal knowledge. Hence, a CMF will be vulnerable to criticisms with regard to the classes of the norms that it might be unable to represent.

The remainder of this chapter is structured as follows: Section 5.2 discusses the evaluation approach. Section 5.3 provides detailed evaluations of the selected CMFs, where we examine the use of their conceptual foundations to deal with the normative requirements related to regulatory compliance. Section 5.4 is a short discussion of the evaluation results, and the shortcomings of evaluated CMFs is then presented; Section 5.5 discusses related work, and Section 5.6 summarises the chapter's contributions.

5.2 Approach

This section presents the research approach used to conduct the evaluations. A systematic case study based evaluation strategy Clark and Dawson (1999) was adopted; this allowed us to start the evaluations with minimal information available on the CMFs. Following this systematic strategy, a three–step approach was employed; in the first step, the objectives for this evaluation were defined; a set of evaluation criteria meeting the evaluation objectives was then determined; and in the last step, a range of CMFs were selected for evaluation. Figure 5.1 illustrates the overall evaluation approach.

Evaluation Objectives: The main objective of the conceptual evaluation was to examine the conceptual foundations of existing CMFs. We specifically looked at the conceptual approach that a framework proposes for checking the compliance of business processes, and reasoning support for various obligations types discussed in Chapter 3. More specifically, the goals of the conceptual evaluation were to determine:

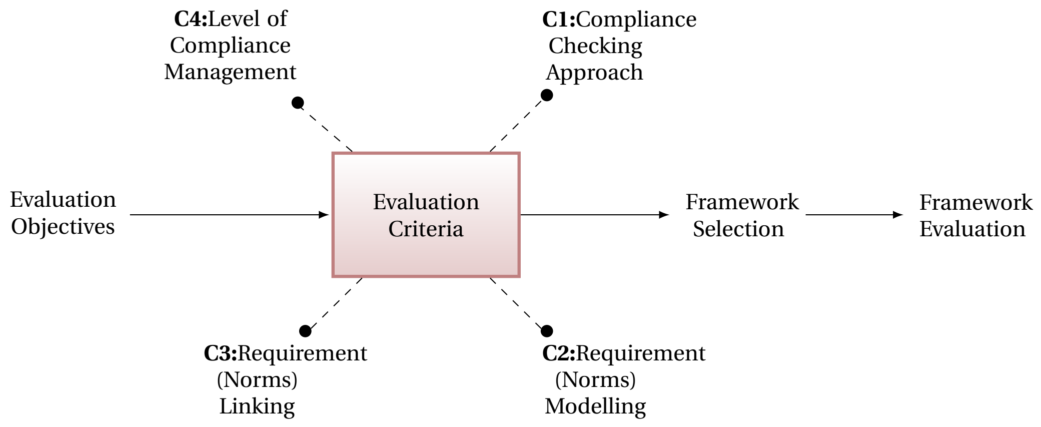

C4:Level of Compliance Management
C1:Compliance Checking Approach
Evaluation Objectives
Evaluation Criteria
Framework Selection
Framework Evaluation
C3:Requirement (Norms) Linking
C2:Requirement (Norms) Modelling

Figure 5.1: Conceptual Evaluation Approach

1. *what constructs are provided for modelling the norms, and the formal language for doing so*;
2. *how the norms are linked to business processes for compliance checking;*
3. *the level of compliance management support—that is, the framework contribution type, and whether all obligations types can be supported.*

Evaluation Criteria: We determined a four–step selection criteria to identify representative frameworks for this evaluation namely:

1. *Compliance checking approach:* This criterion is divided into two dimensions, namely: *process lifecycle* aspects and the *orientation* of the compliance checking approach. The lifecycle aspect aims to examine whether the proposed approach in a CMF is design–time, run–time or post–execution time compliance checking. The orientation dimension, on the other hand, looks at whether the approach in a CMF is focused on the verification (or validation), or is purely business oriented (see Chapter 2 for details on various approaches). We use this criterion because of the focus of this study is on design–time compliance management frameworks.
2. *Requirements modelling:* This criterion enabled us examine how CMFs model different types of compliance requirements and the formal logic they use to do so. Specifically, we used this criterion to identify the modelling constructs for a specific obligation type proposed in a CMF,

and the ways in which it is modelled. For this purpose, we followed the approach used in (van der Aalst et al., 2002), where the authors evaluated the workflow patterns by describing the conditions that should hold for a pattern to be applicable under real life situations. For our purpose, we also aimed to examine whether constructs proposed in a CMF can fully capture the compliance rules from the real life regulations. Hence, our recourse was to check the one–to–one mapping between the construct provided in a CMF and the obligation types of our classification model, to determine whether the construct can fully represent the intuition of the obligation type. Notice that, the earlier discussed obligations types only describe the temporal properties of obligations with respect to their validity and effects of the violations. They do not cover the obligations related to the data and resources aspect of a business process. Hence, we do not consider the constructs that are concerned with the rules prescribing the conditions on the data and resource aspects of the business processes.

3. *Requirements linking:* Using this criterion, we identified how a CMF links compliance rules with business process models. The reason for using this criterion, as argued in previous chapter, was to verify the compliance of norms requirements needed to provide both the model of normative requirements and that of a business process. Furthermore, if both the models are represented in different formal languages, then a bridging mechanism serving as an interface between the formalisation of the business processes and the formalisation of norms is mandatory. For effective compliance checking of the compliance rules, it is imperative that they are properly modelled and linked to the business processes; if they are not properly modelled and linked, the results of the compliance checking approaches are not reliable. Hence, with this criterion we aimed to look at the way in which existing CMFs link the compliance requirements to ensure that the compliance checking process remains transparent to the stakeholders.

4. *Level of compliance management:* This criterion describes the level of support a CMF provides; that is, modelling, linking, compliance checking, and handling violations of the norms. Only CMFs that provide

full compliance management support were selected those merely provide a compliance checking algorithm or a modelling language were not considered.

Sample Frameworks Collection: Although we reviewed and analysed a large number of CMFs - 19 in total (for example, MASTER framework (MASTER, 2008); REO–Toolkit (Arbab, 2004); COMPAS (Elgammal et al., 2011a); SeaFlows (Ly et al., 2010b, 2012); REALM (Giblin et al., 2005); NORMC (Kazmierczak et al., 2012); and PSA@R (Rieke et al., 2014) to name but a few), and 45 compliance checking approaches (such as static compliance checking, logic-based, and pattern-based approaches), we refrained from undertaking a systematic literature survey, as done in (Abdullah et al., 2010; Becker et al., 2012; El Kharbili, 2012). Rather, we selected CMFs based on expert discussions, and those mostly cited in the literature. As a first step, we conducted an extensive literature search using the methodology from Bandara et al. (2011), and along the regulatory compliance management frameworks dimensions listed in (El Kharbili, 2012). To search the resources, we used the keywords based on these dimensions such as compliance rules modelling, verification, violation management, temporal rules; and also the keywords closely related to the evaluation criteria. This resulted in the extraction of more than 100 articles covering a range of compliance dimensions. Not all articles were relevant to the evaluation objectives, and unrelated articles were discarded. Only articles that contained key terms such as design–time, run–time, compliance framework, compliance checking approach were selected. Based on these, we selected seven frameworks meeting the evaluation criteria. We believe that the selected CMFs give a fair representation of most compliance approaches; for example, design–time, run–and post–execution based approaches. In addition, they are widely cited in the literature for example, PCL (Sadiq et al., 2007), BPMN–Q (Awad et al., 2008a), PENELOPE (Goedertier and Vanthienen, 2006c), and COMPAS (Elgammal et al., 2011b) as illustrated in Table 5.1.

Table 5.1: Articles and # of Citations of Selected CMFs between December 2013–August 2015

Framework	Article (Author/Year)	2013	2015	Difference
PCL	- (Sadiq et al., 2007)	269	342	73
	- (Governatori et al., 2006b)	209	240	31
	- (Lu et al., 2007)	74	100	26
	- (Governatori and Sadiq, 2009)	62	86	24
	- (Governatori and Rotolo, 2010a)	29	48	19
BPMN–Q	- (Awad et al., 2008a)	159	210	51
	- (Awad, 2007)	95	127	32
	- (Awad et al., 2008b)	48	61	13
	- (Awad et al., 2011)	38	70	32
SEAFLOWS	- (Ly et al., 2012)	64	98	34
	- (Ly et al., 2010a)	51	62	15
	- (Ly et al., 2010b)	44	62	18
	- (Ly et al., 2011)	43	66	23
Auditing Framework	- (Ghose and Koliadis, 2007)	166	201	35
	- (Hinge et al., 2009)	25	38	13
DECLARE	- (van der Aalst et al., 2009)	214	297	83
	- (Montali et al., 2010)	98	138	40
	- (Maggi et al., 2011a)	66	101	35
	- (Maggi et al., 2011b)	15	33	18
	- (Ramezani et al., 2012a)	13	25	12
	- (Ramezani et al., 2013)	2	12	10
	- (Ramezani et al., 2012b)	17	49	32
COMPAS	- (Elgammal et al., 2010)	30	45	15
	- (Schumm et al., 2010)	22	35	13
	- (Türetken et al., 2011)	11	28	17
	- (Túretken et al., 2012)	8	22	14
	- (Elgammal et al., 2011a)	9	18	9
PENELOPE	- (Goedertier and Vanthienen, 2006c)	131	163	32
	- (Goedertier and Vanthienen, 2006b)	32	38	6

5.3 Conceptual Evaluation of Compliance Frameworks

Chapter 3 discussed various classes of normative requirements of the classification model, based on the temporal validity of norms and the effects they produce. Essentially, different types of obligations represent different types of compliance requirements and different properties thus have different complexities. Hence, a '*one–size–fits–all*' modelling approach is far from being satisfactory from a conceptual point of view, and a CMF that does not represent the various nuances of obligations is not conceptually sound. Hence, a conceptually weak CMF might not be suitable to provide any certification of compliance acceptable to accredited certifying organisations. In this section, we use the classification model as a template to examine the conceptual foundations of the selected frameworks, by specifically looking at what constructs they provide to represent the classificatory classes of our classification model. The first CMF evaluated is PENELOPE.

5.3.1 PENELOPE

PENELOPE (Goedertier and Vanthienen, 2006c) is a formal language–based declarative CMF that captures obligations and permissions constraints imposed on an organisation's processes by business policies. Aiming to provide a *design–time* compliance verification, the language uses an algorithm that progressively generates the *state space* and *control–flow* of a business process. The state space in the PENELOPE–generated process is a set of obligations and permissions that are active at a particular state. The interaction between the generated process models flows from one state to another, and all the states are enumerated until no obligation or permission holds at a state, or if there is a violation that cannot be repaired. Once all the states are computed, the algorithm draws the BPMN model for a role involved in the business interaction. The tasks of the process are drawn whenever an obligation set contains all obligations fulfilled by a role in the activity.

PENELOPE allows the modelling of the interaction between all involved partners and any violations from a third partner are represented as time–out events in the generated BPMN model. In addition, errors and end events are drawn if there is a violation of an obligation or permission by a role in a state. With the designed compliant process models, various inconsistencies such as deontic conflicts can be identified. The deontic assignments in the PENELOPE are modelled using Event–Calculus (EC) that provides a rich semantics with which to reason about the normative requirements. Table 5.2 illustrates the EC based deontic properties proposed in PENELOPE.

Table 5.2: Deontic Properties of PENELOPE (Goedertier and Vanthienen, 2006c)

Term	**Meanings**
$\texttt{Xor}(\alpha_1,\alpha_2)$	*compound activity* α_1 *XOR* α_2
$\texttt{Or}(\alpha_1,\alpha_2)$	*compound activity* α_1 *OR* α_2
$\texttt{And}(\alpha_1,\alpha_2)$	*compound activity* α_1 *AND* α_2
$\texttt{Oblig}(A,\alpha,\delta)$	*agent A must do the activity* α *by due date* δ
$\texttt{Perm}(A,\alpha,\delta)$	*agent A can do the activity* α *prior to due date* δ
$\texttt{CC}(A,\alpha_1,\delta_1,\alpha_2,\delta_2)$	*agent A must do activity* α_2 *by due date* δ_2 *after activity* α_1 *is performed prior to due date* δ_1
$(A) Terminates(\alpha, Oblig(A,\alpha,\delta),\tau) \longleftarrow \tau \le \delta$	
$(B) Terminates(\alpha, Perm(A,\alpha,\delta),\tau) \longleftarrow \tau \le \delta$	
$(C) Happens(violation(Oblig(A,\alpha,\delta)),\delta) \longleftarrow HoldsAt(Oblig(A,\alpha,\delta)) \wedge \sim Happens(\alpha,\delta)$	
$(D) Initiates(\alpha_1, Oblig(A,\alpha_2,\delta_2),\tau) \longleftarrow \tau \le \delta_1 \wedge HoldsAt(CC(A,\alpha_1,\delta_1,\alpha_2,\delta_2)),\tau)$	

Next, we examine in detail each of the deontic properties, and check whether these properties have some correspondence to the earlier discussed obligation types. Since the first three properties are related to the structure of the process—such as Choice (OR) and XOR–Split, and Parallel (AND) gateways—we discard them from our analysis and directly evaluate the deontic properties.

Deontic Property-1: (*Obligation*)
Term: $Oblig(A,\alpha,\delta)$
Description: The term *Oblig* is used to capture the notion of obligations, A is the subject of the obligation, α refers to the conditions (or actual contents) of the obligation, and δ is the deadline until which the obligation must be fulfilled.
Evaluation: In combination with the *Initiates* predicate and *Terminates* predicate, the term can be used to model achievement obligations. The predicates *Initiates* and *Terminates* capture the effects when an obligation enters into force; and when it is terminated, respectively, depending on whether the obligation is

fulfilled or removed. The term $Oblig(A,\alpha,\delta)$ defines the parameters of the agent who has to obey the contents of the obligation and fulfilled it by a deadline.
Correspondence: The term has one–to–one correspondence with the semantics of the achievement obligation (Definition 5).

Deontic Property-2: (*Permission*)
Term: $Perm(A,\alpha,\delta)$
Description: This deontic property is used to capture the notion of permission, where in the term A refers to the agent, α refers to the conditions (or actual contents) of obligation, and δ is the deadline until which the permission must be discharged.
Evaluation: Similar to property-1, the term *Perm* can be used to represent $permissions$ in combination with the *Initiates* and *Terminates* predicates.
Correspondence: One-to-one correspondence with *Permission*

Deontic Property-3: *Terminates* (Obligation / Permission)
Term: The term of the *Terminates* property for obligation and permission is as follows:

(1) **terminates-obligation**: *Terminates*$(\alpha, Oblig(A,\alpha,\delta),\tau) \longleftarrow \tau \le \delta$
(2) **terminates-permission**: *Terminates*$(\alpha, Perm(A,\alpha,\delta),\tau) \longleftarrow \tau \le \delta$

Description: The meanings of the *Terminates* predicate is that the event α terminates the obligation fluent α at time τ.
Evaluation: PENELOPE uses EC's *Terminates* predicate to terminate the effects of the normative notions of obligations and permissions. As mentioned earlier, obligations remain for a certain time period and then they are removed (either upon fulfillment of violation). The *Terminates* predicate indicates when an obligation or permission ceases to hold.
Correspondence: Though the predicate has no one-to-one mapping with our classes of obligations. However, in conjunction with *Initiates*, it can be used to capture the cessation of the effects of all obligations.

Deontic Property-4: (*Conditional Commitments*)
Term: $CC(A,\alpha_1,\delta_1,\alpha_2,\delta_2)$
Description: PENELOPE uses the term *CC* to model conditional commitments (mostly in the sense of contractual interactions). The key idea behind the

conditional commitments, as in most business scenarios and concrete applications, is that an agent commits himself to another agent to produce some effects when the antecedent of the conditional commitment holds. The meanings of the term is that an agent A must perform a certain activity (α_2) by the due date δ_2, only after the agent has performed activity α_1 before the due date α_1.
Evaluation: Since conditional commitments are not considered in our classification model, we do not evaluate this property. However, this deontic property can be useful from a structural compliance of a business process perspective, because a conditional commitment might represent the absence of the occurrence of an activity (that is, α_2) until another activity (α_1) does not occur in the interaction.
Correspondence: - NA -

Deontic Property-5: Initiates (*Conditional Commitments*)
Term: The term for the *Initiates* predicate for the conditional commitments is as follows:

$$\mathit{Initiates}(\alpha_1, Oblig(A, \alpha_2, \delta_2), \tau) \leftarrow \mathit{HoldsAt}(CC(A, \alpha_1, \delta_1, \alpha_2, \delta_2), \tau) \wedge \tau \leq \delta_1$$

Description: The meanings of the term is that the occurrence of event α_1 *Initiates* the obligation for the agent A to do α_2 by the deadline δ_2.
Evaluation: There is only difference between this property and deontic property–4; that is, it gives the initiation conditions when a conditional commitment that an agent has to fulfill enters into force.
Correspondence: - NA -

Handling the violations of obligations is one of the major requirements for a compliance framework, as argued in (Awad, 2010). Timely reporting of the violations not only allows the analysts to respond immediately but also saves much time and effort. PENELOPE uses EC's *Happens* predicate to represent violations for which the framework provides the following semantic properties:

Deontic Property-6: Happens (*Violation Handling*)
Term: The term for Happens predicate for the violations handling is as follows:

$$\mathit{Happens}(violation(Oblig(A, \alpha, \delta)), \tau) \leftarrow \mathit{HoldsAt}(Oblig(A, \alpha, \delta)) \wedge \sim \mathit{Happens}(\alpha, \delta)$$

Description: The meaning of the term is that the obligation to do α by the deadline δ is violated at time τ when the obligation fluent α holds and the event fulfilling the

obligation does not happen by the deadline.
Evaluation: As the violations in PENELOPE can only occur in the form of deadlocks situations and temporal conflicts, we understand that the *Happens* predicate is useful from the capturing of the occurrence of events perspective. This is because EC *Happens* predicate can effectively reason about the events and the changes resulting from the occurrences of events over time. However, the violation semantics can only indicate when an obligation is violated. Hence, the semantics property can be used to indicate the violation of obligations. However, the property cannot be used to reason about the effects the violation of an obligation might produce; for example, perdurance of the violated obligation or triggering of compensatory actions. Note that, since *preemptive achievement* obligations are fulfilled even before the obligation enters into force, they cannot be violated. Hence, no reasoning support is required for such obligations.
Correspondence: - NA -

The deontic properties discussed above are used to model obligations, permissions and conditional commitments; other obligation types cannot be represented with these properties. This is because Event-Calculus is not suitable for reasoning all types of obligations[1] (Hashmi et al., 2014). Essentially, the deontic properties for achievement and permission are only possible because these properties allow the explicitly definition of the deadlines in the form of precedence rules. This, in turn, is because the framework generates compliant processes from the rule sets of obligations and permissions. The main problem with PENELOPE is that it does not consider prohibitions under Close World Assumption (CWA) to avoid the anomalies that might occur because of incomplete knowledge of all the parties involved in the business interaction. While, PENELOPE, is able to represent the deadlock situations and temporal conflicts to represents violations, it cannot provide the reasoning support for the deontic conflicts. This is because the framework does not admit prohibitions or waived obligations. Table 5.10 illustrates the types of normative requirements that are supported by PENELOPE.

On the same note, it is not possible to handle the effects of the violations because the violation semantics discussed above can only indicate violation of the obligation. Because violation properties in PENELOPE are based on the notion of violations

[1]A detailed evaluation of the semantic properties, and the reasons EC is not capable of providing a full reasoning and modelling support for all types of olbigaiton, are discussed in Chapter 6.

without reparation (VWR), a penalty is imposed, whenever there is a violation of an obligation. No deontic properties are provided for capturing the effects of violations—that is, perdurance and compensatory actions; this is left to the analysts.

5.3.2 COMPAS

COMPAS (Elgammal et al., 2011a) is a comprehensive compliance governance framework that provides an all–around compliance support for service–oriented–architecture (SOA)–based systems. The framework adopts a model–driven development approach for designing compliant processes/services, using a view–based modelling framework and domain–specific languages to model the compliance concerns in process models (Daniel et al., 2009).

For compliance checking, business processes are annotated with compliance constraints in the form of (re–usable) process fragments. These fragments underline the required behaviour of the control-flow of a process model, and they are formalised using Linear Temporal Logic (LTL). The annotated process fragments are then assessed to validate the compliant behaviour of the process models at run–time, using event logs. A protocol component evaluates the generated event logs to check whether the process model complies with the behaviour described in the attached compliance constraints process fragment. If the monitoring protocol detects any non–compliant behaviour it reports a violation and publishes it as a violation event.

As far as the modelling of compliance requirements is concerned, COMPAS uses a compliance request language (CRL) (Elgammal, 2012; Elgammal et al., 2014) for modelling normative requirements. The core of the CRL is LTL–based graphical compliance patterns, which are high–level compliance templates to model the compliance constraints—predominantly, compliance requirements from the control flow (structural) perspective of business processes. In addition, most of these patterns are used by other frameworks, and more, recently additional patterns representing features specific to normative reasoning, such as exceptions to rules and compensation of violations have been included (Elgammal et al., 2014).

The CRL graphical patterns representing different types of compliance rules are categorised into three distinct categories of patterns: *atomic patterns*, *resources patterns*, and *timed patterns*. The atomic patterns aim to describe the requirements involving the ordering of occurrence of the process elements, as depicted in

Table 5.3: Atomic Patterns and CRL Expression (Elgammal et al., 2014)

Atomic Pattern	**Patter-Expssions**	**Description**
isAbsent	ϕ isAbsent	ϕ should not exist throughout the process model
Exists	ϕ Exists	ϕ should occur at least once within the process model
Bounded-Exists	ϕ BoundedExists $\leq 2*$	shows that ϕ must occurs at most 2 times within the process
	ϕ BoundedExists $\geq 2*$	shows that ϕ must occurs at least 2 times within the process
isUniversal	ϕ isUniversal	P should always be true throughout the process
Precedes	ϕ Precedes ψ	ψ is always preceded by ϕ
Chain-Precedes	ϕ *Precedes* (σ,τ)	meaning that a sequence of σ,τ must be preceded by ϕ
	(σ,τ) *Precedes* ϕ	ϕ must be preceded by a sequence of σ,τ
LeadsTo	ϕ LeadsTo ψ	ϕ must always be followed by ψ
Chain-LeadsTo	ϕ LeadsTo (σ,τ)	shows that ϕ must be followed by a sequence of σ,τ
	(σ,τ) *LeadsTo*ϕ	shows that a sequence of σ,τ must be followed ϕ
Exists-Often	ϕ Exists-Often	ϕ must occur frequently within the process model
DirectlyFollowedBy	ϕ DirectlyFollowedBy ψ	shows that required ϕ to be followed by ψ

Table 5.3. Some atomic patterns are based on Dwyer's property specification patterns (Dwyer et al., 1999), and categorised into *occurrence* and *ordering* patterns. Accordingly, *timed patterns* are concerned with the constraints that include temporal requirements, and *resources patterns* are related to constraints that are used to describe the recurring requirements pertaining to the resources such as authorisation or task assignment constraints. For our purpose, we only consider the patterns relevant to describe compliance rules based on the obligation types and discard those that are used to describe compliance constraints concerning the resources aspect of the business processes.

In addition to the atomic patterns, CRL defines *composite patterns*, as illustrated in Table 5.4. Composite patterns aim to describe more complex compliance rules, and are built conjunctively by combining multiple atomic patterns using boolean

Table 5.4: Composite Patterns and Equivalent CRL Expressions (Elgammal et al., 2014)

Composite Pattern	Description	Atomic Pattern Equivalence.
ϕ CoExists ψ	The presence of ϕ mandates that ψ is also present	(ϕ Exists) $\rightarrow$ (ψ) Exists)
ϕ CoAbsent ψ	The presence of ϕ mandates that ψ is also absent	(ϕ isAbsent) $\rightarrow$ (ψ) isAbsent)
ϕ Exclusive ψ	The presence of ϕ mandates the absence of ψ, and presence of ψ mandates the absence of ϕ	((ϕ Exists) $\rightarrow$ (ψ isAbsent)) $\wedge$ ((ψ Exists) $\rightarrow$ (ϕ isAbsent))
ϕ Substitute ψ	ψ substitutes the absence of ϕ	(ψ isAbsent) $\rightarrow$ (ϕ Exists)
ϕ Corequisite ψ	ϕ and ψshould either exist together or be absent together	(ϕ EXists) iff (ψ Exists)) = ((ϕ Exists) $\rightarrow$ (ψExists)) $\wedge$ ((ψ Exists) $\rightarrow$ (ϕ Exists))
ϕ MutexChoice ψ	Either ϕ or ψ exists but not any of them or both of them	(ϕ Exists)XOR(ψ Exists) = ((ϕ Exists) $\wedge$ (ϕ isAbsent)) $\vee$ ((ψ Exists) $\wedge$ (ϕ isAbsent))

logical operators such as `NOT, AND, OR, XOR, Implies` and `Iff` operators. These patterns are mapped into LTL formulas (see the description above) enabling the translation of CRL expressions into a set of LTL formulas. Essentially, these patterns are used to represent different types of compliance requirements.

We now discuss CRL compliance patterns in detail, and examine whether they have correspondence with various obligation types.

Pattern-1: (*isAbsent*)
Description: ϕ should not exist throughout the process model.
LTL Formula: $\mathsf{G}(\neg\phi)$
Evaluation: As the name implies, `isAbsent` is used to indicate that some activity ϕ must not hold in the whole trace or throughout the business process model. This is because *prohibitions* specify the conditions that the bearer must avoid during interaction. As a contrasting achievement, prohibitions do not prescribe deadlines; that is,if a constraint prescribes the prohibition of some action, this must be maintained throughout the execution of the process. Hence, the pattern `isAbsent` can be used to express compliance constraints prohibiting some actions. The `isAbsent` pattern has a duality relation with `Existence or Eventually` patterns.

Correspondence: The pattern has correspondence with *prohibitions*.

Pattern-2: (*Exists*)
Description: ϕ should occur at least once within the process model.
LTL Formula: $\mathsf{F}(\phi)$
Evaluation: The `Exists` (or eventually) pattern is used to specify that the execution of the process model contains the instance of some proposition ϕ. The existence pattern has a strong relationship with the `Absence` pattern because of the duality relation; it can be used to specify the negation and explicit queries for the existence of some propositions. As for the definition of persistent obligations, an achievement obligation can specify the conditions that might eventually hold in the future; thus, the pattern can be used to represent achievement obligation. Cases where the obligation specifies some additional conditions—that is, the occurrence of the proposition ϕ with some finite number of times—can be handled with the `Bounded Existence` patterns. COMPAS offers two `Bounded Existence` patterns for this purpose.

- The Bounded-Exists (ϕ BoundedExists $\leq 2*$) shows that ϕ must occur at most 2 times within the process.
  LTL Formula: $\neg\phi \mathsf{W}(\phi \mathsf{W}(\neg\phi \mathsf{W}(\phi \mathsf{W} \neg \mathsf{F}(\phi))))$

- The Bounded-Exists (ϕ BoundedExists $\geq 2*$) specifies that the proposition ϕ must occurs at least 2 times within the process.
  LTL Formula: $\neg\phi \mathsf{W}(\phi \mathsf{W}(\neg\phi \mathsf{W}(\phi)))$

Correspondence: The pattern can be used to represent *achievement* obligations.[2]

Pattern-3: (*isUniversal*)
Description: ϕ should always be true throughout the process.
LTL Formula: $\mathsf{G}(\phi)$
Evaluation: The pattern `isUniversal` (also called `'Always'or 'Global'`) aims to define the part of process execution that includes states in which the presence of proposition ϕ is always desired. As for the maintenance obligations, a special case of persistent obligations, the obligation conditions must hold for all the instances of

[2]The `BoundedExists` pattern can be useful for achievement, only if such conditions are prescribed by the norm. However, in every instance, an obligation can have independent conditions associated with it, with different objectives.

the interval in which the obligation is in force. The `isUniversal` pattern can be used to handle maintenance obligations. Accordingly, the pattern has a close relationship with the `Absence` and `Existence` patterns, and it can be equally applied in situations where the `isAbsent` pattern can be applied. This is because the universal presence of some proposition ϕ can be seen as the absence of its negation; that is, $\mathsf{G}(\neg\phi)$. Hence, it can also be useful for representing cases of *prohibitions.*

Correspondence: The pattern can be used to represent *maintenance* obligations (Definition 6) and *prohibitions*

Pattern-4: (*Precedes*)
Description: indicates that activity ψ is always preceded by another activity ϕ
LTL Formula: $\neg\psi\mathsf{W}\phi$
Evaluation: The `Precedes pattern` (also called *precedence*) describes the relationship between two propositions; for example, ϕ and ψ where the execution of ϕ is mandatory for the execution of ψ. In other words, the execution of the first proposition enables the execution of the second proposition. A common norm example for precedence is: a payment can be made only after an invoice is issued; that is, the issuance of the invoice is a pre–condition for make payment event. The `Chain Precedes` is the variant of the Precedes pattern COMPAS uses to capture the ordering of the sequence of activities after the execution of the first activity. Essentially, the aim of chain patterns is to define the requirements pertaining to complex pairing of individual proposition relations. The LTL formula for the `Chain Precedes` pattern is as follows:

- The Chain Precedes pattern specifies that a sequence of activities σ, τ must be preceded by the occurrence of activity ϕ.
  LTL Formula: $(\mathsf{F}(\sigma \wedge \mathsf{XF}(\tau))) \rightarrow (\neg\sigma)\mathsf{U}(\phi))$

The pattern is only useful for structural compliance checking of the processes where the norms can prescribe conditions on the ordering of the activity occurrence However, the pattern is not suitable for representing the semantics of any obligation modality in our classification model.
Correspondence: –NA– (Ordering pattern only)

Pattern-5: (*LeadsTo*)
Description: indicates that the execution of the activity ϕ must always be followed by the execution of activity ψ.
LTL Formula: $\mathsf{G}(\phi \rightarrow \mathsf{F}(\psi))$
The `LeadsTo` pattern (or *Response pattern*) is a cause–effect pattern describing the relationship between two propositions ϕ and ψ. Essentially, the relationship is established when the execution of ϕ (the cause) must be followed by the execution of the ψ (the effect). Generally, the properties of the `LeadsTo` pattern frequently occurs in the concurrent systems (Dwyer et al., 1999), and have a contrary relationship with the `Precedes` pattern; in other words, they cannot be equivalent. Accordingly, the variant `Chain-LeadsTo` indicates that activity ϕ must be followed by a sequence of σ, τ; and for a reverse interaction, the sequence of σ, τ must be followed by activity ϕ. The LTL formulas for the two cases of the `Chain-LeadsTo` pattern are respectively, as follows:

- **[Case-1:** ϕ **LeadsTo** (σ, τ) **]:** $\mathsf{G}(\phi \rightarrow \mathsf{F}(\sigma \wedge \mathsf{XF}(\tau)))$
- **[Case-2:** (σ, τ) **LeadsTo** ϕ **]:** $\mathsf{G}(\sigma \wedge \mathsf{XF}(\tau) \rightarrow \mathsf{X}(\mathsf{F}(T \wedge \mathsf{F}(\phi))))$

Similar to its counterpart `Precedes`, the pattern is useful only for structural compliance checking of the processes. However, given their property, they have different semantic conditions, which do not correspond to any of the obligation types.

Correspondence: –NA– (Ordering pattern only)

Pattern-6: (*Exists-Often*)
Description: indicates that proposition ϕ must occur frequently within the process model
LTL Formula: $\mathsf{GF}(\phi)$
Evaluation: The `Exist-Often` pattern implies the frequent occurrence of proposition ϕ during the whole execution of the process. Essentially, conversely to the `Exist-Bounded` pattern, this pattern can have indefinite occurrences of the propositions. From a legal norm perspective, the pattern is not suitable for providing semantic representation of any obligation type; however, from a structural compliance rule perspective, it can be used to model an activity that has to occur multiple times during the execution of the process. The pattern is comparable to the

`isUniversal` pattern; the only difference is that the occurrence of the proposition ϕ must hold for all the instances of the portion of the process where the obligation is in force.

In contrast, the `Exist-Often` proposition ϕ might appear in many instances of the part of the process (that is, ϕ can occur several times in part of the process). Hence, the pattern cannot be used to represent maintenance obligation because, with the `isUniversal` pattern, it might be possible that the obligation defining the conditions for proposition ϕ is triggered only once for the duration of the validity of the obligation. However, with this pattern, the obligation has to be triggered for every instance where the proposition needs to occur. Thus, it cannot give a full representation of the semantics of maintenance obligations. Accordingly, the pattern is dual of `Absence-Often`, and can be used to specify the negation and explicit queries for existing to define an instance of the absence pattern (Dwyer et al., 1999).

Correspondence: –NA– (Ordering pattern only)

Pattern-7: (*DirectlyFollowedBy*)
Description: shows that the required proposition ϕ is to be followed by another proposition ϕ.
LTL Formula: $\mathsf{G}(\phi \rightarrow \mathsf{X}(\psi))$
Evaluation: The `DirectlyFollowedBy` pattern defines the relationship between two propostions ϕ and ψ, where if second proposition ψ occurs then the first proposition ϕ must have occurred *immediately before* ψ. The pattern is useful only for structural compliance rules, as it does not seem to have conceptual relevance to the legal norm; thus, it cannot be used to represent any obligation type.

Correspondence: –NA– (Ordering pattern only)

As previously argued, what makes deontic effects distinctive from other normative effects is that they can be violated. Generally, violations lead to the imposition of penalties. In some violations cases, however, corrective measures can still make the process compliant. Hence, a CMF should be able to handle the compensatory actions to amend the violations. More recently, in addition to the patterns discussed above new patterns have been introduced in CRL (Elgammal et al., 2014), for the specification and verification of *compensation actions* namely: *Else* and *ElseNext*. These patterns are conjunctively built with *LeadTo* and *DirectlyFollowedBy* atomic patterns as:

$$\begin{aligned}&\phi(LeadsTo|DirectlyFollowedBy)\phi_1(Else|ElseNext)\\&\phi_2\dots(Else|ElseNext)\phi_n\end{aligned} \tag{5.1}$$

where ϕ is the rule condition, ϕ_1 is the primary action, and $\phi_2,\dots,\phi_n$ are compensatory actions. Essentially, the compensation pattern implements the *if–then–else* conditional structure of the compensatory rules. Accordingly, *DirectlyFollowedBy* and *LeadsTo* define the ordering of the primary activity, whether ϕ_1 directly occurs *immediately after* ϕ or at some time in the future. The LTL equivalence formula for compensatory patterns is as follows:

$$\begin{aligned}&\mathsf{G}(\phi\rightarrow\mathsf{F}|\mathsf{X}(\phi_1\wedge_{1\le i<n-1}\ (\mathsf{F}|\mathsf{X}(\phi_i NotSucceed)\wedge\\&(\phi_i NotSucceed\rightarrow\ \mathsf{F}|\mathsf{X}(\phi_{i+1})))))\end{aligned} \tag{5.2}$$

ϕ gives the antecedent of the compensatory rule, that is, the rule's conditions; ϕ_1 is the head of the rule representing the primary action that must be taken; $\phi_2,\dots,\phi_n$ represents the compensatory actions that must be taken if the conditions of the rule are violated; and i is a natural number; that is, $n\in\mathbb{N}$; and $\phi_i NotSucceed$ represents the decision point that checks whether ϕ_i holds.

In addition to the generic rules patterns, CRL offers patterns for modelling non–monotonic requirements. This allows CRL to model *exceptions*. More specifically, exceptions provide conditions under which the primary requirement might not hold. Following (Baral and Zhao, 2007), CRL has two patterns for exceptions: one for *strong exceptions* and one for *weak exceptions*: A strong exception on the primary rule mandates that whenever the strong exception holds, the primary rule *must* not hold. A weak exception, on the other hand, indicates that when the weak exception holds, the primary rule *might* or *might not* hold. The patterns for strong and weak exception are as follows:

1. strong exception: $[[R]]$*Pattern* and
2. weak exception: $[R]$*Pattern*

where $[[R]]$ and $[R]$ are the LTL formulas encoding the exception conditions and *Pattern* is the LTL formula corresponding to the primary requirement the exception applies to.

Given the potential recursive nature of exceptions (that is, having the ability to capture exceptions to exceptions), CRL recursively resolves the dependencies of the exception conditions, using the following translation to LTL

1. $[[R]]$*Pattern* is translated to $\phi \rightarrow \neg\psi$,
2. $[R]$*Pattern* is translated to $\phi \vee \psi$;

where ϕ is the LTL formula corresponding to R and ψ is the LTL counterpart of *Pattern*.

The above analysis of the COMPAS patterns shows that, currently, the framework is able to represent a fraction of obligation types, and that the support for all types of normative requirements is limited. This is because the use of LTL as its underlying formal language has limited its ability to give a faithful representation of legal norms in a conceptually rich and sound way.

5.3.3 DECLARE

Declare(Pesic and van der Aalst, 2006) is a prominent framework for *run–time* verification of constraint–based declarative models. The declarative models describe *what a model does* by specifying the business constraints as rules that *should not be violated.* The business knowledge in Declare is defined in terms of constraints using ConDec (Constraint Declarative, Pesic and van der Aalst (2006)[3]), a language which provides graphical notations to model the flows of business interactions. The Declare models (also templates) are enacted by a workflow engine that is used to verify the compliant interaction between the tasks in the model. The framework includes two types of constraints, that is, *mandatory* and *optional* constraints on the process models. In the Declare model, a process instance can only be active when there is no violation of the mandatory constraints and all the constraints are fully satisfied at the end of the execution of an instance. The verification results of each constraint of an active instance are expressed as *satisfied*, *temporarily violated*, and *violated*. In the case where all the constraints are satisfied, the activities are not executed any further; however, if there is a violation state, no further execution of the process would be allowed to satisfy the constraints. Accordingly, in the temporarily violated state, the constraints are not satisfied; however, there would be the possibility of satisfying the constraints.

[3]In November 2012, the name of the ConDec language changed to Declare (see `http://www.win.tue.nl/declare/2011/11/declare-renaming/`).

Table 5.5: Declare Constraint Patterns and Meanings (Pesic and van der Aalst, 2006)

Constraint Name	Meanings
Existence Constraints	
`Absence(`ϕ`)`	activity ϕ cannot be executed
`Absence(n+1,`ϕ`)`	activity ϕ can be executed *at most n times*
`Existence(n,`ϕ`)`	activity ϕ must be executed *at least n* times
`Existence(n,`ϕ`)`	activity ϕ must be executed *exactly n* times
`Init(`ϕ`)`	activity ϕ must be the first executed activity
Relation Constraints	
`Resp_existence([`ϕ`],[`ψ`])`	if ϕ is executed, then ψ must be executed *before or after*
`Coexistence([`ϕ`],[`ψ`])`	*neither* ϕ nor ψ is executed, or they *both* are executed
`Response([`ϕ`],[`ψ`])`	if ϕ is executed, then ψ must be executed *thereafter*
`Precedence([`ϕ`],[`ψ`])`	ψ can be executed only if ϕ has been *previously* executed.
`Succession([`ϕ`],[`ψ`])`	ϕ and ψ must be executed in *succession*; i.e.,ψ must follow ϕ and ϕ must precede ψ
`Chain_response([`ϕ`],[`ψ`])`	if ϕ is executed, then ψ must be executed *next*
`Chain_precedence([`ϕ`],[`ψ`])`	if ψ is executed, then ϕ must have been executed *immediately before* ψ
`Chain_succession([`ϕ`],[`ψ`])`	ϕ and ψ must be executed in *sequence*
`Alt_response([`ϕ`],[`ψ`])`	ψ is *response* of ϕ, and *between* every two executions of ψ, ϕ must be executed at least once
`Alt_precedence([`ϕ`],[`ψ`])`	ϕ is *precedence* of ψ, and *between* every two executions of ψ, ϕ must be executed at least once
`Alt_succession([`ϕ`],[`ψ`])`	ψ is *alternate response* of ϕ, and ϕ is *alternate precedence* of ψ
Negation Constraints	
`resp_absence([`ϕ`],[`ψ`])`	if ϕ is executed, then ψ can *never* be executed
`Not_Coexistence([`ϕ`],[`ψ`])`	ϕ and ψ *exclude* each other
`Neg_response([`ϕ`],[`ψ`])`	ψ cannot be executed *after* ϕ
`Neg_precedence([`ϕ`],[`ψ`])`	ϕ cannot be executed *before* ψ
`Neg_succession([`ϕ`],[`ψ`])`	ϕ and ψ cannot be executed in *succession*
`Neg_alt_response([`ϕ`],[`ψ`])`	ψcannot be executed between two occurrences of ϕ
`Neg_alt_precedence([`ϕ`],[`ψ`])`	ϕ cannot be executed between any two ψs
`Neg_alt_succession([`ϕ`],[`ψ`])`	ψ cannot be executed between any two ϕs and vise-versa
`Neg_chain_response([`ϕ`],[`ψ`])`	ψ cannot be executed *next* to ϕ
`Neg_chain_precedence([`ϕ`],[`ψ`])`	ϕ cannot be executed *immediately before* ψ
`Neg_chain_succession([`ϕ`],[`ψ`])`	ϕ and ψ cannot be executed in *sequence*

Business constraints (norms) in the Declare framework are modelled by means of *Declare expressions.* These expressions are grouped into four categories, namely: *existence, relations, choice* and *negative* constraints. Table 5.5 illustrates the Declare constraints and their meanings. The majority of these constraint patterns are used to express obligations, while negative constraints express prohibitions; and correspond to LTL expressions that provide semantics for Declare's graphical notations.

We now examine in detail the Declare's constraints in each group to determine whether they have correspondence with the obligation types.

Existence Constraints: Declare provides four generic patterns, and one special unary constraint pattern that define the number of times an activity might or might not occur in a trace or during the whole execution of the process as illustrated in Table 5.5. Essentially, existence patterns have visual resemblance to UML's multiplicity constraints (Pesic et al., 2007). The existence patterns are categorised into: `Existence(n,`ϕ`)`,`Existence(n,`ϕ`)`, and two absence patterns `Absence(n+1,`ϕ`)`, `Absence(`ϕ`)`. The aim of the existence constraints is to specify that the activity ϕ must be present in the execution of trace, whereas absence constraints specify that activity ϕ must occur in the trace. The special `init(`ϕ`)` cardinality pattern has different meanings, as it specifies that the ϕ must be the first executed activity in the model.
Evaluation: The Declare existence constraints have similar meanings to the COMPAS atomic patterns `isAbsent`,`Exists` and `Bounded Exists`. Hence, these patterns will have the same expressiveness to represent obligations and prohibitions. The only difference is that Declare's patterns explicitly define the cardinality of the constraints; that is, the number of occurrences.
Correspondence: The existence patterns can support *Achievement* obligations and *prohibitions.*

Relation & Negative Constraints: Relation constraints are *binary* constraints that define positive dependency between two activities in a trace. In other words; these constraints impose conditions that—the presence of an activity ϕ is bounded with the presence of another activity ψ; thus, the relation constraints are reactive. Essentially, most of the relation constraints specify the execution order of activities; thus, possibly impose qualitative temporal constraints between activities (Pesic and van der Aalst, 2006). If the temporal ordering is not followed, it results in non-compliance of the constraint. However, `Resp_existence` and `Coexists` are relation constraints that do not specify any temporal ordering, thus leaving the activity to execute freely. For example, if activity ϕ occurs, then the activity ψ must also occur in the trace; however, the temporal ordering does not matter if ϕ occurs first or after of ψ.

The `response`, `precedence`, `succession`, and extended variant of `resp_existence` impose strict temporal ordering between activities. `Alt` and `chain` constraints, on the other hand, impose even tighter conditions on the ordering of the execution between activities, thus making `response`, `precedence` and `succession` even stronger. The relation constraints allow defining different positive relations between the activities, from very loose to very strict temporal ordering relations. Table 5.5 depicts the relation constraints.

In addition to relation constraint, Declare also offers negated variants of relation constraints. Negative constraints aim to prevent the occurrence of activity(ies) between the time bounds defined by some other constraints (Pesic et al., 2007). In other words, a negative constraint means that some activity must be absent (must not occur) until the occurrence of some other activity defining a negative relation between them; that is, activities are incompatible. However, it must be noted that negative relation would not mean logical negation of the constraint; rather, both negative and positive constraint can be interpreted as true during the execution of the process. For example, if `Resp_existence` is true then its negated `Neg_Resp_existence` can be evaluated as true in the trace. Declare framework offers negated variants for all relation constraints as illustrated in Table 5.5. Essentially, negative constraints can be used to specify the constraint prohibiting some actions.

Evaluation: Most of the Declare's relation constraints have similar expressive power to the COMPAS's atomic and composite patterns, because these patterns are based on LTL. In addition, these constraint patterns also have similarities with BPMN–Q patterns, which are based on CTL. Since CTL is the superset of LTL, these patterns in these frameworks share the same limitations when it comes to representing different types of obligations. Thus, relation constraints are suitable only for modelling structural compliance rules imposing temporal ordering constraints. However, negative relation constraints can be used for representing *prohibition*.

Correspondence: The negative constraints can have correspondence with *prohibitions*, but no correspondence with relation constraints as they are ordering constraints only.

From the above evaluation it can be seen that Declare has very limited scope, as it can currently model only achievement obligations and prohibitions. No other norm types can be explicitly represented (see Table 5.10). The representation of achievement obligations is only possible because such obligations define deadline, and the obligation conditions must be true for at least once. Declare models with such constraints defining deadlines can be supported because the constraint will be performed at some future time. However, other obligations types for example, persistence and preemptive obligations cannot be expressed. Accordingly, expressing constraints stipulating maintenance obligations can be problematic in Declare because the obligation conditions must hold in all instances throughout the execution of the process. However, there might be some situations when the applicable maintenance obligation constraints might not be present; in these cases, there will be deadlock in the course of interaction between the tasks. In addition, Declare is not able to identify conflicts among constraints in the model; it does not provide any support to handle violations because of the lack of the declarative nature of the LTL, and the non–deterministic behaviour of the process models. Hence, in case of a violation, the interaction between the tasks in the Declare model will be stopped and no further activity can be performed. Accordingly, it is not possible to express permissions, compensation and perdurant obligations.

Remark 5. *The Declare framework also provides choice (branching) constraints that are concerned with a multiplicity of activities (see Pesic et al., 2007, for more details). Choice constraints are interpreted as disjunctive, and aim to combine reactive behaviour of relation and negative constraints. However, they still have the same limitations as relation constraints; hence, we safely omit them from our evaluation.*

5.3.4 Business Process Modelling Notations–Query Language

BPMN–Q (Business Process Modelling Notation-Query Awad et al., 2008a, 2011) is a query–based automated compliance checking framework capable of answering YES/NO type answers to query questions. The framework can model control–flow, data–flow and conditional flow–related compliance rules using visual patterns. These visual patterns are translated into CTL formulas for checking the structural compliance of a process model. The framework adopts a systematic approach to generating the patterns of compliance rules in the form of query templates. These templates are used to identify the set of process models subject to compliance checking in the process repository. Compliance checking is carried out in several steps. First, BPMN–Q sub–graphs are extracted from the process repository using temporal query templates. The query processor only extracts processes that structurally match the query template. These sub–graphs are then reduced by eliminating irrelevant activities and gateways, and are translated into a Petri net model to generate the state space. Alongside the state space generation, BPMN–Q queries are translated into CTL formulas, which are then fed into a model checker, together with the generated state space. In turn, the model checker yields YES/NO to indicate whether the extracted process models comply with the query templates. The framework uses a visual language BPMN–Q to express various types of rules; for example, control–flow, data–flow, and conditional control–flow patterns based on different occurrence *Scope* property specification patterns, as proposed in (Dwyer et al., 1999). Table 5.6 depicts the *scope patterns*, a special case of occurrence patterns, aim to define the ranges (called regions) over which the pattern must be evaluated as true. With the scope patterns, the requirements pertaining to the existence or absence of certain propositions over the ranges of process executions are defined. The range of the scope is determined by defining the start and end point of the range. Five scope patterns can be derived, namely: *global, before, after, after–until, and between scope.* The *global pattern* defines the range over the whole execution of the processes, whereas *before pattern* specifies the ranges before which the proposition must be true. Meanwhile, *after scope* defines the range from which the proposition must hold true. In contrast, *between scope* specifies the ranges at any portion of the process execution in which the proposition must hold true. Similar to between, the *after–until* pattern specify the range until which the proposition must continue to hold, even if the until part of the range is not executed.

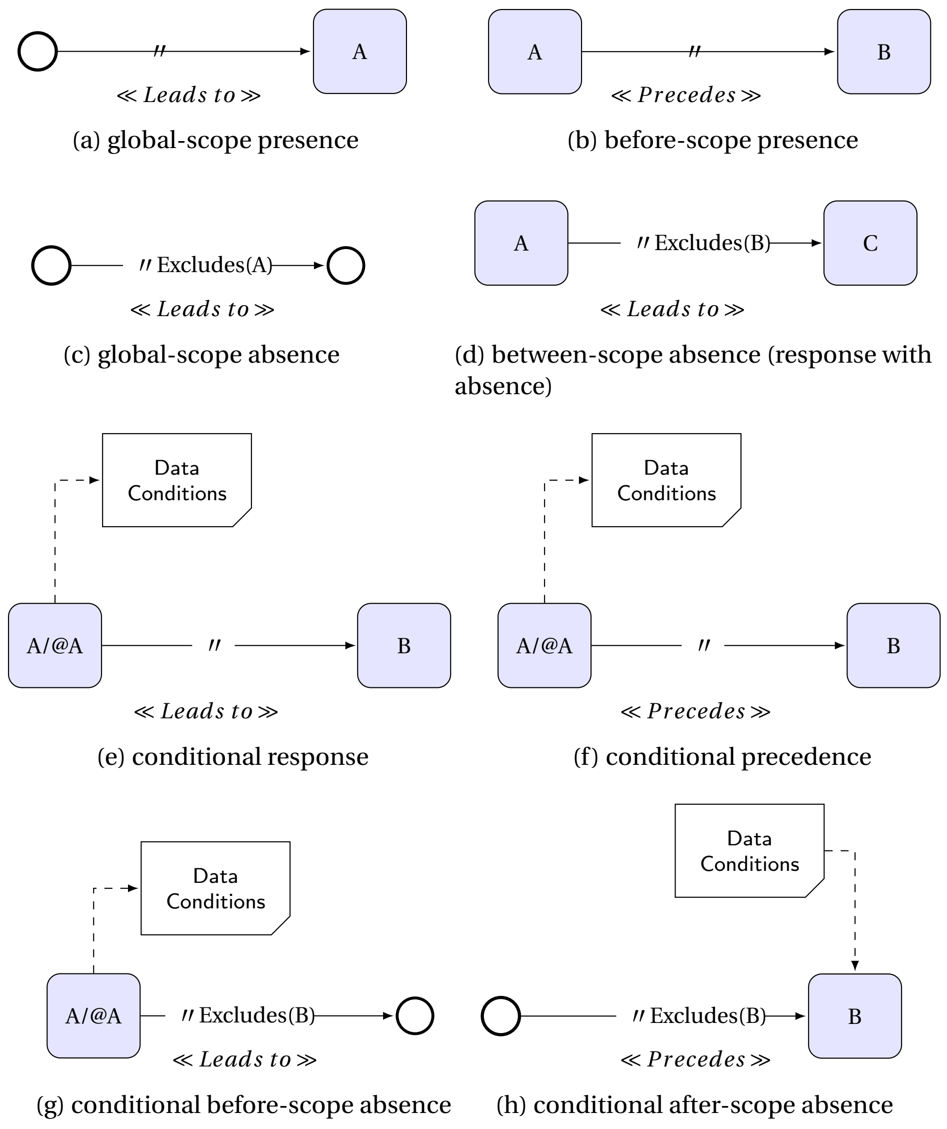
BPMN–Q uses the scope patterns to express different types of obligations by means of visual patterns, as illustrated in Figure 5.2[4]. Each of these patterns, similar to the standard BPMN notations, contains a Computations Tree Logic (CLT) formula giving the specifications of a specific obligation type. For example, the pattern 5.2a shows the global–scope pattern referring to a single activity that might be required to be executed in all process instances. Maintenance obligations can be expressed

[4]This is not an exhaustive list of the BPMN–Q patterns see Awad et al. (2011) for more details.

Table 5.6: BPMN–Q Constraints Name, CTL Representation and Meanings (Awad et al., 2011)

Constraint Name	CLT Representation	Meanings
`Global-scope presence(`ϕ`)`	$\mathsf{AG}(\text{start} \rightarrow \mathsf{AF}(\mathit{executed}(\phi)))$	aims to reflect the requirements of executing a certain activity ϕ in all the instances
`Global-scope absence(`ϕ`)`	$\mathsf{AG}(\text{start} \rightarrow \mathsf{A}[\neg\text{executed}(\phi)\,\mathsf{U}\text{ end })])$	indicates the requirement of not executing an activity ϕ by assuring that there is no state of the process execution, from start to end, where ϕ was executed
`Before-scope presence(`ϕ,ψ`)`	$\neg\mathsf{E}[\neg\text{ready}(\phi)\,\mathsf{U}\text{ ready}(\psi)]$	also known as precedence indicating that the execution of activity ψ is preceded by the execution of another activity ϕ
`Before-scope absence(`ϕ,ψ`)`	$\neg\mathsf{EF}(\text{start} \wedge \mathsf{EF}(\text{executed}(\phi) \wedge \mathsf{EF}(\text{ready}(\psi))))$	An activity ϕ might be required to be absent before the execution of another activity ψ
`After-scope presence(`ϕ,ψ`)`	$\mathsf{AG}(\text{executed}(\phi) \rightarrow \mathsf{AF}(\text{executed}(\psi)))$	also known as response pattern, indicates that after the execution of an activity ϕ, another activity ψ has to be eventually executed
`After-scope absence(`ϕ,ψ`)`	$\mathsf{AG}(\text{executed}(\phi) \rightarrow \mathsf{A}[\neg\text{executed}(\psi)\ \mathsf{U}\text{ end}])$	Similar to before-scope absence, the pattern indicates that certain activities ϕ are prohibited to be executed after the execution of another activity ψ
`Between-scope absence(`ϕ,ψ,φ`)`		indicates that between the execution of two activities ϕ and φ, it is prohibited to execute certain other activities ψ. There are two variations of *Between-scope absence* namely
1. $\mathsf{AG}(\text{executed}(\phi) \rightarrow \mathsf{A}[\neg\text{executed}(\psi)\ \mathsf{U}\text{ executed}(\varphi)])$ [**Response with Absence:**] indicates activity φ is required to be executed after ϕ while there is no chance in between to execute ψ		
2. $\neg\mathsf{E}[\neg\text{executed}(\phi)\,\mathsf{U}\text{ ready}(\varphi)] \wedge \neg\mathsf{EF}(\text{executed}(\phi) \wedge \mathsf{EF}(\text{executed}(\psi) \wedge \text{ ready}(\varphi))))$ [**Response with Absence:**] indicates whenever φ is executed; ϕ must have been executed before without executing ψ in between .		

using *global space presence* patterns (shown above), which enable the execution of specific activity throughout the process. Conversely, the pattern 5.2c illustrates the case where a certain activity must not execute at all. Since prohibitions must be observed in any case during the execution of a process, they are represented by *global space absence* patterns. Accordingly, BPMN–Q also provides constructs to model



(a) global-scope presence

(b) before-scope presence

(c) global-scope absence

(d) between-scope absence (response with absence)

(e) conditional response

(f) conditional precedence

(g) conditional before-scope absence

(h) conditional after-scope absence

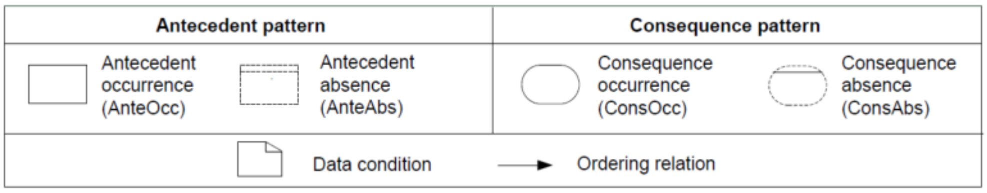
Figure 5.2: List of BPMN–Q visual patterns to model norms Awad et al. (2011)

obligation types where the obligations might prescribe the conditions applicable to the data and resources aspects of a business process. BPMN–Q incorporates most of the patterns used in COMPAS; thus, the framework is able to handle almost the same obligations types as COMPAS (cf. Table 5.10). On the same note, BPMN–Q does not provide any conceptual or formal constructs that can be used to model permissions. Similar to COMPAS, BPMN–Q is also able to handle the violation of obligations. A graphical violation–handling approach using *anti–patterns* is discussed in (Awad and Weske, 2009). The anti–patterns are derived from the BPMN–Q visual patterns, and are comparable to `LeadsTo` and `isAbsent` patterns (see Awad and Weske, 2009, for details on anti–patterns). Finally, BPMN–Q visual patterns are not suitable patterns to model the specifications of the cases of compensations and perdurant obligations.

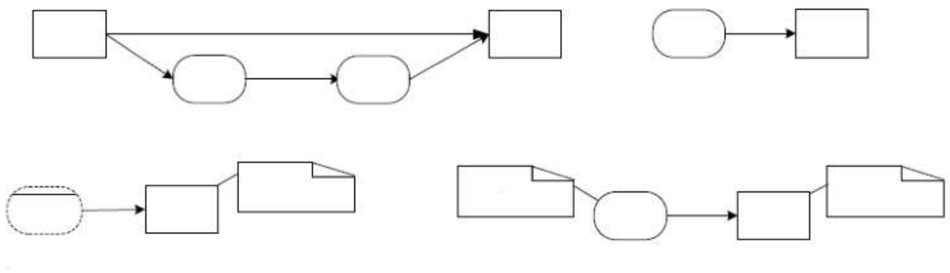
5.3.5 SEAFLOWS

SeaFlows (Ly et al., 2010b, 2012) is compliance verification is a compliance–by–design framework for behaviour and structural compliance verification of blocked–structured process models. It incorporates a graphical language that provides primitives to capture process related complex business rules. These compliance rules are modelled in the form of *first–order–logic* (FOL) predicates equivalents and can be instantiated to the compliance rule graphs (CRG). SeaFlows employs a structural compliance checking strategy for the verification of compliance rules, where node relations are verified against the imposed constraints. The verification is done in three steps: in the first step, a set of structural templates based on the queries on the relations of nodes in the process models is automatically derived. The process model is then checked against the derived templates to detect any non–compliant structural templates. The queried templates are then aggregated and fed into the SeaFlows' compliance module for a further compliance report in the last step. The compliance results are generated on the execution of traces of the process models, where a process model is fully compliant when all the activities in the trace comply with the instantiated rule. A 'No', on the other hand, is returned to indicate rule violations when no activity in the execution trace satisfies the rules.

To model the compliance rules, the SeaFlows framework adopts a compositional graph-based modelling formalism, allowing the modelling of the typical antecedent—consequence patterns for structure of rules as illustrated in Figure 5.3a. These graphs serve as place–holder for the FOL representation of the relevant rules. SeaFlows provides four CRGs, each indicating occurrence and absence of activities of an associated type namely: `ANTEOcc` and `ANTEAbs` which are used to model the *antecedent pattern* triggering a compliance rule. `CONOcc` and `CONAbs`, on the other hand, map the consequence patterns of the rules. In addition, the ordering of the


Antecedent pattern
Consequence pattern
Antecedent occurrence (AnteOcc)
Antecedent absence (AnteAbs)
Consequence occurrence (ConsOcc)
Consequence absence (ConsAbs)
Data condition
Ordering relation

(a) Primitives for Rule Graphs



(b) Compliance Rule Graphs

Figure 5.3: Primitive and Compliance Rule Graphs (Ly et al., 2010b)

nodes in a CRG is defined using a relation primitive, whereas a data conditions primitive is used to represent the data conditions of a compliance rule, as shown in Figure 5.3b. The CRG are not merely visual notations; they are also equipped with the formal semantics for checking verification of process model. This is because SeaFlows is able to check the compliance of behavioural as well as structural compliance rules.

The framework defines five structural patterns as criteria for determining the compliance status of the process. These criteria can be considered as queries on the relations of the nodes of the business process model. The structural criteria patterns are: *containment, occurrence, precedence*, and *precedence relation* as shown in Table 5.7.

Next, we examine these patterns in details to check their correspondence with the obligation types.

Pattern-1: `Contains` (ϕ)

Syntax: $\oplus\ \phi$

Description: The unary structural contains relation whether ϕ is in the process model.

Evaluation: The `Contains` ϕ pattern (also known as *eventually*) is used to indicate

Table 5.7: SeaFlows' Structural Criteria (Ly et al., 2010b)

Structural Criteria	Usage
`Contains` (ϕ)	a unary structural containment relation that indicates if ϕ is contained in the process model
(ϕ) `Excludes` (ψ)	a structural occurrence relation that indicating whether ϕ and ψ are located on different branches of an exclusive gateway
(ϕ) `Implies` (ψ)	a non-directed structural occurrence relation that indicates ψ must not be located on the same branch of an exclusive gateway, on which ψ is located, such that ϕ and ψ both exist
(ϕ) `Implies` ($\psi_1 \vert \psi_2 \vert \ldots \vert \psi_n$)	a non-directed structural occurrence relation indicating that whether `A` is always executed together with $\psi_1, \psi_2, \ldots$ or ψ_n
(ϕ) `Precedes` (ψ)	a structural precedence relation that indicates if there is a directed path in the process model leading from ϕ and ψ

if the proposition ϕ is contained in the portion of the process model. The pattern is similar to the COMPAS and Declare `Existence` pattern, and comparable to CRL's `Bounded Existence` where one may specify the number of occurrences at most (or at least) some bounded number of times. The most common example of `Contains` is specifying termination; for example, on all the executions of the process model eventually, we reach a terminating state (Dwyer et al., 1999). As far as using the `Contains` pattern is concerned `Contains` ϕ can be used to represent achievement obligations because by the persistent obligations definition, the obligation will hold in some future time.
Correspondence: The pattern can be used to represent *achievement* obligations.

Pattern-2: ϕ Excludes ψ
Syntax: $\phi \otimes \psi$
Description: In the process model, the structural relation patterns indicates that whether ϕ and ψ are located on different branches of an exclusive gateway.
Evaluation: The `Excludes` defines the relation between two activities, indicating that ϕ and ψ do not exist in the same trace of the process model. This contra relation pattern is similar to COMPAS's `Exclusive`, and Declare's `Not-CoExistence` pattern that specifies that two activities are incompatible (Montali, 2010). For example, a

rule might put constraint that if one activity (ϕ) occurs, then the other activity (ψ) cannot occur at the same time; for example, if an order is accepted, it cannot be rejected in an interaction with the customer. Although the pattern is useful for representing structural compliance rules, similar to BPMN–Q's `Global-Scope-Absence` visual pattern, it might be useful for representing prohibitions.

Correspondence: The pattern might be useful to model *prohibitions.*

Pattern-3: ϕ Implies ψ

Syntax: $\phi \ \triangleright \ \psi$

Description: The structural ordering pattern defines the relation between two activities ϕ and ψ, prescribing the condition that both activities must not be in the same execution trace; however, they must occur in the execution of the whole process. Essentially, the pattern is suitable for modelling structural rules that stipulate contra relation conditions for two activities. For example, if the order is less than $50,000$, then no solvency check is required if the customer has a premium status. A composite expression (`CoExists`$\wedge$ $\neg$`CoAbsent`), built with COMPAS's atomic patterns, can give the similar meanings that a solvency check cannot co-exist with the customer's premium status on the same branch of the exclusive gateway. As the pattern defines a negative relation between two activities, it can be used to represent prohibition–based deontic norms because prohibitions do not specify temporal properties.

Correspondence: The pattern can be used to model *prohibitions.*

Pattern-4: (ϕ) `Implies` ($\psi_1|\psi_2|\ldots|\psi_n$)

Syntax: $\phi \ \triangleright \ \psi_1|\psi_2|\ldots|\psi_n$

Description: The pattern shows a non-directed structural occurrence relation, indicating whether ϕ is always executed together with $\psi_1,\psi_2,\ldots,\psi_n$.

Evaluation: The `Implies` pattern defines the occurrence relation between an activity ϕ and a set of activities $\psi_1,\psi_2,\ldots,\psi_n$, where the activities are co-located in a set of execution trace. A Declare expression giving similar meanings can be built by combining `resp_existence` and `Coexistence` patterns, as the `Implies` pattern properties do not specify any ordering of the activity execution. However, the pattern can be used only for checking the compliance of structural rules, and cannot

be used for representing the temporal properties of any obligation modality of legal norms.
Correspondence: -NA- (ordering pattern only)

<u>**Pattern-5:**</u> `(`ϕ`) Precedes (`ψ`)`
Syntax: $\phi \gg \psi$
Description: The precedes (precedence relations) indicates whether there is a directed path in the process model leading from ϕ and ψ.
Evaluation: The pattern has the similar objectives to the `Precedes` pattern of COMPAS and Declare, as both frameworks also provide extended variant `Chain-Precedes` to capture the ordering sequence of activities after the execution of the first activity. SeaFlows does not offer the extended `Chain-Precedes` variant; however, the `Chain-Precedes` is also extendible to SealFows. As far as modelling obligation types is concerned, the `Precedes` pattern has the same limitations as COMPAS framework.
Correspondence: -NA- (ordering pattern only)

Essentially, SeaFlows is able to model achievement obligations that stipulate the occurrence of some event in the future by means of the occurrence pattern ($\oplus\ \phi$). The SeaFlows patterns are useful from a structural compliance of business processes perspective. These patterns have no temporal relevance to the semantics of our obligation types because they are based on `ANTEOcc` and `CONSOcc` primitives of the CRGs. The CRGs are based on FOL formulas, and provide the formal semantics of structural compliance rules. However, with the formulas representing the CRGs, it is not possible to give a one–to–one mapping of temporal semantics of other obligation types such as permissions, prohibitions, compensation, and maintenance (see Table 5.10). Such limitation is due to the fact, FOL has no conceptual relevance to the legal domain as it only provides quantifiers; it does not provide temporal operators, which are imperative for the modelling of obligation types (Herrestad, 1991). Hence, the formalism is not suitable for reasoning about the normative requirements.

5.3.6 Process Compliance Language (PCL)

The Process Compliance Language (PCL, Governatori and Rotolo, 2010a) is a formal framework based on defeasible and deontic logic. It provides a conceptually rich formal foundations for modelling norms, and is able to efficiently capture the intuition of almost all types of norms. These norms are modelled in the form of PCL rules for which the framework provides rich semantics.

The state variables and the tasks in the process are represented by a set of propositional literals. PCL formulas, also known as *PCL specifications*, are written based on a set of primitive propositions using $\neg$ negation, $\otimes$ (a non–boolean connective modelling violations chains), and deontic operators representing obligations and permissions. The tasks in business processes are annotated with PCL specifications that are either provided by the domain experts or they are automatically extracted from the schemas of the databases or data sources linked to the processes, using the technique proposed in (Hashmi et al., 2012). These annotations are used to analyse whether the behaviour of an execution path is consistent with the annotated specifications. For this purpose, a three-step algorithm is used in which the process graph is first traversed to find the set of effects for all tasks. These effects are then used to determine the norms in force for the tasks. The effects of the tasks and the pertinent obligations are then compared (in the last step) to find any divergent behaviour. The compliance of the norms is reported as *fully compliant, partially compliant*, or *not compliant* by the algorithm.

Table 5.8: Types of Obligations Operators in PCL (Governatori and Rotolo, 2010a)

Obligation Operators	**Meanings**
O^p	punctual
$O^{a,X}_{pr}$	achievement, persistent, preemptive
$O^{a,X}_{n-pr}$	achievement, persistent, non-preemptive
$O^{a,\tau}_{pr}$	achievement,non-persistent,preemptive
$O^{a,\tau}_{n-pr}$	achievement,non-persistent,non-preemptive
O^m	maintenance

The rich combination of defeasible and deontic logic allows PCL to model all types of obligations (as depicted in Table 5.10), and other aspects of normative reasoning; for example, reasoning with the superiority relation of compliance rules, and reasoning with the contrary–to–duty norms, to name but a few. This is the result of the use of two logics, where the *deontic logic* provides the support for modelling violations of obligations and chains of reparation, while *defeasible logic* (Governatori and Rotolo, 2010b) handles the issue of partial information and inconsistent prescription. To model basic obligations, PCL provides three major constructs: punctual (O^p), maintenance (O^m), and achievement (O^a). Achievement obligations are further refined in perdurant/non–perdurant and preemptive/non–preemptive (Hashmi et al., 2013). PCL allows to explicitly define the deadlines in the formalised PCL expressions, where it allows the algorithm to distinguish between different types of obligations. Table 5.8 illustrates the various obligations operators that PCL provides for representing different types of obligations.

Violations, and obligations arising from the violations, are major concerns in CMFs, and PCL provides effective management of these violations and their compensations. For this purpose, PCL defines a special contrary–to–duty non-boolean $\otimes$ connective that is used to create reparation chains for handling multiple violations of obligations. As far as the persistence of obligation after the violation is concerned, the notion of perdurant obligation has not been addressed in the current version of PCL. However, the notion has been addressed in (Allaire and Governatori, 2014), enabling PCL to offer a holistic and more conceptually rich and sound reasoning support for all types of normative requirements.

5.3.7 Business Process Compliance Auditing Framework

The compliance auditing framework (Ghose and Koliadis, 2007) is a compliance checking framework to verify business process compliance against regulatory requirements.

For this purpose, the analyst first defines a local context description of the accumulated effects. This is because the framework evaluates the compliance locally, at the parts of the process where these effects are applicable. The accumulation process involves the derivation of a set of scenario labels at a point in the process (Hinge et al., 2009). Then, the effects of relevant activities are accumulated over each task, and the annotation of the processes begins. Once the processes are annotated with the context description effects, they are encoded as directed graphs called Semantics Process Networks (SPNs). These networks are used to verify the properties related to the execution ordering of activities, using an algorithm that exhaustively traverses all the execution traces of the effects-annotated processes to check the rule violations. In the last step, the compliance results are reported (in boolean form) to indicate whether a process model satisfies the applicable compliance requirements.

The compliance requirements in this framework attach to process models in the form of parsimonious effect–annotations. There are two types of effect annotations that can be derived from the literature *formal and infomal*. The said framework incorporates formal annotation effects, which are represented and parsed using CTL, a state–based logic. The parsimonious annotations are used to validate the compliant behaviour of the business processes. Unlike PCL's semantic annotations, the formal annotations in this framework cannot make any distinction between different types of obligations, as it is not clear that how such distinction can be made. As the violation of obligation largely depends on the temporal conditions (that is, deadlines), it is not possible to analyse when an obligation is violated; it is only possible to determine whether an activity annotated with the formalised rule description exists or is absent from the graph. In addition, as the framework used a heuristic–based approach for asserting and resolving compliance issues, it uses the structural compliance patterns and semantic patterns. The structural patterns used in the framework are: (a) *Activity/Event/Decision Inclusion*; (b) *Activity/Event/Decision Coordination*; (c) *Activity/Event/Decision Assignment*; (d) *Actor/Resource Inclusion*; and (e) *Actor/Resource Interaction*.

The structural patterns formalised in CTL provide the basis for resolving the non–compliance issues in the processes, albeit in a semi–automated way. Informally, the structural patterns can contain the information on the compliance rule that is violated, and on the actions to repair the problem. In contrast, the semantic patterns might contain suggestions on the required changes to amend the violation in order to restore the compliance issues. The audting framework proposes three semantic patterns namely: (i) *Effect Inclusion*; (ii) *Effect Coordination*; and (iii) *Activity Modification*. Currently, the framework is only able to model achievement obligations while, maintenance, permissions, and perdurant obligations canno tbe represented because the framework does not provide any conceptual or formal constructs to represent such obligation types.

5.4 Discussion

For a CMF to be sound and effective, it needs to be based on sound conceptual and formal models. If the CMF is not based on strong foundations, the compliance checking approach proposed in the CMF cannot be relied upon. We evaluated the conceptual models of seven CMFs based on pre–defined criteria, in particular based on the different classes of normative requirements.

We examined salient features of CMFs, such as their compliance checking approach, the nature of their norms modelling constructs, their underlying formal language, and how they link normative requirements with process models. Table 5.9 summarises these conceptual evaluation results. We also investigated *what is lacking* in terms of technical support in the compliance domain from the perspective of the modelling of normative requirements. The evaluated CMFs incorporate various strategies for checking the compliance of normative requirements, using different graph–based compliance requirements patterns, formalised using variants of temporal logic such as LTL and CTL. In contrast, some CMFs use deontic constructs and norm properties specifications such as PCL and PENELOPE, respectively.

To link the compliance requirements with business process models, the evaluated CMFs use different techniques—for example, COMPAS links formalised compliance rules by means of CRL property patterns, while they are specified in terms of declarative expression in Declare. Accordingly, PCL and SeaFlows annotate the specifications of compliance requirements on business processes. We also examined *the level of compliance management support for all types of normative requirements*. The results highlight that most of the existing CMFs provide only, a *partial support* for all types of normative requirements; in other words, not all types of normative requirements are supported in these CMFs. The exception is PCL, which offers a full modelling support for all types of normative requirements.

In the evaluation, we examined the conceptual foundations of the CMFs to examine which modelling constructs are provided to model different types of obligations; in particular, the constructs that can provide the reasoning and modellign support for obligation types discussed in Chapter 3. The evaluation results are summarised in Table 5.10, which illustrates the available support for a specific type of norms. The '+' symbol indicates that the CMF is able to provide the reasoning and modelling support for a specific obligation modality, and '–' indicates that the obligation modality is not supported (or it is not considered in that CMF).

From Table 5.10, it is evident that only a fraction of normative requirements are supported by the vast majority of the CMFs. For example, PENELOPE is only

Table 5.9: Summary of Conceptual Evaluation of Existing CMFs

CMF/approach language/system	**Compliance checking approach**	**Modelling constructs**	**Underlying formalism**	**Linking norms**	**Level of support**
PENELOPE	Design-Time	Norms property specifications	EC	BPMN Models are generated from deontic assignments	Partial
COMPAS	Run-Time (Structural Compliance Checking)	Domain–pecific patterns	LTL	By means of CRL Property patterns	Partial
DECLARE	Run-Time (Structural Compliance Checking	Control flow–based Declare expressions	Temporal Logic	Constraints specified by means of Declare expressions	Partial
BPMN–Q	Model Checking	BPMN query templates	PLTL / CTL	By means of visual patterns	Partial
PCL/FCL	Design-Time	Deontic constructs specifications	Defeasible and Deontic Logic	Specification of deontic aspects as annotations	Full
SEAFLOW	Design-Time (Structural Compliance Checking)	Compositions graph–based modelling	First order Logic	Annotations	Partial
Auditing Framework	Post Execution	Unspecified	CTL	By means of Parsimonious Effects annotations	Partial

able to support obligations and permissions. It is unable to model other obligation types, and violations because Event–Calculus (EC) is not suitable for reasoning about legal constraints. In contrast, PCL supports all types of obligations because of the non–monotonic characteristics of the formal logic it uses. The combination of defeasible and deontic logic allows PCL to provide reasoning for deontic modalities and violations, especially for temporally varying obligations such as achievement obligations and their persistence over time however, its language is restricted to literals.

DECLARE, BPMN–Q, and COMPAS are LTL based frameworks, and only address 'structural compliance' where the tasks are defined by the constraint models. These frameworks cannot capture the intuition of all types of obligations, violations, and their compensations. DECLARE can only support achievement obligations and prohibitions. BPMN–Q can support achievement and prohibitions, and provide reasoning support for violation handling. More recently, COMPAS framework has been extended with new CRL patterns that enables it to represent maintenance and contrary–to–duty (compensation) obligations (Elgammal et al., 2014). However, COMPAS still has limited scope in representing other obligation types such as permissions and perdurant obligations. In contrast, SeaFlows which is based on FOL, is able to offer modelling support for achievement and prohibitions only.

Table 5.10: Summary of Norms Support in Existing CMFs

	Obligations Types									
Framework	**Achievement**	**Preemptive**	**Non-Preemptive**	**Maintenance**	**Punctual**	**Perdurant**	**Compensation**	**Permissions**	**Prohibitions**	**Violations**
PENELOPE	+	–	–	–	–	–	–	+	–	–
PCL	+	+	+	+	+	+	+	+	+	+
DECLARE	+	–	–	–	–	–	–	–	+	–
BPMN–Q	+	–	–	–	–	–	–	–	+	+
SEAFLOWS	+	–	–	–	–	–	–	–	+	+
COMPAS	+	–	–	+	–	–	+	–	+	+
AUDITING BPC	+	–	–	–	–	–	–	–	–	–

It is generally highly desirable that a formal language for compliance covers most of the properties, and properties of the environment of the unit under verification (for example, normative requirements). In addition, it should also support the complex properties from simpler ones. Temporal logic has limited reasoning capabilities for legal norms because, it has no conceptual relative correspondence to the legal domain; thus, the CMFs grounded on LTL cannot expressively model the properties of the norms. Accordingly, EC and FOL also have their limitations when it comes to providing reasoning and modelling support for all types of obligations.

The conceptual evaluation results portray a somewhat *bleak picture* when it comes to seeing how existing frameworks represent legal knowledge for compliance checking, because none is able to support all types of normative requirements. Primarily this is because of the formal language each framework uses to model the norms. However, this would not necessarily mean that the framework does not have expressive power to model the notion, but that the concept is not considered or analysed in that framework, including the cases where the deontic concepts cannot be faithfully represented. Accordingly, it is possible that each CMF might be designed with different objectives in mind. Regardless of the objectives, each CMF must properly model the legal component of compliance and provide reasoning support for all types of norms. Governatori (2015) provides a fitting example where not paying attention to legal reasoning principles leads to results contrary to those that legally trained professionals would produce. This implies that adopting formalisms that are not conceptually grounded in legal practice creates a framework that is unreliable, and not suitable to be used in real–life applications.

5.5 Related Work

The evaluations presented in this chapter are comparable to several existing evaluations reported in the literature. Becker et al. (2012) offer a literature survey based on the generalisability and applicability of business process compliance frameworks. Their evaluation is based on the reported implementation results for the surveyed frameworks. In evaluating the compliance rules and generalisability of the frameworks, they used a narrow and medium rules generalisability criteria, thus restricting their survey to the checking of simple and complex compliance patterns representing the compliance requirements.

El Kharbili (2012), on the other hand, make a detailed comparative analysis of the functional and non–functional capabilities of Regulatory Compliance Management (RCM) solutions in the domain of business process management (BPM), based on a three categories evaluation criteria. In the first category, they evaluate the RCM solutions from the business users, and the methodology and the architecture of the RCM perspective. Whereas, in the second category, they evaluate nine functional areas of the RCM from a BPM perspective (for example, the strategy model, business process model), and compliance dimensions such as compliance enforce, audit, and verification. In the last category, they use the functional and non–functional capabilities as the evaluation criteria. From the compliance dimensions, they extract three distinct types of rules—*structural, temporal* and *contractual* rules—that are supported by the modelling languages. However, their comparative evaluation does not systematically evaluate "*legal requirements*" from the reasoning and proper modelling of the legal requirements perspective. Also, they do not consider specific types of legal reguirements and how they can be modelled. Cabannilas et al. (2010) study various frameworks using a four–point criteria, including the study of modelling languages that are used to model business processes and rules, with focus on which *modelling languages will be used for the purpose*. Otto and Anton (2007) examine various approaches to regulation modelling languages, and the extraction of key legal concepts from legal documents. Elgammal et al. (2011a), on the other hand, make a detailed comparative analysis of three languages for modelling the business process compliance requirements, with a focus on the *design–time* modelling phase. Their analysis is based on the capabilities and limitations of each selected language chosen from temporal and deontic families of logics, and they list 11 features that a process modelling language should have for the formal specification of compliance requirements. In contrast, Turki and Bjekovic-Obradovic (2010) investigate the practice of regulation analysis and the approaches that aim to achieve and maintain regulatory compliance with given regulations, from an information system and services perspective. Ly et al. (2013) report on an evaluation of five frameworks from various domains using a set of core compliance management functionalities derived from the compliance literature and various case studies. Their work lacks a comparison of the compliance modelling languages and constructs for the specifications of norms. In contrast, Bonatti et al. (2004) study the existing approaches to logic and rule–based systems behaviour specifications from business and security policy rules to identify the possible usage for rule–based policies in a semantic web context.

The presented evaluation is both complementary to these studies, and differs from them. It differs in that we primarily evaluated existing CMFs to examine *what they can do in terms of providing round–up compliance*, and *what constructs* they provide to model different types of normative requirements. The study by Elgammal et al. (2011a) is somewhat close to the above–presented evaluation; however, it is more generic in the sense that the authors examine how the specifications of compliance requirements can be modelled by a specific modelling language, whereas we examined at the specific constructs provided by the CMFs for modelling a specific type of norms. In addition, by using the classification of normative requirements, we also examined whether existing CMFs can provide reasoning support for all types of normative requirements.

5.6 Summary

This chapter contributed a detailed methodological evaluation of seven existing CMFs, using a sound methodology that examined their conceptual foundations under pre–defined evaluation criteria. Specifically, we looked at the conceptual approaches that existing CMFs use to deal with the normative requirements related to regulatory compliance. The presented evaluation is complementary to—and different from existing works as discussed in previous section. This is because we evaluated existing CMFs to check what they can do in terms of providing round–up compliance, and what constructs they offer to model different types of normative requirements. In addition, we also examined whether existing CMFs can offer reasoning support for all types of norms.

The evaluation results portray a somewhat *bleak picture* when it comes to seeing how existing frameworks represent the legal knowledge for compliance checking, as none is able to support all types of norms. Primarily, this is because of the formal language that each framework uses to model the norms—in particular, where the language used in the CMF lacks expressiveness to cover a specific concept. This highlights an exigent need for new compliance rules–modelling languages, with sound theoretical and formal foundations, to effectively model and faithfully represent the legal knowledge thus, and thus increase the effectiveness of CMFs.

In the next Chapter, we examine the formal foundations of some CMFs and report on formal semantics evaluations where we examine the modelling behaviour and constructs provided, and their correspondence to a specific modelling language. We also examine the expressive power of the formal language used in the CMF to identify the strengths and weaknesses of the language.

CHAPTER 6

FORMAL EVALUATION OF CMFs

6.1 Background

From a business process compliance perspective, the main problem is to ensure that the activities to be executed during the execution of the process are in alignment with the specifications of the norms controlling the process behaviour. Formalising the normative specifications of compliance rules using some formalism enables automated verification of normative specifications over business process specifications. Essentially, from a formal representational perspective, most of the existing formalisms are able to represent the specifications of norms and business processes. For example, temporal logic is able to formally represent the specifications of a business process; that is, sequence of states corresponding to the tasks of a business process (Governatori, 2015). The logic is equally able to represent the specifications of the legal constraints. As norms have their lifespan, when considering the temporal aspect, they can be classified according to their temporal validity and the effects they produce when applied.

Earlier (in Chapter 3) we presented a classification of norms and their semantics that provide the new classes of normative requirements. With the new classification of norms the question is: Can existing formalisms faithfully represent different types of norms—such as obligations, permissions, in an expressive and conceptually sound way. The lack of expressiveness or complexity of the chosen formalism can significantly hinder the ability of the compliance checking technique proposed in a CMF to validate the specifications of norms. Addressing this question is of utmost importance before the adoption of a formalism to propose a compliance checking technique in the CMF for real life cases. Unless we have a positive answer to this question, the effectiveness of a CMF based on some formalism for representing and checking the compliance of different types of norms is pointless.

In the previous chapter, we examined the conceptual foundations of the selected CMFs and looked at the constructs these CMFs provide to represent various classes of norms. Each of these CMFs adopts a specific formalism to represent the specifications of norms, for example, Event–Calculus (EC). In this chapter, we formally evaluate the underlying formal languages and the constructs provided by different CMFs based on the formal semantics proposal in Chapter 3. We have chosen to evaluate CMFs based on Linear Temporal Logic (LTL) and Event–Calculus (EC). This is because LTL is a successful formal language and used in many of the existing CMFs, whereas EC is able to capture the time–varying properties of the semantics of our obligations classes. Specifically, we evaluate whether these formal languages are able to provide a faithful representation of different types of norms in a conceptually sound way, and identify potential issues resulting from the modelling of different types of obligations, if any.

The chapter is structured as follows: next the COMPAS framework is briefly discussed (Section 6.2) and followed by a terse introduction to LTL in (Section 6.2.1). We then examine the scenario introduced in Governatori (2015), using it to point out the shortcomings of the use of temporal logic to model norms (Section 6.2.2), and we study how the example affects compliance request language (CRL) and LTL. The PENELOPE framework is then discussed in details (Section 6.3), following which a short introduction to EC (Section 6.3.1) is given. We then model different types of obligations with PENELOPE semantics, using real-life examples, and discuss the identified problems (Section 6.3.2). After that a deontic extension to EC addressing the problems with PENELOPE semantics is discussed (Section 6.4). Then we show how the deontic extension can be used to address the identified problems with EC predicates (Section 6.5). The second last section is dedicated to related work (Section 6.6), and the summary (Section 6.7) highlights the contributions of this chapter.

6.2 COMPAS

The COMPAS (Elgammal et al., 2011a) is a compliance governance framework for service oriented architecture (SOA)-based systems. The framework is grounded on LTL–based graphical patterns for representing different types of obligations in the form of CRL expressions. The CRL expressions are translated into LTL formulas to give the specifications of legal norms for automated verification of compliant behaviour of business processes. The reasons behind using a pattern–based verification approach in COMPAS is to address the usability and comprehensibility problem of understanding of formal languages such as temporal logic. The usability problem is another concern for the non–technical users with less knowledge of the formal languages (see Elgammal et al., 2014, for a detailed discussion of the issues of comprehensibility and usability). This issue lead to the emergence of graphical pattern–based approaches, as used in COMPAS and many other CMFs such as BPMN–Q (Awad et al., 2011) and DECLARE (Pesic et al., 2007). Pattern–based approaches embed the complex logic formulas, translating the compliance requirements into an easy to understand visual patterns. This allows the non–technical users to have better understanding on the state of the affairs in the evolution of the system.

Governatori (2015) argues that temporal logic has been successfully used for the verification of industrial applications, and that it is equally suitable for giving the specifications of business processes. Thus, many researchers adopted LTL as the underlying formal language for their compliance frameworks (see Awad et al., 2011; Elgammal et al., 2014; Pesic et al., 2007); however, it seems that LTL is not appropriate for the modeling of norms and norms compliance. Hence, the debate about whether LTL is suitable for representing legal norms has been a long one. It includes the work of (Thomason, 1981), and who supports it, and (Governatori, 2015) who raises the questions about whether frameworks based on LTL are able to determine whether a business process complies with the set of legal norms. As discussed earlier, COMPAS models compliance requirements by means of graphical patterns, which are categorised into *atomic patterns*, *resources patterns*, and *composite patterns*. These patterns are mapped into LTL formulas, enabling the translation of CRL expressions into a set of LTL formulas. Hence, taking Governatori's argument into consideration, this boils down to question whether the COMPAS patterns based on LTL fully capture the meanings of legal norms; that is: can the results obtained from the COMPAS framework be relied upon to verify the compliant behaviour of business processes?

In the rest of this section, we focus on COMPAS and its underlying formal language the CRL. This is because it includes most of the patterns used by the other frameworks, and it also provides additional patterns meant to represent the features of specific norms such as exceptions to rules, and compensations of violations. However, the analysis can be equally extended to other LTL–based CMFs.

6.2.1 Logic Background: Linear Temporal Logic

Linear Temporal Logic (LTL) (Pnueli, 1977) is a formal logic for specifying the temporal notion of time for the specification and verification of reactive systems. The logic is called LTL because of the qualitative nature of time, which is *path–based* and can be seen as linear. In other words, at a particular moment in time, a state can only have one possible unique future, which can be linearly modelled. Temporal logic is equipped with unary and binary temporal operators. The unary temporal operators are:

- $\mathsf{X}\phi$: **Next** ϕ (ϕ will hold in the next state)
- $\mathsf{F}\phi$: **Eventually** ϕ (ϕ will hold sometime in future)
- $\mathsf{G}\phi$: **Globally** ϕ (ϕ will always hold in future)

Whereas the binary temporal operators are:

- $\phi\ \mathsf{U}\ \psi$: ϕ **until** ψ(ϕ will hold until ψ holds)
- $\phi\ \mathsf{W}\ \psi$: ϕ **weak until** ψ(ϕ will hold until ψ holds and ψ might not hold)

The equivalence of these operators that establish their inter–definability are as follows:

- $\mathsf{F}\phi \overset{def}{=} \top\ \mathsf{U}\ \phi$
- $\mathsf{G}\phi \overset{def}{=} \neg\mathsf{F}\neg\phi$
- $\phi\ \mathsf{W}\ \psi \overset{def}{=} \phi\ \mathsf{U}\ \psi \vee \mathsf{G}\phi$ [1]

Business process compliance aims to verify at which state a set of norms is evaluated as true (or false) during the execution of a business process. The semantics of LTL that gives the specifications of a business process can be provided in terms of *transition*

[1] The W is not a standard LTL temporal operator; however, it can be represented as U temporal operator and has the same expressive power, see Baier and Katoen (2007) for details.

systems. A transition system TS can be modelled as the form:

$$TS = \langle S, s_i, R, v \rangle \tag{6.1}$$

where S is a finite set of states; s_i is *initial* state; $R \subseteq S \times S$ is a transition relation, for which it holds $\forall s \in S, \exists t \in S : (s, t) \in R$; and v is a labeling function associating a set of propositions with each state $v : S \mapsto 2^{Prop}$. $Prop$ is a set of atomic propositions; that is $\{p_1, p_2, \ldots\} \in Prop$.

The formulas in LTL are evaluated against a *trace*. A *trace* is a sequence of *states* in S connected by a relation R representing the transition, and denoted by σ where $\sigma = \{s_0, s_1, s_2, \ldots, s_n\}$ is a trace such that $(s_i, s_{i+1}) \in R$, where $i = 0, 1, 2, \ldots$ is a natural number. Given a trace σ, σ_i represents the subsequence of σ starting with i-th element, and $\sigma[i]$.

Given the above definitions, the satisfaction conditions for the labeling function v for various temporal operators is as follows:

- $TS, \sigma \models p (p \in Prop)$ iff $p \in v(\sigma[0])$;
- $TS, \sigma \models \neg\phi$ iff $TS, \sigma \not\models \phi$
- $TS, \sigma \models \phi \wedge \psi$ iff $TS, \sigma \models \phi$ iff $TS, \sigma \models \psi$
- $TS, \sigma \models \mathsf{X}\phi$ iff $TS, \sigma_1 \models \phi$
- $TS, \sigma \models \mathsf{F}\phi$ iff $\exists k \geq 0, TS, \sigma_k \models \phi$
- $TS, \sigma \models \mathsf{G}\phi$ iff $\forall k \geq 0, TS, \sigma_k \models \phi$
- $TS, \sigma \models \phi \;\mathsf{U}\; \psi$ iff $\exists k : k \geq 0, TS, \sigma_k \models \psi$ and $\forall j : 0 \leq j < k, TS, \sigma_j \models \psi$

The formula ϕ is true in the trace σ if and only if ϕ is true in the first element of the trace. The definition giving the satisfaction of formula ϕ in a state $s_i \in S(TS, s_i \models \phi)$ is:

$$TS, s_i \models \phi \text{ iff } \forall \sigma : \sigma[0] = s_i, TS, \sigma \models \phi \tag{6.2}$$

As we have seen, the semantics of LTL is given by a discrete and totally ordered set of time instants. This structure is isomorphic to a subset of the set of natural numbers, and thus it is isomorphic to a trace of a process.

In Section 3.4, we discussed the function *State* (Definition 1) to populate states resulting from the execution of the tasks in a process trace. Given a process trace t, the correspondence between *State* and the valuation function v can be immediately seen, it is easy to model the conditions for definitions of the various types of obligations in LTL. If we ignore the triggering conditions and deadlines, an achievement obligation (Definition 23) can be modelled using F temporal operator and a maintenance obligation (Definition 24) G operator. The full definition for a maintenance obligation for ϕ can be given by:

$$\mathsf{G}(\tau \rightarrow \phi \,\mathsf{U}\, \delta) \tag{6.3}$$

where τ is a formula corresponding to the condition of activation of the obligation and δ is a formula encoding the deadline for the obligation. Similarly, for an achievement obligation for ϕ, we have

$$\mathsf{G}(\tau \rightarrow \neg(\neg\phi \,\mathsf{U}\, \delta)). \tag{6.4}$$

Given a model encoding the trace of a business process and a set of formulas encoding the relevant norms, compliance then is reduced to the problem of determining whether the formulas can be satisfied by the model.

6.2.2 Motivating Example: Privacy Act

In this section, we strengthen our argument by illustrating the limitations of LTL for modelling legal norms by using a synthetic *Privacy Act* proposed in Governatori (2015) as a real life case.

Suppose that a Privacy Act contains the following norms:[2]

Section 1. The collection of personal information is forbidden, unless acting on a court order authorising it.

Section 2. The destruction of illegally collected personal information before accessing it is a defence against the illegal collection of the personal information.

Section 3. The collection of medical information is forbidden, unless the entity collecting the medical information is permitted to collect personal information.

Moreover, the Act defines and specifies *personal information* and *medical information* as separate entities. In addition, the Act specifies what personal information and medical information are, and they turn out to be disjoint.

Let us assume that an entity, subject to the Act, collects personal information without being permitted to do so; at the same time, it collects medical information.

[2]The Privacy Act presented here, though realistic, is a fictional one. However, (i) it is based on the novel Australian Privacy Principles (APP), Privacy (Enhancing Privacy Protection) Act 2012; and (ii) sections with the same logical structure as the clauses of this fictional act are present in the APP Act.

The entity recognises that it has illegally collected personal information (that is, without being authorised to do so by a court order), and decides to remediate the illegal collection by destroying the information before accessing it. In this case, is the entity compliant with the Privacy Act above? Given that the personal information was destroyed, the entity was excused from the violation of the first section (illegal collection of personal information). However, even if the entity was excused from the illegal collection, it was never entitled (that is, permitted) to collect personal information[3], and consequently, was not permitted to collect medical information; thus, the prohibition of collecting medical information was in force. Accordingly, the collection of medical information violates the norm forbidding such an activity.

6.2.3 Modelling Privacy Act with LTL/CRL

First, in this section, we formally show *how to represent* the Privacy Act (discussed above) in CRL, and then combine this representation with a simple business process model that implements the activity of collecting data. Then, based on the CRL representation, we analyse whether the process complies with the Privacy Act.

To this end, the first step is to extract the conditions of the Privacy Act. Following the analysis in (Governatori, 2015), Section 1 establishes two conditions:

i. Typically, the collection of personal information is forbidden; and
ii. The collection of personal information is permitted, if there is a court order authorizing the collection of that information.

Section 2 can be paraphrased as follows:

iii. The destruction of personal information collected illegally before accessing it excuses the illegal collection.

Similarly, Section 3 prescribes two conditions:

iv. Typically, the collection of medical information is forbidden; and
v. The collection of medical information is permitted provided the collection of personal information is permitted.

Based on the above analysis, if we abstract from the actual contents of the norms, the structure of the act can be represented by the following set of norms (extended form):

[3]If the entity was permitted to collect personal information, then the collection would not be illegal, and it would not have to destroy it.

E1. A is forbidden.

E2. A is permitted, given C (alternatively: if C, then A is permitted).

E3. The violation of A is compensated by B.

E4. D is forbidden.

E5. If A is permitted, so is D.

To compensate a violation, we have to have a violation that the compensation compensates. Moreover, to have a violation, we have to have an obligation or prohibition that the violation violates. Accordingly, it makes sense to combine $E1$ and $E3$ into a single norm, obtaining thus the following set of norms (condensed form):

C1. A is forbidden; its violation is compensated by B.

C2. A is permitted, given C (alternatively: if C, then A is permitted).

C3. D is forbidden.

C4. If A is permitted, so is D.

Based on the above analysis, we can handle the issue of *how to represent the norms as CRL requirements.* In $C1$, if something is forbidden, it should not appear in the process; thus, we can use the *isAbsent* pattern. As for the compensations, the natural choice is to use the *Else*/*ElseNext* pattern (see, Section 5.3.2). $C2$ and $C4$ set (weak) exceptions to the primary norms, $C2$ to the prohibition of the norm in $C1$, and $C3$ to the norm in $C3$.

Accordingly, the first approximation in CRL is as follows:

CRL1. R_1: ($[R_2]A$ *isAbsent*) *Else* B,

CRL2. R_2: C,

CRL3. R_3: $[R_4]D$ *isAbsent*,

CRL4. R_4: A *isPermitted*.

First of all, it is appropriate to point out that a prohibition corresponds to a maintenance obligation, and is represented by *isAbsent*. However, the first problem we have here is that the translation of the *Else*/*ElseNext* pattern cannot be used for maintenance obligations. The translation given in equation (5.2) results in the following LTL formula:

$$\mathsf{G}(\mathsf{F|X}(\mathsf{G}\neg A \wedge \mathsf{F|X}(A \wedge (A \rightarrow \mathsf{F|X}B)))). \tag{6.5}$$

This formula is always false, given the conjunction of $\mathsf{G}\neg A$ and $\mathsf{F|X}A$. The key reason the pattern does not work for maintenance obligations and prohibitions, is that the condition for a maintenance obligation for not succeeding is that the obligation has been violated. In other words, we have the opposite of the obligation (see, Section 4.2.1, Definition 24, and note that in a temporal logic setting, $o \notin Ann(t,k)$ is equivalent to $\neg o \in Ann(t,k)$ or, in LTL parlance, $TS, t_k \models \neg o$). Consequently, as remarked in (Governatori, 2015), a violation of a maintenance obligation is represented in LTL by the formula $\mathsf{G}\phi \wedge \neg\phi$. To obviate this problem, we can use the solution advanced in Governatori (2015), where the compensation of maintenance obligation is semantically defined as:

$$\begin{aligned} TS,\sigma \models \phi \otimes \psi \text{ iff } \forall i \geq 0,\ TS,\sigma_i \models \phi; \text{ or} \\ \exists j,k : 0 \leq j \leq k,\ TS,\sigma_j \models \neg\phi \text{ and } TS,\sigma_k \models \psi. \end{aligned} \tag{6.6}$$

Syntactically, this can be represented by the LTL formula as:

$$\mathsf{G}\phi \vee \mathsf{F}(\neg\phi \wedge \mathsf{F|X}\psi). \tag{6.7}$$

The next problem we have to address is *how to model permissions in CRL*. Given that it is not possible to violate a permission, permissions seem not to play any role in compliance, and CRL does not provide specific patterns for their modelling. It is true that permissions cannot be violated, and thus, they are not needed for the semantics for compliance.

To determine when a process is compliant, one has to know what obligations are in force for the various states traversed by the traces of the process. Accordingly, a domain expert who understands the regulatory requirement can populate the *Force* function (Definition 2), based on their understanding of the legal framework to which the process is subject to. This means that, in that approach, one can dispense with a logical representation of the regulatory requirements. However, this approach rapidly becomes unattainable, given that, even for small to medium size business processes, the number of traces and states in the traces is large, as is the number of obligations and prohibitions (Hashmi et al., 2015a, reports on a real life case study with a medium size process, containing approximately 40 tasks, that would require populating over 25,000 states, with over 100 obligations).

The discussion so far suggests that we need methods to (automatically) determine what obligations are in force, given a set of regulatory requirements. Furthermore, the scenario given in Section 6.2.2 demonstrates that permissions can play a role in compliance: they can be used as conditions that determine when other obligations or prohibitions are in force. For example, consider the compliant process where: (1) the entity checks whether the collection of personal information is authorised under a court order; if so (2) , it proceeds to collect personal information; and (3) it collects medical information. This process would be deemed as not–compliant, since R_4 would resolve in D (collection of medical information), which is absent but it occurs in the process.

As we have seen in Section 6.2.2, in legal theory, a permission is considered as the absence of obligation to the contrary. Thus, in deontic logic, the deontic operator for permission (P) is assumed to be dual of the operator for obligation (OBL); that is:

$$\mathsf{P}\phi \overset{def}{=\!=} \neg\mathrm{OBL}\neg\phi. \tag{6.8}$$

In the case at hand, the obligation is a maintenance obligation and, as we have argued, it corresponds to the *isAbsent* pattern, which is translated as $\mathsf{G}\neg A$, and its dual is $\mathsf{F}A$. Thus, based on this analysis the translation from CRL to LTL gives the following two formulas:

LTL1. $\mathsf{G}(C \vee (\mathsf{G}\neg A \vee \mathsf{F}(A \wedge \mathsf{F}B)))$;

LTL2. $\mathsf{G}(\mathsf{F}A \vee \mathsf{G}\neg D)$.

CRL2 and CRL4 are incorporated in the translations of CRL1 and CRL3; that is, LTL1 and LTL2 respectively.

Consider now the following process (Figure 6.1) for collecting information:

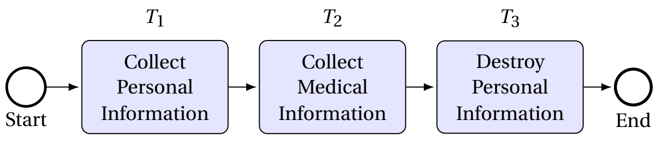

T_1
T_2
T_3
Start
Collect Personal Information
Collect Medical Information
Destroy Personal Information
End

Figure 6.1: Data Collection Process

This process has a single trace, $\langle Start, T_1, T_2, T_3, End\rangle$. The transition system corresponding to this trace has the following transitions:[4]

$$(start, T_1), (T_1, T_2), (T_2, T_3), (T_3, end), (end, end) \tag{6.9}$$

Suppose that for a particular instance of the process, there is no court order authorising the collection of medical data; that is, $\neg C$ holds for all the states reached

[4]The (*end*, *end*) is mandated by the semantics of LTL that requires each state to have a successor; for the state corresponding to the termination of the process, the successor is itself.

by the execution of the process. Thus, the evaluation function associated with the trace is as follows:

- $v(start) = \{\neg A, \neg B, \neg C, \neg D\}$;
- $v(T_1) = \{A, \neg B, \neg C, \neg D\}$;
- $v(T_2) = \{A, \neg B, \neg C, D\}$;
- $v(T_3) = \{A, B, \neg C, D\}$;
- $v(end) = \{A, B, \neg C, D\}$.

It is easy to verify that the transition system corresponding to the trace of the process is a model of the formulas encoding the privacy act, LTL1 and LTL2. This means that the formulas are satisfied in all their states in it. For LTL1, we notice that the first disjunct, $\neg C$ is always false, but the second disjunct is satisfied: every state in the transition system has a state following it where A holds, and a state following it where B holds. For LTL2, the first disjunct is true; for each state, there is a state following it where A holds; thus, $\mathsf{F}A$ holds. Hence, the process is compliant with the LTL formulas encoding the CRL patterns modelling the privacy act. However, there is state T_2, where both $\neg C$ and D hold. In Section 6.2.2, we argued that a situation where $\neg C$ and D both hold is not compliant. Therefore, we have a paradox: the formalisation indicates that the scenario is compliant, and the course of actions described by the transition system does not result in a contradiction, so no illegal action is performed (or better, the collection of personal information is illegal, but its compensation [destruction of the personal information], makes full amends for it); however, our legal intuition suggests that the collection of medical information in the circumstances of the scenario is illegal.

We now evaluate the formal semantics of PENELOPE, a design–time CMF based on EC to examine whether EC too suffers from limitations as LTL to reason about the legal norms.

6.3 PENELOPE

PENELOPE (Goedertier and Vanthienen, 2006c) is a declarative framework that declaratively captures obligations and permissions requirements on the tasks of business processes in the form of deontic assignments. Aiming to provide design-time compliance verification of business processes, PENELOPE proposes deontic properties for modelling the deontic notions of obligations, permissions, and conditional commitments as illustrated in Table 6.1.

Table 6.1: Deontic Properties of PENELOPE (Goedertier and Vanthienen, 2006c)

Term	**Meanings**
Xor(α_1,α_2)	*compound activity* α_1 *XOR* α_2
Or(α_1,α_2)	*compound activity* α_1 *OR* α_2
And(α_1,α_2)	*compound activity* α_1 *AND* α_2
Oblig(π,α,δ)	*agent* π *must do the activity* α *by due date* δ
Perm(π,α,δ)	*agent* π *can do the activity* α *prior to due date* δ
CC$(\pi,\alpha_1,\delta_1,\alpha_2,\delta_2)$	*agent* π *must do activity* α_2 *by due date* δ_2 *after activity* α_1 *is performed prior to due date* δ_1
$(A) Terminates(\alpha, Oblig(\pi,\alpha,\delta),\tau) \longleftarrow \tau \leq \delta$	
$(B) Terminates(\alpha, Perm(\pi,\alpha,\delta),\tau) \longleftarrow \tau \leq \delta$	
$(C) Happens(violation(Oblig(\pi,\alpha,\delta)),\delta) \longleftarrow HoldsAt(Oblig(\pi,\alpha,\delta)) \wedge \sim Happens(\alpha,\delta)$	
$(D) Initiates(\alpha_1, Oblig(\pi,\alpha_2,\delta_2),\tau) \longleftarrow \tau \leq \delta_1 \wedge HoldsAt(CC(\pi,\alpha_1,\delta_1,\alpha_2,\delta_2)),\tau)$	

To generate control–flow and temporally compliant business processes from the rule sets of obligations and permissions, PENELOPE uses a proprietary algorithm (see, Algorithm 1), which progressively operates to generate the state space and control–flow of a business process interaction. The *state space* in the generated process corresponds to a set of obligations and permissions that are in force at a particular state, and these obligations and permissions are modelled with EC (Kowalski and Sergot, 1989).

The interaction between the activities linearly flows from one state to another, and all states are enumerated until no obligation or permission holds at a state, or if there is a violation that cannot be repaired. Once all state spaces are computed, the algorithm draws the BPMN model for an agent of a business interaction. The tasks of the process are drawn whenever an obligation set contains all the obligations to be fulfilled by an agent in the activity. Since the modelling of business interactions of all participating agents in the interaction is allowed in PENELOPE, any violations of obligations by a third partner agent (represented in the generated BPMN model) are drawn as intermediate time-out events. On the other hand, the errors and end events are drawn if there is a violation of an obligation or a permission by an agent in a state. With the designed compliant process models, various types of inconsistencies can be identified; for example, deontic, temporal and trust conflicts. The generated process models with PENELOPE are not meant to execute the process; rather, they are meant

Algorithm 1 PENELOPE (Goedertier and Vanthienen, 2006c)

```
1: PO$(\pi,\delta) = \{\alpha : HoldsAt(Oblig(\pi,\alpha,\delta),\delta)\}$
2: PP$(\pi,\delta) = \{\alpha : HoldsAt(Perm(\pi,\alpha,\delta),\delta)\}$
3: OTP$(\pi,\delta) = \{\alpha : HoldsAt(Oblig(\phi,\alpha,\delta),\delta), recipient(\alpha) = \pi\}$
4: PTP$(\pi,\delta) = \{\alpha : HoldsAt(Perm(\phi,\alpha,\delta),\delta), recipient(\alpha) = \pi\}$
5: OO$(\pi,\delta) = \{\alpha : HoldsAt(Oblig(\phi,\alpha,\delta),\delta), \phi \neq \pi\}$
6: OP$(\pi,\delta) = \{\alpha : HoldsAt(Perm(\phi,\alpha,\delta),\delta), \phi \neq \pi\}$
7: drawControlFlow($\pi,\tau$)
8: if $\neg endState(S(\tau))$ then
9:     $\delta \longleftarrow earliestDueDate(\tau)$
10:    if $\{\alpha : \alpha \in PO(\pi,\delta), atomic(\alpha)\} \neq \emptyset$ then Draw tasks in sequence
11:    if $\{and(\alpha_1,\alpha_2) : and(\alpha_1,\alpha_2) \in PO(\pi,\delta)\} \neq \emptyset$ then Draw tasks in parallel
12:    $\exists xor(\alpha_1,\alpha_2) \in PO(\pi,\delta)$ or $PP(\pi,\delta) \neq \emptyset$ then Draw XOR gateway
13:    $ACs \longleftarrow allCombinations(OO(\pi,\delta) \cup OP(\pi,\delta) \cup PP(\pi,\delta))$
14:    forall $AC \in ACs$
15:    $As \longleftarrow AC \cup PO(\pi,\delta)$
16:    if $\exists\alpha : \alpha \in As, \alpha \in xor(\alpha_1,\alpha_2), xor(\alpha_1,\alpha_2) \in PO(\pi,\delta)$ then Draw task $\alpha$
17:    if $\exists\alpha : \alpha \in As, \alpha \in atomic(\alpha), \alpha \in PP(\pi,\delta)$ then Draw (start event and) task $\alpha$
18:    if $\exists\alpha : \alpha \in As, \alpha \in xor(\alpha_1,\alpha_2), xor(\alpha_1,\alpha_2) \in PP(\pi,\delta)$
19:    then Draw (start event and) task $\alpha$
20:    if $\exists\alpha_1,\alpha_2 : \alpha_1 \in AS, \alpha_2 \in AS, and(\alpha_1,\alpha_2) \in PP(\pi,\delta)$ then Draw (start event
21:    and) tasks $\alpha_1,\alpha_2$ in parallel
22:    if OTP$(\pi,\delta) \cup PTP(\pi,\delta) \neq \emptyset$ then Draw event gateway
23:    if $\exists\alpha : \alpha \in OPT(\pi,\delta), \alpha \in As$ then Draw event start/intermediate event $\alpha$
24:    if $\exists\alpha : \alpha \in OTP(\pi,\delta), \alpha \in As$ then Draw event intermediate time-out event $\alpha$
25:    if $\exists\alpha : \alpha \in PTP(\pi,\delta), \alpha \in As$ then Draw event start/intermediate event $\alpha$
26:    perform activities $As$
27:    drawControlFlow ($\pi,\delta$)
28:    revoke activities $As$
29:    end forall
30: else
31:    $\{v : v \in VTM(\pi,\delta)\} \neq \emptyset$ then Draw error event
32:    $\neg\exists v : v \in VTM(\pi,\delta)$ then Draw end event
33: end if
```

to help process designers to check the impact of control–flow and timing constraints on business process design.

Legal norms in PENELOPE are modelled using EC predicates. EC has a long history of use in the agent–based systems and artificial intelligence and legal reasoning domains. In Section 6.2, we argued that LTL has the ability to efficiently model the specifications of business processes; however, it is severely compromised by major limitations when it comes to modelling legal norms. In the coming sections, we formally show that EC also has problems with its semantics when it comes to capturing the effects of obligations; in particular, when an obligation comes into force and when an obligation is terminated, and with respect to the effects of violations on other types of obligations. As discussed earlier, the first step in verifying the compliant behaviour of business processes is to determine which obligations are in force at the *n*–th instant of time at a particular state of the execution trace of a business process. PENELOPE uses *HoldsAt* and *Initiates* predicates to capture the effects of the obligations which give the semantics of state function; however, these predicates have fundamental deficiencies with respect to properly capturing the effects of the obligations. Hence, again, we need to question whether the PENELOPE's EC–based deontic properties can provide a conceptually sound reasoning support for modelling all types of obligations.

6.3.1 Logic Background: Event-Calculus

Event-Calculus (EC Kowalski and Sergot, 1989) is a well–known event–based formalism for reasoning about *events and change* and *the effects of change* resulting from the occurrence of events over time. EC provides a set of rich axioms for capturing the behaviour of dynamic occurrences of both domain–independent and domain–dependent events; hence, the logic is particularly suitable for modelling the dynamic behaviour of a variety of systems. It is based on the idea of the states that time-varying properties of the world, called *fluent* hold at a particular time–point initiated by some event at an earlier time, and is not terminated by some other event during that time period. Accordingly, a fluent does not hold at some time if it was previously terminated and not resumed during that time (Miller and Shanahan, 1999). In contrast, domain–independent axioms illustrate the situations under which an event initiates and terminates. For the rest of this chapter, we use the predicates and axioms from (Miller and Shanahan, 2002), as illustrated in Table 6.2.

The formalism provides predicates expressing the various types of states of an event occurrence; for example, *Happens* (occurrence of an event at a time point); *Initiates* (an event that triggers a property of a system), *Terminates* (an event terminates the property of the system); and *HoldsAt* (that the property of a system holds at a time point); as well as auxiliary predicates to express premature termination (*Clipped*) and resumption (*Declipped*) of an event at a particular point

Table 6.2: Event-Calculus Predicates and Meanings

Predicates	Meanings
Basic	
$Initiates(X,P,T)$	Event X initiates the variable (fluent) P at time T.
$Terminates(X,P,T)$	Event X terminates the variable (fluent) P at time T
$InitiallyTrue(P)$	The variable (fluent) P is true from the beginning of time.
$InitiallyFalse(P)$	The variable (fluent) P is false from the beginning of time.
$Happens(X,T)$	Event X occurs at time T.
$HoldsAt(P,T)$	The variable (fluent) P holds at time T.
Auxiliary Predicates	
$Clipped(T_1,P,T_2)$	The variable (fluent) P is interrupted sometime between T_1 and T_2.
$Declipped(T_1,P,T_2)$	The variable (fluent) P is resumed/initiated sometime between T_1 and T_2.
Independent Axioms	
$HoldsAt(P,T_2) \longleftarrow$	$HoldsAt(P,T_1) \wedge (T_1 < T_1) \wedge \neg Clipped(T,P,T_1)$
$HoldsAt(P,T_2) \longleftarrow$	$Happens(P,T_1) \wedge Initiates(X,P,T_1) \wedge (T_1 < T_2) \wedge$ $\neg Clipped(T_1,P,T_2)$
$\neg HoldsAt(P,T_2) \longleftarrow$	$Happens(X,T_1) \wedge Terminates(X,P,T_1) \wedge (T_1 < T_2) \wedge$ $\neg Declipped(T_1,P,T_2)$
$\neg HoldsAt(P,T_2) \longleftarrow$	$\neg HoldsAt(P,T_1) \wedge (T_1 < T_2) \wedge Declipped(T_1,P,T_2)$
$Clipped(T_1,P,T_2) \overset{def}{=}$	$\exists X,T : Happens(X,T) \wedge (T_1 \leq T < T_2) \wedge Terminates(X,P,T)$
$Declipped(T_1,P,T_2) \overset{def}{=}$	$\exists X,T : Happens(X,T) \wedge (T_1 \leq T < T_2) \wedge Initiates(X,P,T)$

in time between the interval. The *InitiallyTrue* and *InitiallyFalse* allow the modelling of the system for states where only partial information about the domain is available. In contrast, the domain–independent axioms describe the states when a *variable (fluent)* holds—or does not hold at a particular point in time.

The basic domain–independent axioms are:

$$\begin{aligned} HoldsAt(P,T_2) \longleftarrow\ & Happens(P,T_1) \wedge Initiates(X,P,T_1) \wedge \\ & (T_1 < T_2) \wedge \neg Clipped(T_1,P,T_2) \end{aligned} \tag{A1}$$

The axiom (A1) states that the fluent P continues to hold until an event occurs that terminates it.

$$\begin{aligned} \neg HoldsAt(P,T_2) \longleftarrow\ & Happens(X,T_1) \wedge Terminates(X,P,T_1) \wedge \\ & (T_1 < T_2) \wedge \neg Declipped(T_1,P,T_2) \end{aligned} \tag{A2}$$

(A2), on the other hand, states that the fluent P that has been terminated by the event X, continues to hold until it is resumed (re-*Initiates*) by some other event occurrence. The above axioms can be used to model the non-deterministic behaviour of the system. Hence, EC is used for modelling obligations that can be affected by unpredictable situations.

6.3.2 Modelling Obligations with PENELOPE

The PENELOPE's deontic properties that modelling the deontic notions of obligations, permissions, and conditional commitments are based on the EC predicates and events. Generally, norms have an IF—THEN like structure and produce effects depending on the conditions of the obligation hold (with or) after the occurrence of an event. In Section 3.4, we introduced the *Force* and *State* functions. The *Force* function identifies which obligations are in force at the *n*–th instant of time in a given time–line; the objective of *State* function, on the other hand, is to identify what formulas are to be evaluated as true at the *n*–th time instance in the time–line. The semantics of PENELOPE corresponding to the semantics of *State* function proposed in Chapter 3 are as follows:

$$\mathit{HoldsAt}(X,T) \iff X \in \mathit{State}(T) \tag{A3}$$

$$\mathit{Happens}(X,T) \iff X \in \mathit{State}(T) \tag{A4}$$

whereas the semantics of *Force* function, defining when an obligation is in force in the process, are:

$$\mathit{HoldsAt}(Oblig(X),T) \iff X \in Force(T) \tag{A5}$$

$$\mathit{HoldsAt}(Perm(X),T) \iff \neg X \notin Force(T) \tag{A6}$$

Next, we examine how different types of obligations can be modelled with the PENELOPE's deontic properties and when their effects come into force onto tasks of a process when an event occurs. As argued in Chapter 4, with the formalised rules, we get the types and effects of the obligations, which are then linked to the processes, using some logical model to evaluate their truth value. In PENELOPE, the effects of the obligations are acquired using the *Initiates* predicate to populate the *State* and *Force* functions. For this purpose, the regulations described in Section 7 of the synthetic business contract (see Appendix A) are used.

Example: *Internal Complaints Resolution Regulations:* all the complaints pertaining to this contract, herein shall be dealt with in the following manners:

1. **Internal complaints resolution:** all the complaints pertaining to this contract herein shall be dealt with in the following manners:
   a) Making Complaints: Complaints can be made in person, by phone, or by email.
      i. **Acknowledgment**
         A. Any complaint received in person or by phone shall be immediately acknowledged, or
         B. Within 2 working days where received by email or letter.
   b) All received complaints shall be resolved within 7 working days.

The clauses of Section 7 prescribe two different types of obligations, namely: (a) *punctual* obligation, and (b) an *achievement* obligation, to acknowledge oral and written complaints. Now, consider the process fragment in Figure 6.2 that describes the complaint–handling process where a received complaint must be acknowledged before its resolution, and is subject to above regulations.

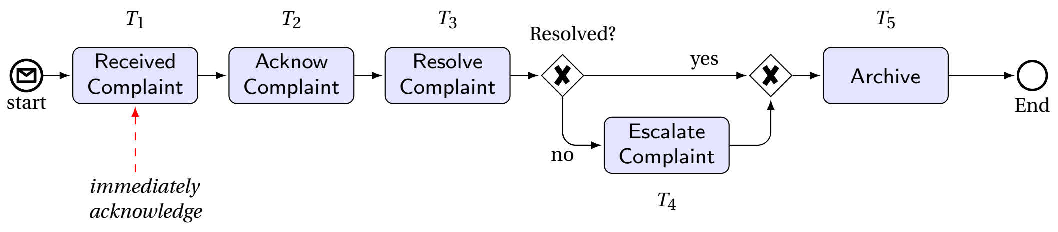


Figure 6.2: Complaint–Handling Process

We now formalise the above rules and examine when the obligation holds in the process trace. The PENELOPE's *Initiates* predicate, giving the conditions of *HoldsAt* from axiom (A1) is as follows:

$$HoldsAt(P, T_2) \longleftarrow Happens(P, T_1) \land Initiates(X, P, T_1) \land \\ (T_1 < T_2) \land \neg Clipped(T_1, P, T_2)$$

The clause 7.1.a.i.A, which prescribes a *punctual* obligation, is represented as:

$$HoldsAt(Acknow, 2) \leftarrow \\ Happens(Complaint, 1) \land Initiates(Complaint, Acknow, 1) \land \\ (1 < 2) \land Clipped(1, Acknow, 2) \tag{A7}$$

The meanings of axiom (A7) is that the *punctual* obligation to *Acknowledge* the received complaint starts to hold from time instant 2 when the event $complaint$ triggering the obligation occurs at time 1. From axiom (A7), we have

$$\mathit{Happens}(Complaint, 1) \wedge \mathit{Initiates}(Complaint, Acknow, 1)$$

which means that we have a complaint at $\mathit{State}(1)$. Also, we get $\mathit{HoldsAt}(Acknow, 2)$, meaning that an obligation starts to hold from the next time instant; that is, $\mathit{State}(2)$. Hence, based on above, we have the following situation:

$$\mathit{Force}(2) = \{Acknowledge\}$$
$$\mathit{State}(1) = \{Complaint\}$$

However, there is a problem with this representation because, for a *punctual* obligation, by Definition 3, the obligation fluent starts to hold as soon as the event triggering the obligation occurs; that is, at $\mathit{State}(1)$. Thus, ideally, we should have representation such as:

$$\begin{aligned}&\mathit{HoldsAt}(Oblig(Acknow), 1) \leftarrow\\&\quad \mathit{Happens}(Complaint, 1) \wedge \mathit{Initiates}(Complaint, Acknow, 1)\end{aligned}$$

that corresponds to:

$$\mathit{Force}(1) = \{Acknowledge\}$$
$$\mathit{State}(1) = \{Complaint\}$$

The representation in (A7) does not reflect this situation because the obligation fluent starts to hold from $\mathit{State}(2)$, not from $\mathit{State}(1)$. Thus, the obligation is not in the *Force* function because we do not have the effects of the obligation on task T_1 and, consequently, it cannot be checked for compliance. This is because PENELOPE's *Initiates* predicate cannot capture the effects of deontic constraints that enter into force at the time of event occurrence, rather than from the next instant.

Next, we examine the cases of violations handling and the obligations arising from the violation of a primary obligation. Reporting and handling the violations of rules is one of the major requirements for a compliance management framework (Awad, 2010). Timely reporting of the violations allows the analysts to address the problems at the very beginning of the process design, thus saving a lot of efforts and time. Now, we consider whether PENELOPE handles various violation situations whether the notion of violations can be effectively handled with PENELOPE's violation semantics, and whether modelling the compensatory cases (contrary–to–duty obligations) is possible.

Consider the terms of payment process model in Figure 6.3. This illustrates that when an invoice is received, payment must be made within the 15 days; otherwise, a penalty of 3% interest is applicable. In the case where the obligation is not fulfilled and the interest is not paid within next 7 days, another 2.5% is admissible. If the obligation conditions are violated again, the contract can be terminated without any further notice. The process is subject to the *terms of payment* regulations in Section

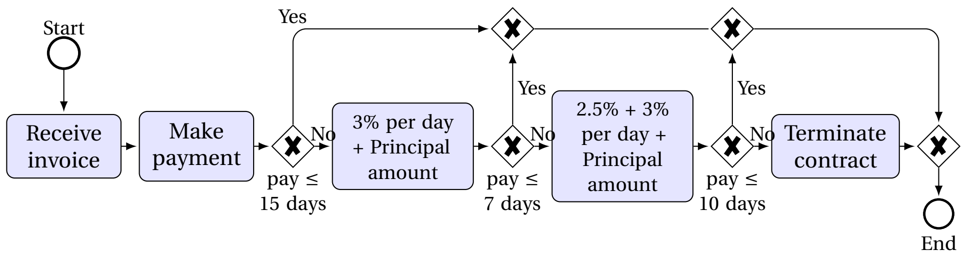


Figure 6.3: Terms of payment process fragment

5 of the business contract in Appendix A. The following conditions[5] can be extracted from the regulations:

C1. *Contractor* must issue the invoice to claim any payments;

C2. *Principal* receives the issued invoice;

C3. *Principal* must pay the received invoice (in full) within 15 days;

C4. If invoice deadline is violated, a 3% (per day) interest compensates it and must be paid within 7 days.

C5. If the defaulted invoice is violated, another 2.5% (per day) interest compensating the violation must be paid within next 10 days.

The conditions of C3., where the *Contractor* has the obligation to pay the invoice within 15 days, is modelled as:

$$\begin{aligned} &\mathit{Initiates}(ReceiveInvoice, \mathbf{OBLIG}(Principal, PayInvoice, \delta), \tau) \longleftarrow \\ &\quad \mathit{Happens}(ReceiveInvoice, \tau) \wedge \mathit{Happens}(IssueInvoice, \tau') \\ &\quad \mathit{HoldsAt}(PaymentClaims, \tau') \wedge \tau' < \tau \wedge \delta = \tau + 15days \end{aligned} \tag{A8}$$

Axiom (A8) gives full instantiation of the interaction of the event, from where we get $\mathit{Initiates}(ReceiveInvoice, \mathbf{OBLIG}(Principal, PayInvoice, \delta), \tau)$; in other words,

[5]Here, we purposefully use the parts of the rules that are relevant to the violation cases of the terms of payment conditions only.

an obligation to pay the invoice enters into force at time τ that must be paid by the deadline δ. We also get *Happens*$(IssueInvoice, \tau')$, which means that *Contractor* issued it at an earlier time instant at τ'. Now, assume that the obligation to pay the invoice is violated. This can be modelled using PENELOPE's violation semantics (given in Table 6.1) as follows:

$$\begin{aligned}&\mathit{Happens}(violation(\mathbf{OBLIG}(\pi,\alpha,\delta),\delta) \longleftarrow \\ &\quad \mathit{HoldsAt}(\mathbf{OBLIG}(\pi,\alpha,\delta),\delta) \wedge \sim \mathit{Happens}(\alpha,\delta)\end{aligned} \tag{A9}$$

The meanings of axiom (A9) is that the obligation to do α is violated if the obligation fluent α that holds at time δ, does not happen at δ. In other words, if the obligation is not fulfilled by the deadline, a violation of the obligation is triggered.

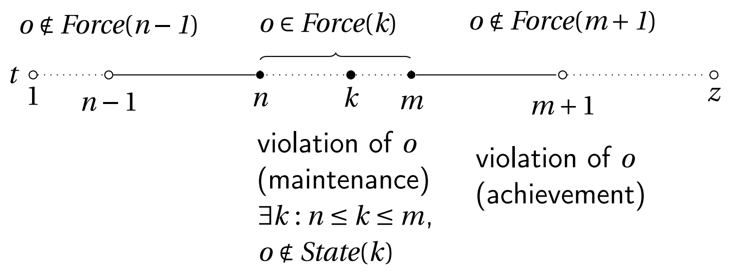
Let us now model the violation conditions of C3. with the above violation semantics, assuming the obligation in axiom A8 is violated:

$$\begin{aligned}&\mathit{Happens}(violation(\mathbf{OBLIG}(Principal, PayInvoice,\delta),\delta) \longleftarrow \\ &\quad \mathit{HoldsAt}(\mathbf{OBLIG}(Principal, PayInvoice,\delta)),\delta) \wedge \sim \mathit{Happens}(PayInvoice,\delta)\end{aligned} \tag{A10}$$

From axiom (A10), we have *HoldsAt*$(\mathbf{OBLIG}(Principal, PayInvoice,\delta),\delta)$. This means that the obligation to pay the invoice by the deadline (that is, 15 days from the issue date) is in force; and if not paid in time, we have the violation; for this, from A10, we derive *Happens*$(violation(\mathbf{OBLIG}(Principal, PayInvoice,\delta)),\delta)$; that is, obligation is violated at deadline if the obligation fluent $PayInvovice$ does not happen at state δ; that is, $\sim$ *Happens*$(PayInvoice,\delta)$.

Here we have an issue with the temporal properties of the above violation semantics because the violation has been evaluated at the deadline, which is not feasible for detecting violations for different cases of norms. In the legal domain, for norms, the violation can only be evaluated once the prescribed deadline has passed. Axiom (A9) evaluates the violation at state δ, and the violation is triggered at δ. On the contrary, in real life situations, the violation of an obligation is evaluated only after the deadline has passed. Consider, for example, the violation semantics of persistent obligations (Definitions 4-6) depicted in Figure 6.4, where the obligation o holds between n and m. The obligation fluent can hold until the last moment and could be fulfilled by the deadline; that is, m. A violation can be triggered only if the obligation contents are not achieved after the deadline has passed; that is, $m+1$.

Generally, the violation conditions for obligations are provided by the analysts; here, however, the objection to PENELOPE's semantics properties is that they detect



$o \notin Force(n-1)$
$o \in Force(k)$
$o \notin Force(m+1)$
t
1
$n-1$
n
k
m
$m+1$
z
violation of o (maintenance) $\exists k : n \leq k \leq m,\ o \notin State(k)$
violation of o (achievement)

Figure 6.4: Violation semantics of Persistent Obligations

the violations at state δ; this is unrealistic for many real life obligation types. For example, for a maintenance obligation, a violation might occur even before the deadline, because such obligations are enforced in intervals and must hold for all time instants between the interval for the whole period of the validity of the obligation. However, the violation axiom evaluating the violation at deadline, indicates that PENELONE is not able to handle maintenance obligations.

Another issue with the violation semantics that are built upon the notion of violations without reparation (VWR) is as follows:

$$\begin{aligned} VWR(\tau) : \{\alpha : &\mathit{Happens}(violation(Oblig(\pi,\alpha,\delta),\tau) \wedge \\ &\neg\exists \mathit{Initiates}(violation(Oblig(\pi,\alpha,\delta),o,\tau)\} \end{aligned} \tag{A11}$$

The meanings of axiom (A11) is that after the violation of an obligation, no other obligation is initiated. This would mean that after every violation, the penalty can be directly imposed. Essentially, the notion of violation without reparation in PENELOPE is closely related to the Arend's *reduction view of norms* (Arend, 2001); that whenever there is a violation, a direct penalty is imposed. In the legal domain, however, there are several cases where a violation penalty is not always imposed, but a sub–ideal situation can still make the process compliant. In other words, fulfilling a contrary–to–duty obligation after the violation can make amends for the violation.

The violation conditions of an obligation generally include the conditions whether the obligation is compensable after violation, and whether it can be further violated and compensated for. Such compensatory conditions are provided by the analysts at the time of modelling the norms. The rule conditions in C4. is one such case where a sub–ideal situation is sought if the primary obligation is not realised;

this, fulfillment of a contrary–to–duty obligation can be represented as:

$$\begin{aligned}&\mathit{Initiates}(violation(\mathbf{OBLIG}(Principal, PayInvoice, \delta')),\\ &\quad \mathbf{OBLIG}(Principal, PayInvoice + 3\%Interest, \delta), \tau) \wedge \delta' < \tau \leq \delta\end{aligned} \tag{A12}$$

Axiom (A12) illustrates that the obligation to PayInvoice and 3%Interest comes into force at *state* τ, after the primary obligation has been violated at a previous state δ' (that is, deadline). Notice that, EC does have expressiveness to represent the notion of compensation, which can be trivially modelled, as shown in axiom (A12); however, PENELOPE does not admit *compensatory obligations*. However, the same problems of getting the effects of the obligations on the tasks (as discussed earlier) remain. Furthermore, as the with the violation semantics, the violation is triggered at the deadline even if the obligation is still in force. Thus, we cannot have a faithful representation of that actual situation.

6.4 Deontic Extension to Event-Calculus

In the previous sections, we have formally shown that both LTL and EC have shortcomings in providing a conceptually sound reasoning and modelling support for different types of obligations. Thus, the CMFs grounded on these formalisms are inherently unreliable for the verification of business processes against a set of legal norms. The effectiveness of a CMF might only be guaranteed if the problems with these formalisms are efficiently addressed; that is, if they are fully able to fully provide both the specifications of the processes models and that of the legal norms without introducing any complexity and/or compromising the efficiency of the formalism.

In this section, we introduce a deontic extension to EC to show how the problems with a formal language can be addressed to increase its ability to provide full reasoning support for all types of legal norms in a conceptually sound way. Hence, the results of compliance verification of business processes from a CMF would be more reliable. We extend EC calculus with new predicates and events to address the issues with the base predicates *Initiates* and *Happens*, as raised earlier. We extend EC because the formalism has relatively flexible semantics when compared to LTL, which is based on standard possible worlds semantics. Governatori (2015) argues that the problems identified by the scenario presented in Section 6.2.2 depends on how permissions are evaluated in standard possible worlds semantics. Accordingly, the issue of how to model norms in a conceptually sound way is left as an open problem. We believe that, by the virtue of the flexibility of the EC semantics, problems with these semantics can be easily resolved without introducing any complexities compromising its efficiency.

Predominantly, the identified problems with the PENELOPEs arise from getting the effects of obligations on the tasks of a business process, and representing violations, and their compensations. This in turn, are the result of inherited problems with the EC's base predicates: *Initiates*, *Happens*, and *Terminates*. This raises questions about the effectiveness of the PENELOPE framework when it comes to representing different types of obligations for their compliance checking.

Next, we discuss in details the issues related to the EC predicates affecting the expressiveness of the PENELOPE's semantics, and introduce new predicates to extend the EC (Hashmi et al., 2014).

6.4.1 Issues with Event-Calculus

The classical EC (Kowalski and Sergot, 1989) is a widely used formalism for modelling norms because it provides a logical framework for representing and modelling the effects of events and the current state of affairs, in terms of fluent(s). It also has the ability to model the time when fluents come into existence and cease to hold dynamically (Goedertier and Vanthienen, 2006a).

One might argue that the modelling of deontic notions with EC is rather well developed, as several variants of EC exist (see Miller and Shanahan, 2002; Sadri and Kowalski, 1995, for further listings of EC variants), and historically, many studies have used EC for reasoning and representing the legal knowledge. For example, Fornara and Colombetti (2009) provide formal specifications of commitments and pre–commitments, and institutionalised power and context using EC, whereas Bandara et al. (2003) use EC for translating the policies and systems behaviour specifications into formal specifications. Meanwhile, Alrawagfeh (2013) represents norms that enable agents to use these norms for their practical reasoning, (see Hashmi et al., 2014, for more approaches).

Another argument about the suitability of EC for representing norms, is its use in modelling the dynamic behaviour of non–deterministic systems, especially where the norms can be affected by unpredictable situations. However, in the previous section, we have formally shown that, by vitrue of EC semantics, PENELOPE has major issues with reasoning about the legal norms. One of the such issues is related to the base predicate of EC $Initiates(E, X, T)$. Its meaning is that event E at time T initiates the fluent X, and the fluent holds at the next instant of time. This effectively means that the norm comes into force at the next instant; however, for legal norms, this might not be the case. There are cases where the norm enters into force at the same instant as the triggering event happens; for example, the obligation to remove shoes when one enters in a mosque.

Figure 6.5 illustrates the case where the obligation fluent X comes into force; that is, X starts to hold when the event E_1 is initiated at time 10.

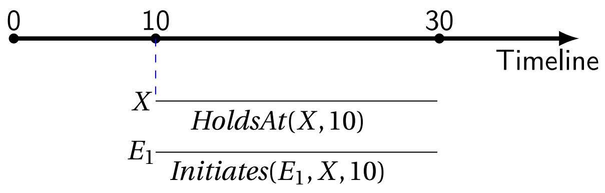


Figure 6.5: Domain–dependent axiom (Simultaneous effects)

Accordingly, in some cases, norms might not take immediate legal effects when an event is triggered; rather, they enter into force after a delay. In other words, the triggering of an event does not necessarily mean the *actual initiation* of the legal norm; that is, the fluent is not in its argument. For example, a complaint cannot be acknowledged until all details pertaining to the issue have been received.[6] Figure 6.6 shows the case where obligation fluent X, that is, acknowledge a complaint, comes into force not at the time when the event E_1 is triggered at time 10; rather, X starts to hold from the next instant; that is, at time 11 .

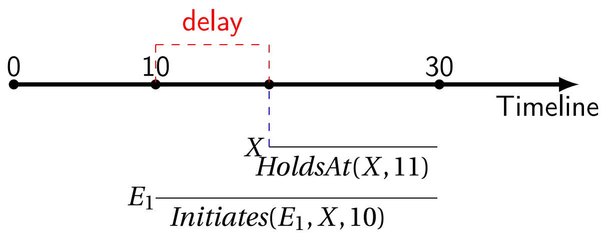


Figure 6.6: Domain–dependent axiom (delayed effects)

[6]It is possible to have a norm that comes into force retroactively; for example, *preemptive obligations*. Thus, the fluent holds before the event that initiates it. We blatantly ignore this aspect throughout this chapter.

Another problem with the predicate *Initiates* is that it does not guarantee that the fluent in its arguments is actually *initiated* by the event. Suppose that: the domain-dependent axioms specify that both the events E_1 and E_2 individually initiate the fluent X; and event E_1 happens at time 10 and event E_2 at time 20; X does not hold initially; and no other event initiates or terminates fluent X between 0 and 30. This means that X starts to hold from 11 and continues to hold up to 30, and event E_2 is irrelevant to determine the status of X. Figure 6.7 illustrates the case where the same fluent X is initiated at different times by two different events—E_1 and E_2 respectively.

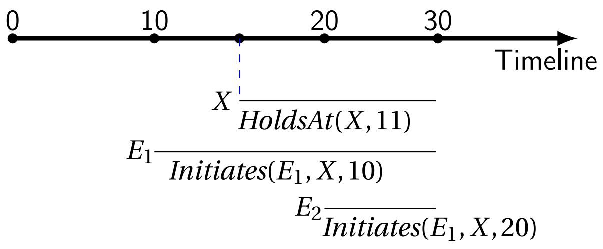


Figure 6.7: Domain dependent axiom(no fluent in the argument)

6.4.2 Extending Event-Calculus

In this section, we discuss the deontic extension to EC by adding the deontic aspect to EC, and introduce new predicates and events to alleviate the above–discussed problems with the EC's base predicates, *Initiates* and *HoldsAt*. The new predicates and events not only allow capturing the legal effects when they enter into force, but also faithfully model all types of obligations and conditions associated with them. In what follows, we discuss in details the newly added predicates and events (illustrated in Table 6.3).

6.4.2.1 *DHoldsAt* Predicate

As the standard EC *Initiates* and *HoldsAt* predicates are not able to capture the legal effects of the obligations on the tasks of a process, we introduce a new *deontically holds at* predicate *DHoldsAt*(X, T), meaning that the *deontic fluent* that is, a specific type of obligation X holds at time T.

Similar to *HoldsAt* predicate, the $DHoldsAt$ predicate requires the same number of arguments, however the main difference is on the conditions of *initiation*—that is:

Table 6.3: Deontic EC Predicates and Meanings

Predicates	Meanings
$DHoldsAt(X,T)$	The variable (fluent) X deontically holds at time T.
$Happens(trigger(O^{x,T}X,N),T)^{\dagger}$	The event $trigger(O^{x,T}X,N)$ with delay N occurs at T.
$DTerminates(trigger(O^{x,T}X,N),T)$	The event $trigger(O^{x,T}X,N)$ with delay N deontically terminates the variable (fluent) X at time T.
$compensates(O^{y,T_{S_c}}Q, O^{x,T}X)$	The variable (fluent) Q at time T_{S_c} compensates another variable (fluent) X.
Events	
$Happens(violation(O^{x,T}X),T_v)$	The violation of variable (fluent) X occurs at time T_v.
$Happens(deadline(O^{x,T}X),T_d)$	The deadline to fulfil the variable (fluent) X occurs at time T_d.
$Happens(compensation(O^{x,T}X),T_{S_c})$	The compensation event for variable (fluent) X occurs at time T_{S_c}.
Boolean Switch	
$FulfillTerminable(O^{x,T}X)$	The variable (fluent) X is terminable upon fulfilment.
$ViolationTerminable(O^{x,T}X)$	The variable (fluent) X is terminable at violation.
$RecursivelyCompensable(O^{x,T}X)$	The variable (fluent) X is recursively compensable for violation.

† $O^{x,T}$ *represents the obligation type and time when the variable(fluent) comes into force.*

- each obligation (or a deontic fluent) has its own specific triggering events, and only one of the triggers can initiate the deontic fluent,
- a trigger does not *initiate* the deontic fluent, if the deontic fluent already holds,
- there could be delay (which could be a null value) between the time the triggering event occurs and the time when the obligation enters into force. (see, Section 6.4.1).

Notice that, an obligation can deontically hold only if it is deontically initiated, which is not possible with the current *HoldsAt* predicate. Accordingly, with the current EC *Initiates* predicate as pointed earlier, it is not possible to deontically initiate an obligation because the *Initiates* predicate cannot capture the delay. To obviate this problem we introduce a special *triggering–event* predicate $trigger(O^{x,T}X,N)$, where $O^{x,T}X$ is a deontic fluent, and N is the delay. $O^{x,T}$ represents the type of obligation (cf. Chapter 3) and the time when the obligation comes into force T, X is the variable attached to the obligation representing the contents of the obligation—which can be either an event or a fluent. Notice that, $O^{x,T}$ in the triggering predicate, has only one time stamp because one can be certain that an obligation holds after it is deontically initiated, but one cannot be certain when it is going to be terminated. As was mentioned above, the aim of the triggering event is to *Initiate* the obligation, thus for getting the effects of obligation on the task of a process we embed the triggering event into the *Happens* predicate that is, *Happens*$(trigger(X,N),T)$. This replaces the *Initiates* predicate to deontically initiate an obligation. However, for a trigger to be effective, one has to specify the conditions defining the trigger for an obligation. Also, the delay must be specified because the delay N determines the difference in time when the triggering event happens and when the obligation enters into force.

6.4.2.2 *DTerminates* Predicates and Events

To handle the termination of deontic fluent associated with the obligations, we introduce a new predicate—*deontically–terminates* predicate. The *DTerminates*(E,X,N,T_{Ter}) means that an event E deontically terminates[7] the deontic fluent X, with some delay N at time T_{Ter}. The delay N defines the time distance from when the terminating event occurs to the actual termination of the deontic fluent happens. Essentially, the deontic termination of an obligation means that it has no legal effects on the execution of the process from the time it is terminated.

Also, for specifying the deadlines of the obligations, in the same way, we define a special *deadline–triggering* event *deadline*$(O^{x,T}X, T_d)$, where $O^{x,T}$ and X are the arguments for the deadline event and serve as the triggering event, and T_d represents the time of deadline event occurrence. The purpose of the deadline event is to signal the time (deadline) until which the obligation conditions must be fulfilled; otherwise, a violation of the obligation is triggered otherwise.

Accordingly, in many cases, not all the obligation conditions are fulfilled, and, these unfulfilled obligation conditions lead to the violations. Hence, for specifying

[7] A deontic fluent can only be deontically terminated if it was deontically initiated.

the violations of the obligations, a *violation–triggering* event $violation(O^{x,T}X, T_v)$ is introduced, where $O^{x,T}$ and X in the arguments of the violation event is obligation with type and X is obligation fluent respectively, while T_v represents the time of violation event occurrence. The violation event signals the time when the obligation conditions are violated.

6.4.2.3 *Terminability* Predicates

In many cases, once an obligation has been fulfilled it is no longer required; thus, the obligation is terminated. However, in some cases, the contents (full or part) of an obligation might still be required to complete other obligations. To handle such cases, to determine whether a deontic fluent will still be required after the fulfilment, we introduced *FulfillTerminable* a boolean switch predicate $FulfillTerminable(O^{x,T}X)$, where $O^{x,T}$ is the obligation with type, the time when obligation comes into force T, and X is the variable attached to the obligation. The aim of this boolean switch is to signal whether a fluent is still required for the fulfilment. The conditions for the switch are provided by the analysts as either YES or NO. With the Yes condition, the obligation is terminated; for NO, the obligation remain in the set of active obligations, even they are fulfilled.

Accordingly, in the similar way, we introduced the *ViolationTerminable* predicate $ViolationTerminable(O^{x,T}X)$ to signal that an obligation is terminable after it has been violated. The predicate takes the same arguments as that of the *FulfillTerminable* and operates in a similar manner as it does for the cases of violations[8].

6.4.2.4 *Compensability* Predicates

As discussed earlier (in Section 3.3), that violated obligations may be compensable. Thus, to handle the compensation of violations we introduced a special event *compensation* predicate $Compensation(O^{x,T}X, T_{s_c})$ where $O^{x,T}$ and X are the arguments for the *Compensation* event serving as trigger for compensation, and T_{s_c} is the time of the compensation event occurrence. The aim of the compensatory event is to compensate the violated obligation. Also, we introduce a binary predicate

[8]Note that if the *ViolationTerminable* is evaluated as *Ture*, it would not necessarily mean that the violated obligation could not be compensated for. A violated obligation can still be compensated for depending upon the conditions of the violated obligation.

Compensates$(O^{y,T_{sc}}Q, O^{x,T}X)$, where the two arguments are two deontic fluents. The meaning of *compensates* is that fulfilling the first deontic fluent makes amends for the violation of the second deontic fluent, and implements the *Comp* function introduced in Section 3.4, Definition 7.

As compensation obligations are obligations themselves, they can be further violated. Accordingly, a compensation obligation can be further compensable upon violation. Based on the violation conditions of the compensation obligation, it can be recursively compensable; thus, we introduced a special *recursive compensation* predicate *RecursivelyCompensable*$(O^{x,T}X)$. This is a boolean switch meant to capture the intuition given by the condition 2 of Definition 9.

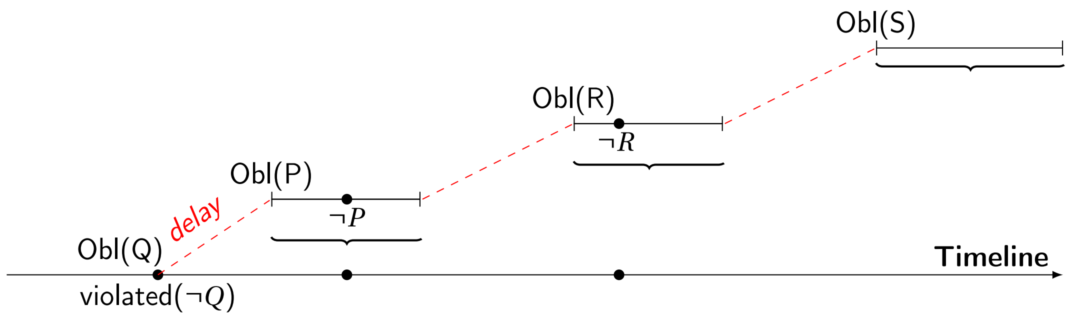


Figure 6.8: Recursive Compensation

Figure 6.8 illustrates the idea behind the recursive compensation of the violated obligation. Assume an obligation in force Q, at some point in time (black dot), is violated ($\neg Q$), and depending on its violation conditions, Q is compensated by P; that is, *Compensates*(P, Q). Then P at some point in time, is violated ($\neg P$) and, after some delay, P is compensated by another obligation R; that is, *Compensates*(R, P) that aim to amend the violation of Q. Now, if R is violated ($\neg R$) and, given its violation conditions, it is further compensable a new obligation S comes into force to compensate; that is, *Compensates*(S, R).

Now, given the violation and compensation conditions $\neg$*RecursivelyCompensable*(P), the violation of P itself, is not recursively compensable; but it is compensated to compensate the violation of Q. In the same manner, $\neg$*RecursivelyCompensable*(R) is not recursively compensable; however, to compensate the violation of P, which is compensated to recursively compensate Q, and the violation of R, is compensated by S to recursively compensate Q; that is, *RecursivelyCompensable*(S, Q). Note that the aim of the recursive compensation is to amend the effects of the violation of an obligation. The cycle of compensation, depending on the violation and compensating conditions, continues until a sub–ideal situation is achieved, or a penalty for the violation is enforced.

6.4.3 Modelling Obligations with Extended EC

Next, we use these predicates and events, giving the deontic EC semantics for the various types of norms described in Chapter 3. Also, we give the generic axioms required for modelling various cases of obligations prescribed in the business contract. These axioms specify the conditions for no legal effects (that is, not deontically *HoldsAt*) after termination of an obligation (A13), and the conditions when no fluent deontically holds (A14):

$$\neg DHoldsAt(X, T+1) \longleftarrow \exists E : DTerminates(E, X, N, T) \tag{A13}$$

$$\begin{aligned} \neg DHoldsAt(X, T_k) \longleftarrow \neg DHoldsAt(X, T) \wedge \neg Happens(trigger(X, N), T_j) \wedge \\ (T \leq T_k) \wedge (T \leq T_j + N \leq T_k) \end{aligned} \tag{A14}$$

Remark 6. *In what follows, we will have several situations where the trigger for the obligation not only triggers the initiation for the obligation but also the termination. This means that we have to write the expression in the following form:*

$$DTerminates(trigger(X, N), X, N, T) \tag{Eq.1}$$

where we have to repeat the parameters X and N twice. To avoid the repetition of these parameters, we used the convention of dropping the X and N from the arguments DTerminates to improve readability; thus, we have:

$$DTerminates(trigger(X, N), T) \tag{Eq.2}$$

Accordingly, the expressions in (Eq.2) is a shorthand for the expression in (Eq.1).

6.4.3.1 Punctual Obligation

The deontic EC axioms that describe when a *punctual* obligation holds as follows:

$$\begin{aligned} & DHoldsAt(O^{p,T_s}X, T_s) \leftarrow \\ & \quad \exists T_t, N : Happens(trigger(O^{p,T_s}X, N), T_t) \wedge \\ & \quad (T_s = T_t + N) \wedge N \geq 0 \end{aligned} \tag{A15}$$

$$\begin{aligned}&\mathit{DTerminates}(\mathit{trigger}(O^{p,T_s}X,N),T_s) \leftarrow\\ &\quad \exists T_t,N: \mathit{Happens}(trigger(O^{p,T_s}X,N),T_t) \wedge\\ &\quad (T_s = T_t + N) \wedge N \geq 0\end{aligned} \tag{A16}$$

Let us examine in detail the above axioms. In axiom (A15), an obligation is represented as a fluent; specifically, the (punctual) obligation of X is represented by the fluent $O^{p,T_s}X$ where O^{p,T_s} is an obligation modality (a specific type of obligation) and time when the obligation comes into force (T_s), and X is a variable referring to the contents of an obligation. The special triggering event *trigger*($O^{x,T}X,N$) initiates the obligation by replacing the *Initiates* and is embedded in *Happens* predicate. (A16) specifies that the same event that triggers the obligation terminates the obligation, and the obligation terminates at the same time instant when it is initiated. Thus, in combination with (A13), we have a punctual obligation is in force for a one time instant only. The axiom specifying when a punctual obligation is violated as follows:

$$\begin{aligned}&\mathit{Happens}(\mathit{violation}(O^{p,T_s}X),T_v) \leftarrow\\ &\quad \mathit{DHoldsAt}(O^{p,T_s}X,T_s) \wedge\\ &\quad \neg \mathit{Happens}(X,T_s) \wedge \neg \mathit{HoldsAt}(X,T_s) \wedge (T_v = T_s)\end{aligned} \tag{A17}$$

The violation of a punctual obligation happens when we do not have the contents of the obligation at the right time. This can happen in the following two cases:

- *the content is a fluent and does not hold at the time*; or
- *it is an event and does not happen at the time.*

6.4.3.2 Persistent Obligation

The following axiom describes a persistent obligation with a *natural deadline* when the fluent holds in interval:[9]

$$\begin{aligned}&\mathit{DHoldsAt}(O^{per,T_s}X,T_k) \leftarrow\\ &\quad \exists T_t,N: \mathit{Happens}(\mathit{trigger}(O^{per,T_s}X,N),T_t) \wedge\\ &\quad \neg \mathit{DClipped}(T_s,O^{per,T_s}X,T_k) \wedge\\ &\quad \mathit{DTerminates}(\mathit{trigger}(O^{per,T_s}X,N),T_e) \wedge\\ &\quad (T_s = T_t + N) \wedge (T_e > T_s) \wedge (T_s \leq T_k \leq T_e) \wedge N \geq 0\end{aligned} \tag{A18}$$

[9]The definition of *DClipped* is the same as that for *Clipped* where *Terminates* is replaced by *DTerminates*.

Here, by the *natural deadline* means that if no other (relevant) event happens, the obligation is in force from the T_s and T_e, and that T_e is determined by the same event that triggers the (persistent) obligation.

Achievement and *Maintenance* obligations are two distinct cases of persistent obligation. Next, we give various cases of achievement obligations such as, initiation, violations and termination predicates in deontic EC (DEC).

6.4.3.3 Achievement Obligation

An achievement obligation is a special case of a persistent obligation where there might not be a natural deadline for the obligation. Hence, there are two cases of achievement obligations:

(i) *when the obligation has no termination point*; that is, initiation of the achievement obligation.

$$\begin{aligned} &DHoldsAt(O^{a,T_s}X, T_s) \leftarrow \\ &\quad \exists T_t, N : Happens(trigger(O^{a,T_s}X, N), T_t) \wedge (T_s = T_t + N) \wedge N \geq 0 \end{aligned} \tag{A19}$$

(ii) *The obligation Holds at a particular time point deontically initiated and not clipped between the interval*; that is, start time and the point until it holds.

$$\begin{aligned} &DHoldsAt(O^{a,T_s}X, T_k) \leftarrow \\ &\quad DHoldsAt(O^{a,T_s}X, T_s) \wedge \neg DClipped(T_s, O^{a,T_s}X, T_k) \wedge (T_s \leq T_k) \end{aligned} \tag{A20}$$

Since achievement obligations must be fulfilled within the stipulated time, they can be achieved and terminated even before the deadline. Unlike a punctual obligation, which is terminated by the same triggering event that initiates it, an arbitrary event can terminate the achievement obligations. Accordingly, there are two cases for the termination of achievement obligations:

(A) *An arbitrary event terminates the obligation when the obligation conditions are fulfilled before the deadline of and obligation.*

$$\begin{aligned} &DTerminates(-, O^{a,T_s}X, N, T_k) \leftarrow \\ &\quad Happens(-, T_k) \wedge DHoldsAt(O^{a,T_s}X, T_k) \wedge \\ &\quad (Happens(X, T_k) \vee HoldsAt(X, T_k)) \wedge \\ &\quad FulfillTerminable(O^{a,T_s}X) \wedge (T_s \leq T_k) \end{aligned} \tag{A21}$$

The symbol '$-$' represents an arbitrary event, which can be anything; for example, a new obligation, an activity, or even a deadline that terminates the obligation.

(B) *The deadline itself terminates the obligation.*

$$\begin{aligned} &\mathit{DTerminates}(\mathit{deadline}(O^{a,T_s}X, T_d), T_d) \leftarrow \\ &\quad \mathit{Happens}(\mathit{deadline}(O^{a,T_s}X), T_d) \wedge (T_s \leq T_d) \end{aligned} \tag{A22}$$

The axiom terminating the preemptive achievement obligations is as follows:

$$\begin{aligned} &\mathit{DTerminates}(-, O^{a,T_s}X, N, T_e) \leftarrow \\ &\quad \mathit{Happens}(-, T_e) \wedge \mathit{DHoldsAt}(O^{a,T_s}X, T_s) \wedge \\ &\quad \exists T' : (\mathit{Happens}(X, T') \vee \mathit{HoldsAt}(X, T')) \wedge \\ &\quad \mathit{FulfillTerminable}(O^{a,T_s}X) \wedge \\ &\quad (T_e = T_s + 1) \wedge (T' < T_s) \end{aligned} \tag{A23}$$

The earlier introduced predicate *FulfillTerminable* aims to check whether the obligation can be terminated upon fulfillment. This leaves us to determine the conditions under which a violation of an achievement obligation occurs. To capture the violation of an achievement obligation, we introduced a *deadline event* (see, Section 6.4.2.2), signalling the deadline after which a violation occurs if the achievement obligation is not fulfilled by that time/event.

$$\begin{aligned} &\mathit{Happens}(violation(O^{a,T_s}X), T_v) \leftarrow \\ &\quad \mathit{DHoldsAt}(O^{a,T_s}X, T_e) \wedge \\ &\quad \mathit{Happens}(deadline(O^{a,T_s}X), T_e) \wedge \\ &\quad (\neg \mathit{Happens}(X, T_e) \wedge \neg \mathit{HoldsAt}(X, T_e)) \wedge \\ &\quad \mathit{ViolationTerminable}(O^{a,T_s}X) \wedge (T_v = T_e) \end{aligned} \tag{A24}$$

6.4.3.4 Maintenance Obligation

Maintenance is another case of persistent obligations; it is different from achievement in the sense that the obligation conditions must be fulfilled for every instant of the interval the obligation is in force. The axiom (A18) can represent the maintenance obligations. Unlike an achievement obligation, a maintenance obligation is violated if the obligation contents are not fulfilled for all the instances.

$$\begin{aligned} &\mathit{Happens}(\mathit{violation}(O^{m,T_s}X), T_k) \leftarrow \\ &\quad \mathit{DHoldsAt}(O^{m,T_s}X, T_k) \wedge \\ &\quad \neg \mathit{Happens}(X, T_k) \wedge \neg \mathit{HoldsAt}(X, T_k) \wedge (T_s \leq T_k) \end{aligned} \tag{A25}$$

The violation of a maintenance obligation can terminate the obligation if the obligation is *ViolationTerminable*, which is again, a boolean switch for checking whether a maintenance obligation can be terminated upon violation. The conditions for termination after the violation are:

$$\begin{aligned}&DTerminates(O^{m,T_s}X, T_v) \leftarrow\\&\quad Happens(violation(O^{m,T_s}X), T_v) \wedge\\&\quad ViolationTerminable(O^{m,T_s}X)\end{aligned} \tag{A26}$$

For a non–perdurant maintenance obligation, the violation of the obligation itself terminates the obligation.

$$\begin{aligned}&DTerminates(violation(O^{m,T_v}X), T_v) \leftarrow\\&\quad DHoldsAt(O^{m,T_v}X, t_v) \wedge ViolationTerminable(O^{m,T_s}X) \wedge\\&\quad Happens(violation(O^{m,T_s}X), T_v) \wedge (T_s \leq T_v)\end{aligned} \tag{A27}$$

6.4.3.5 Compensation Obligation

As mentioned earlier in Section 6.3.2, one of the major shortcomings of PENELOPE's semantic properties is that they cannot handle the obligation arising from the violation of an obligation. This is because the PENELOPE semantics do not admit obligations after violations; that is, violations without reparations. In other words, no obligation will be initiated if, for any reasons, an obligation is violated. We now provide deontic semantics that show *how the compensations obligations arising from the violations can be handled.* Since, a compensation is an obligation itself, we introduced a special event *compensates,* where an event triggering a compensation is the violation of a norm that compensation compensates. The domain–specific axioms for the two cases of compensation are:

(1) *Compensation of the violation by a single obligation*:

$$\begin{aligned}&Happens(Compensates(O^{x,T_s}P), T_{s_c}) \leftarrow\\&\quad \exists O^{y,T_{sc}}Q : (Compensates(O^{y,T_{sc}}Q, O^{x,T_s}P), T_{s_c}) \wedge\\&\quad Happens(violation(O^{x,T_s}P), T_v) \wedge\\&\quad DHoldsAt(O^{y,T_{sc}}Q, T_{s_c}) \wedge\\&\quad (Happens(Q, T_{s_c}) \vee HoldsAt(Q, T_{s_c})) \wedge (T_s \leq T_v \leq T_{s_c})\end{aligned} \tag{A28}$$

(2) *Recursive compensation when a compensation obligation itself is violated*:

$$\begin{aligned}&Happens(Compensation(O^{x,T_s}P), T_{s_c}) \leftarrow\\&\quad Compensates(O^{y,T_{sc}}Q, O^{x,T_s}P) \wedge\\&\quad Happens(violation(O^{y,T_{sc}}Q), T_v) \wedge\\&\quad Happens(Compensation(O^{y,T_{sc}}Q), T_z) \wedge\\&\quad RecursivelyCompensable(O^{x,T_s}P) \wedge (T_s \leq T_{s_c} \leq T_z) \wedge (T_v \leq T_z)\end{aligned} \tag{A29}$$

For the two axioms above, we introduced the special event *Compensation*, indicating that a violated deontic fluent has been compensated for, and the binary predicate *Compensates* where the two arguments are two deontic fluents. The meaning of *Compensates* is that fulfilling the first deontic fluent makes amends for the violation of the second deontic fluents and implements the *Comp* function introduced in Chapter 3 (Definition 7). Again, the predicate *RecursivelyCompensable* is a boolean switch meant to capture the intuition given by condition 2 of Definition 9.

The events and predicates introduced in this section allow capturing the *deontic effects* of obligations from *when they come into force, not from when the event is triggered*; we have formally shown this to be impossible with existing variants of EC. Thus, PENELOPE's semantics properties cannot give a faithful representation of legal norms. In addition, new axioms provide the semantics for different types of obligations, and various conditions associated with these obligations.

6.5 Solving PENELOPE'S Issues with Deontic EC

Now, we formally show how newly extended EC can be used to address the issues with PENELOPE's semantics, to acquire the effects of different types of obligations on the tasks of a business process for populating the *State* and *force* functions. The deontic axiom modelling the situations where the obligation enters into force with the occurrence of the event is as follows:

$$\begin{aligned}&\mathit{DHoldsAt}(O^{p,T_s} ImmediatelyAcknowledge, T_s) \longleftarrow\\&\quad \exists T_t, N : \mathit{Happens}(trigger(Complaint, 0), T_t) \land\\&\quad \mathit{Happens}(Complaint, T) \land\\&\quad (\mathit{HoldsAt}(inPerson, T) \lor \mathit{HoldsAt}(byPhone, T)) \land\\&\quad (T_s = T_t + N) \land N \geq 0\end{aligned} \tag{A30}$$

Let the event *Complaint* occur at state T, and the obligation fluent *byPhone* holds at the same state. Then, from domain axiom (A30), we derive $\mathit{Happens}(trigger(Complaint, 0), T_t)$; and then from axioms: (A15),(A16) and (A13), we obtain $\mathit{DHoldsAt}(O^{p,T_s} ImmediatelyAcknowledge, T_s)$ and $\neg\mathit{DHoldsAt}(O^{p,T_s} ImmediatelyAcknowledge, T+1)$. This means that we have an obligation to immediately acknowledge the complaint on its receipt; that is, obligation enters into force with the event occurrence. Now, assume that we model acknowledgment as an *event*, and we have an event *Complaint* at state T—that is,

$Happens(Complaint, T)$—then, at this state, the conditions of violation do not hold. Suppose now that $Happens(ImmediatelyAcknowledge, T)$ is not true; that is, the complaint is not acknowledged; thus, $\neg Happens(ImmediatelyAcknowledge, T)$ is true at state T. In addition, given that *ImmediatelyAcknowledge* is an event, if we have $\neg HoldsAt(ImmediatelyAcknowledge, T)$, then we can use Axiom (A17) to conclude that the obligation to acknowledge the oral complaint by phone on the spot has been violated.

To address the violation–handling problems with PENELOPE's semantics, in a previous section, we introduced a *violation* event. The axioms capturing the effects when an obligation is violated as follows:

$$\begin{aligned} &DHoldsAt(O^{a,T_s}PayInvoice, T_s) \longleftarrow \\ &\quad \exists T_t, N: Happens(trigger(O^{a,T_s}PayInvoice, 0), T_t) \wedge \\ &\quad Happens(PayInvoice, T) \wedge Happens(PayInvoice, T') \wedge \\ &\quad (T_s = T_t + N) \wedge N \geq 0 \wedge T' < T \leq T_s \end{aligned} \tag{A31}$$

$$\begin{aligned} &Happens(violation(O^{a,T_s}PayInvoice), T_v) \longleftarrow \\ &\quad DHoldsAt(O^{a,T_s}PayInvoice, T_e) \wedge \\ &\quad Happens(deadline(O^{a,T_s}PayInvoice), T_e) \wedge \\ &\quad (\neg Happens(PayInvoice, T_e) \wedge \neg HoldsAt(PayInvoice, T_e)) \wedge \\ &\quad (T_v = T_e) \wedge T_s < T_e \end{aligned} \tag{A32}$$

Let us assume that an event *PayInvoice* holds at time T, and that from domain Axiom (A31), we then derive the triggering conditions—that is, $Happens(trigger(O^{a,T_s}PayInvoice, 0), T_t)$—and then, again from Axiom (A31), we obtain $DHoldsAt(O^{a,T_s}PayInvoice, T)$; this mean that (A31) holds at time T. Now assume that the agent does resolve the complaint by the deadline, and the violation of the obligation occurs at T_v. Then, from domain Axioms: (A31) and (A22) we obtain $DHoldsAt(O^{a,T_s}PayInvoice, T_s)$ and $Happens(deadline(O^{a,T_s}PayInvoice), T_e)$ meaning that the obligation to PayInvoice holds at state T_s, and the payment must be made by the deadline T_e. Now suppose that the payment is not made by T_e, then from Axiom (A17) we derive $Happens(violation(O^{p,T_s}PayInvoice), T_v)$; that is, we have state T_v where the obligation is violated. Next, to give the violation semantics, suppose we model the obligation fluent $PayInvoice$ as an event and, in the first instance, we assume that payment is made thus, we have $Happens(PayInvoice, T)$, meaning that fluent obligation fluent is true. Suppose now that $Happens(PayInvoice, T)$ is not true; that is, the complaint is not resolved by T_e; thus, $\neg Happens(PayInvoice, T_e)$ becomes true at state T_e. Accordingly, given that $PayInvoice$ is an event, if $\neg HoldsAt(PayInvoice, T_e)$ is also true, then using Axiom (A24) we can conclude that the obligation to pay the invoice by the deadline has been violated at T_e.

Finally, we give the axiom for the compensation obligation amending the violation of the obligation to the pay invoice in (A32).

$$\begin{aligned}
&Happens(Compensation(O^{a,T_s} PayInvoice), T_{S_c}) \longleftarrow \\
&\quad \exists O^{a,T_{S_c}} PayInvoice + 3\% Interest : \\
&\quad (compensates(O^{a,T_{S_c}} PayInvoice + 3\% Interest, O^{a,T_s} PayInvoice), T_{S_c}) \wedge \\
&\quad Happens(violation(O^{a,T_s} PayInvoice), T_v) \wedge \\
&\quad DHoldsAt(O^{a,T_{S_c}} PayInvoice + 3\% Interest, T_{S_c}) \wedge \\
&\quad (Happens(PayInvoice + 3\% Interest, T_{S_c}) \vee \\
&\quad HoldsAt(PayInvoice + 3\% Interest, T_{S_c}) \wedge \\
&\quad (T_v \leq T_{S_c})
\end{aligned} \tag{A33}$$

Assume a compensation obligation $PayInvoice + 3\% Interest$ comes into force at time T_{S_c}. Then, from Axiom (A33), we derive a special event $compensates(O^{a,T_{S_c}} PayInvoice + 3\% Interest, O^{a,T_s} PayInvoice)$, and then from axioms: (A26) and (A33), we obtained $Happens(violation(O^{a,T_s} PayInvoice), T_v)$ and $DHoldsAt(O^{a,T_{S_c}} PayInvoice + 3\% Interest, T_{S_c})$ respectively. In other words, the deontic fluent $PayInvoice$ violated at time instant T_v is compensated by the compensating event that comes into force at T_{S_c}. The binary predicate $Compensates$, in its arguments, indicates two deontic fluents, and the meaning of the predicate is that fulfilling the first deontic fluent compensates the violation of the second deontic fluent. Accordingly, for the last condition, from (A21) we derive fluent $Happens(PayInvoice + 3\% Interest, T_{S_c}) \vee HoldsAt(PayInvoice + 3\% Interest, T_{S_c}$, which either deontically happens or holds at time T_{S_c}. Hence, in combination with the compensation event derived from axiom (A33); that is, $Happens(Compensation(O^{a,T_s} PayInvoice), T_{S_c})$ compensates the violation.

6.6 Related Work

Primarily EC has been extensively used in multi-agents systems (MASs), for modelling and reasoning about the agents behaviour, interaction, and planning where it has rich publication record (Alberti et al., 2006; Chesani et al., 2013; Flores and Kremer, 2002; Yolum and Singh, 2002). The legal domain, on the other hand, has equally exploited the capabilities of EC for reasoning normative systems along the temporal dimensions.

Fornara and Colombetti (2009) provide formal specifications of commitments and pre-commitments, institutionalised power, and context using the EC. The formal representation of norms is limited to obligations and permissions only, as in (Goedertier and Vanthienen, 2006c). Also, they do not make any distinction between different types of obligations and effects of violations on obligations as we do although the notion of sanctions has been formally presented in the study. A rather similar work by Artikis et al. (2005) proposed EC–based formal specifications of obligations and permissions in the context of Ad–Hoc Networks. In contrast, we make a clear distinction between the various types of obligations in terms of their temporal aspect of validity, the effects they produce and the effects of violations on them.

Bandara et al. (2003) translate both the policies and system behaviour specifications into a formal specification, using EC. The proposed formal specifications are expressive enough to efficiently model the systems, using various types of policies representing obligations. These formal specifications can be used, together with the abductive reasoning, for detecting and representing the conflicts between the policy specifications (particularly those related to the authorisation and permissions). Their formal specifications, used with abductive reasoning, are useful in the sense that a priori knowledge about the event/fluent state can be used to simplify the representation of preemptive obligations. Such obligations are fulfilled even before they come into force. In this work, we do not consider a priori knowledge of the events/fluent, but use the notion of preemptiveness to distinguish different cases of the violation of an achievement obligation and model in the EC.

Yolum and Singh (2004) study norms as *social commitments* capturing the obligations in the context of protocols. The authors employ Shanahan's full EC (Shanahan, 1997) for modelling base–level and conditional commitments. Primarily, the study focuses on the persistent commitments and provides EC axioms for reasoning commitments and operations on them. Also, it deals with the violation of commitments in the context of protocols to identify the non–compliant behaviour of an agent. In contrast, we go beyond the basic obligation types and provide axioms for various obligation types including the persistence effects of obligations even after the violation of an obligation. Overall, the notion of commitments used in this work is somewhat similar to ours; however, their classes of commitment are context–specific, while our classes of obligations are context–independent. Also, unlike our work, they do not have the notion of temporal proposition for modelling time interval, which is imperative for modelling maintenance obligations.

Paschke and Bichler (2005) provide a logical framework for automating the electronic contracts for representing complex business rules and business policies. The authors integrate the EC into other logical formalism—such that, Horn Logic and Deontic Logic and ECA rules—to model the contract states and deontic concepts (for example, obligations and permissions) as time-varying fluent. Evans and Eyers (2008) use EC for encoding deontic clauses of contracts for data use rules and the monitoring of subsequent compliance with these rules. Their work is similar to that of Grosof et al. (1999), with the exception that the logic programming used in their work is formalised in EC to represent the deontic state explicitly while latter is a declarative approach to modelling various contract rules types. Our work is different from these studies: as we consider rather complex obligation types, and the effects of violations, while they simply work on basic deontic notions; that is, achievement, permissions and prohibitions.

In Marín and Sartor (1999), an analysis of two temporal profiles— (i) *internal and external time of the validity*, and (ii) *rules attributing the applicability of legal norms after the triggering of an event*—is provided. These notions are then formalised using EC conceptualisation and correct representation of the legal feature norms for automated reasoning. However, the effects of the occurrence of an event (expected or unexpected) that might cause the validity of the norm, the effects of how and when a norm is terminated, and the violation of norms, have not been covered in their analysis. Also, no analysis has been performed on how such effects can be correctly translated for automated reasoning.

Alrawagfeh (2013) propose a norms representation approach using EC, thus enabling the agents to use norms in their practical reasoning. Also, this study considers only two classes of norms—*obligations* and *prohibitions*—for which authors introduced three fluents; that is, *fPun* and *oPun* referring to obligation norm violation and prohibition norm violation respectively, and *oRew* for obligation fulfilment. The extended fluent can be used for representing the norms that are composed of several actions, together with the norm's context. Alrawagfeh's approach is rather limited because it does not consider other norms and various obligation types as we do. Also, this study follows the Anderson's reduction view of norm that suggests every violation of a norm is followed by a sanction (Arend, 2001), and the similar notion violation without reparation is employed in (Goedertier and Vanthienen, 2006c), and essentially leads to termination of interaction, and a penalty can be imposed. However, we argue that, initially, sanctions are not/cannot be directly imposed in every case, as a sub–ideal situation can still make a business process compliant. The notion of compensation obligations, and obligations perduring after the violations (shown in this chapter), are the norms types that strengthen our argument.

The analysis of three formalisms conducted in (Elgammal et al., 2011a) compares the abilities and limitations of formal languages chosen from the temporal and deontic families of logics. Their analysis specifically looks at the formality of these formalisms for modelling compliance requirements, and 11 other features of the chosen languages. On the other hand, Ly et al. (2013) evaluates five frameworks against core compliance management functionalities from different domains. These evaluations are fundamentally limited in scope as they do not consider the compliance modelling languages and constructs for the specifications of norms. In comparison, we evaluate the expressive power of LTL and EC, using the semantics to define temporal properties of norms in terms of temporal validity of norms and the effects of violations on other obligations; at the same time, we highlight the issues with these formalisms that result from the modelling of different types of obligations.

6.7 Summary

In this chapter, we contributed a detailed formal semantic evaluations of LTL and EC based frameworks. The aim was to find the answer whether existing CMFs based on these formalisms can fully represent legal norms in a conceptually sound way. We used the constructs proposed in the COMPAS and PENELOPE frameworks based on these formalisms. Our evaluations show that the answer to the question seems *negative* because both the evaluated CMFs based on these two formalisms show several shortcomings in providing conceptually sound modelling support for representing legal norms.

We began by modelling the clauses of a plausible scenario inspired by the real life legal norms, using COMPAS patterns based on the compliance request language (CRL) to translate them into LTL formulas. The translation showed that any formalisation based on LTL is not suitable for representing the legal norms of the scenario. However, it does necessarily mean that LTL *per se* is not able to represent the scenario. The formalism is equally suitable to provide the formalisation of the specification of process models, but it lacks the expressiveness to give the formalisation of the specifications of legal norms. However, the CRL's patterns giving the specifications of the scenarios translated into LTL formulas do not fully capture the semantics obligations. For example, the CRL Else/ElseNext compensation pattern formalised into LTL, cannot represent maintenance obligations and prohibitions prescribed by the scenarios; we have shown that how to address this problem. Another problem with the LTL is that it cannot model permissions, since they do not play a direct role in compliance but the clauses in the scenario that specify permissions. Since, legal theory admits permissions as the absence of obligations, we remarked that the F temporal operator can be used to model obligations, and is a natural choice to model prohibitions. But if we do not use F then how to represent permissions—or LTL does not support permissions.

As COMPAS and its underlying compliance requirements language include most of the patterns used by other CMFs, our evaluation of LTL can be equally extendible to other temporal logic based CMFs, such as, DELCARE and BMPMN–Q. As DELCARE is based on LTL, it will have the same limitations as COMPAS. BPMN-Q, on the other hand, is based on CTL, which is the superset of LTL. The natural question is then whether the branching time logics with path quantifiers, such as, CTL and CTL*, are more suited to modelling permissions. In such logics, permissions could be modelled by EF. While modelling permissions using path quantifiers seems a better option and provides more flexibility for modelling norms, it does not solve the problem with the scenario we proposed, given that the problem requires just only a single (non)–branching trace to arise, and thus path quantifiers are essentially irrelevant. We can safely argue, therefore, that, overall LTL–based CMFs do not seem to be suitable for the verification of business processes against the rule sets of legal norms, as the results they provide are not aligned with the expected outcomes based on the legal interpretation of the scenario. The reason is that they fail to represent and reason with the norms in a conceptually sound way. Accordingly, they cannot be used to check compliance of real business processes with real norms.

We also evaluated the formal semantics of PENELOPE, a design-time CMF based on EC. Similar to the case with LTL, EC also has fundamental deficiencies in modelling different types of obligations. In particular, it is not possible to model the *punctual* obligations with PENELOPE semantics because of the problems with the *Initiates* predicate of EC. Since, *Initiates* predicate is not able to capture the effects of various obligations on the tasks of a process, it not possible to check the compliance. We also formally proved that PENELOPE's violation semantics, wrongly evaluate the violations at deadlines. Moreover, PENELOPE does not admit the notion of compensatory obligations because of the use of the notion of *violation without reparation*. To address some deficiencies with PENELOPE semantics, we introduced a *deontic extension* to EC to show how to address the problems with a formalism for representing legal norms. We introduced a special triggering event replacing the *Initiates* predicate. The triggering event, with the help of delay, captures the deontic effects of an obligation from the time it comes into force rather than from the time when the event triggering the obligation occurs. Essentially, the delay determines the difference in time between when the triggering event occurs and when the obligation comes into force thus allows the capturing of deontic effects of legal norms; that is, *DHolds*.

Similar to the trigger for the initiates, we introduced a triggering event *DTerminates*, which deontically terminates an obligation. We then used the newly proposed predicates and events and trivially modelled different types of obligations, the violation situations, and obligations arising from the violations; these cannot be modelled with PENELOPE's existing semantics. Essentially, the proposed extension increases the ability of EC to provide support for all types of obligations; this is not possible with the existing base predicate *Initiates* used in PENELOPE, which thus cannot capture the nuances and effects of the obligations for business process compliance checking.

CHAPTER 7

EPILOGUE

To conclude this thesis, we summarise and discuss its main contributions and limitations, and shed some light on possible avenues for future research.

7.1 Synopsis

In response to the ever–changing organisational compliance reporting requirements, researchers have shown a wider interest in, and have proposed several CMFs for providing automated compliance checking of legal norms. These CMFs address the compliance problem from a variety of perspectives, and offer specific capabilities. Regardless of their nature and type, and how good and flexible these CMFs can be, their effectiveness largely of depends on the underlying conceptual and formal models that provide the reasoning support to model various types of normative requirements. For the most part, CMFs are grounded on various formal models that use different formalisms for reasoning and modelling the legal component of business process compliance. Given the extensibility of the compliance domain, and the existence of a large breed of CMFs determining the suitability of a CMF for effective compliance reporting is a difficult task, and requires special tools and methodologies. However, as the literature on business process compliance suggests, the business process compliance domain lacks the accepted tools and methodologies *to evaluate the abilities of a CMF*, in particular, *to evaluate the effectiveness of its* *conceptual and formal models that provide the reasoning support to model the legal component of the compliance problem.* In this thesis, we addressed this shortcoming and presented a *formal framework* comprising several methodologies to evaluate the abilities of existing CMFs.

In addressing the *key* questions of this thesis, we presented a formal framework that contributes: (i) a classification model for normative requirements, and the formal semantics for each class of the classification required to properly model these normative requirements; (ii) methodologies to evaluate the conceptual and formal foundations of existing CMF; and (iii) a deontic extension to Event–Calculus (EC) showing that how to address the problems with a formal language if we want to use it to properly model and reason about different types of normative requirements in a conceptually sound way. The framework has been exhaustive developed by combining formal methods and case study methods and provides the formal foundations for evaluating the abilities of existing CMFs.

In order to address Question 1, and Question 2, we asked: *What are the generic classes of normative requirements for which* a CMF should be able to provide full reasoning and modelling support? To address this question, we designed a classification model (presented in Chapter 3) that provides a rich ontology of deontic notions, for example, obligations, permissions, violations and compensations. These notions are further divided into sub–classes along temporal dimensions of the validity of the norms. Also, along the temporal dimensions, we specified when an obligation comes into force and until what time it remains in force, or when it is violated at a particular point in time. These classes of normative requirements have been obtained in a systematic and exhaustive way using the well–known divide and conquer method. In order to define the meanings of each class in the classification model, along temporal dimensions, we provided formal definitions in terms of the temporal validity of obligations, what constitutes a violation, and the effects of violations on other types of obligations. We did not restrict ourselves to any particular formalism, as the provided semantics are generic ones and can be represented in any formal language despite beign grounded with deontic logic in mind. We validated this fact, later in (Chapter 4), where we formally modelled these notions over WF-nets giving formal specifications of business processes; and in Chapter 6, we modelled them using Event-Calculus (EC). The proposed classification model provides a list of generic classes (and sub-classes) of the normative requirements with which a CMF need to comply with.

With the new classes and semantic definitions of the norms now established, the natural question was: *How can we properly model and check the compliance of new classificatory classes of the classification model*? To address this question, Chapter 4 proposes a compliance checking approach that provides the formal models of business processes and the specifications of norms, and details the steps required to properly model the legal component of compliance. We began the process by providing the formal models for business process specifications and normative requirements, which are integral components of the modelling of the compliance of business processes. For the specifications of a formal model for business processes, we described a process as the sequence of states corresponding to the execution of a process model using WF–nets. We then used these sequences of states to provide the formal model of the norm classes provided in the proposed classification model. Next, we formally defined what it means to comply with, or violate, a norm. Finally, we integrated these formal models with an intermediary mechanism (adopted from Sadiq et al., 2007) to semantically annotate business processes for compliance checking purposes. To practically demonstrate the effectiveness of the compliance checking approach, we used a real–life complaint–handling process.

To address Question 1, the main question of this research, and to practically demonstrate *how to evaluate existing CMFs, systems, approaches, and languages,* we put the designed framework into practice. The outcome is the contribution of several conceptual and formal evaluations of existing CMFs; these are presented in Chapters 5 and Chapter 6 respectively. The classifications model and the formal semantics provided the bases for these evaluations, as they were used them to examine whether existing CMFs are able to fully represent the classificatory classes of our classification model.

For the conceptual evaluations (Chapters 5), we examined the conceptual foundations of the selected CMFs—in particular, their underlying conceptual models—and asked; What construct are provided for modellign the norms? Which formal languages are used? How are the norms linked for compliance checking and What is the level of compliance management? We adopted a case study–based sound evaluation methodology, which allowed us to start the evaluation with minimal information available on the CMFs. Under this methodology, we selected seven CMFs, using pre–defined evaluation criteria, which were determined as the result of expert discussions. We then evaluated the various constructs proposed in a CMF to examine their correspondence with the semantic definitions of each norm class, in terms of their ability to fully capture the meanings of the specific type of norms. Our conceptual evaluations portray somewhat a *bleak* picture that shows that not all the existing CMFs are fully able to provide compliance checking support for all types of normative requirements. Some frameworks cannot fully represent different types of norms; for example, a specific notion might not be fully represented by a proposed construct, or the idea of the notion might not be considered in the CMF.

Chapter 6 presents the formal evaluations of CMFs from two prominent families of formalism: LTL and EC. The aim of these formal evaluations was to find the answer whether CMFs based on these formalisms can fully represent the classes of our classification model proposed in Chapter 3. The constructs provided in the COMPAS and PENELOPE CMFs, which based on the LTL and EC formalism, were used for the evaluation. Using the plausible scenarios inspired by real life legal norms, we evaluated the COMPAS compliance requirement language (CRL) patterns and translated them into LTL formulas. Our evaluation shows that any formalisation of CRL patterns–based on LTL is not suitable for representing the different types of norms in particular permissions, maintenance, and compensations. Because COMPAS uses the patterns that are mostly used by other LTL based CMFs such as BPMN–Q and DECLARE, the evaluation results can be simply extended to these CMFs. Thus, we concluded the CMFs based on LTL are not suitable for representing legal requirements in a conceptually sound way. Hence, they cannot be used to check compliance of real business processes with real norms.

We also evaluated the formal semantics of PENELOPE, a design–time CMF based on EC. Similar to LTL, EC also has fundamental deficiencies in modelling different types of obligations; in particular, there are problems with the EC predicates *Initiates* and *Terminates*, which fails to capture the effects of the tasks on business processes. Thus, PENELOPE is incapable of checking the compliance of various types of obligations, such as *punctual obligations*. We have also formally shown that the PENELOPE's violation semantics wrongly evaluate the violations at deadlines. Moreover, PENELOPE does not admit the notions of compensatory obligations, because of its use of the notion of violation without reparation. Hence, PENELOPE can by no means be relied upon for the checking and verification of compliance requirements. To address problems with the EC predicates, we then proposed a *deontic extension to EC,* and introduced new predicates and events. The newly introduced predicates increases, the expressive power of EC, enabling it to trivially model all types of obligations and the notions that PENELOPE cannot model. These include, but are not limited to, prohibitions, violations, and compensations. Moreover, the proposed extension also illustrates that *how to address the problems with a formal language,* if we want to use it for properly modelling and reasoning about all types of normative requirements in a conceptually sound way.

7.2 Limitations

This research developed a formal framework to evaluate the abilities of existing CMFs to represent normative requirements in a conceptually sound fashion. Given the breadth of the compliance domain where a plethora of CMFs exist, each of the CMFs addresses the compliance problem either at design-time, run-time or post-execution time, and offers a number of core functionalities. Due to extensibility of the compliance domain, it is not possible to address all the problems faced by the domain. Hence, to keep this research in a manageable scope, the main limitations of this thesis are:

- a CMF might offer a number of functionalities e.g., representation and checking of normative requirements, violation detection and explanation, remedial actions to recover from violations etc. This thesis deals only with norms representation functionality of CMFs *whether they can properly represent legal norms.* Other functionalities such as compliance enforcement, monitoring, traceability, reporting and violation handling etc., are not addressed.
- only design-time CMFs are considered while run-time and post-execution time CMFs have been excluded from this research.

7.3 Avenues for Future Work

Research is a never–ending process. Although the presented study achieved its specific goals, the work presented here can be improved in various ways.

First, to validate the effectiveness of the overall framework—that is, the classification model and the compliance checking approach—we evaluated CMFs from two families of logics: LTL and EC. The evaluation shows that the proposed framework is flexible and can be used to evaluate any CMF, albeit grounded in different formal languages. Further evaluations can be carried out against other formalisms—such as, first–order–logic, π–calculus, deontic and defeasible logic—to determine whether these languages can provide reasoning support for all types of normative requirements for a more comprehensive validation of our framework. A step in this direction can be the works proposed in (Governatori, 2015; Hashmi et al., 2014).

Second (in Chapter 5), we evaluated norms modelling constructs of the selected CMFs. Besides these CMFs there are formalism independent languages to represent legal norms. For example, Nómos 3 is a *primitive*–based compliance verification language that uses primitives as notations to design graphical models. Graphical models are used to reason about the compliance requirements, and the roles with the norms. Essentially, Nómos is a *conceptual graph-based* norm modelling language (Croitoru et al., 2012). Analysis of such formalism independent languages can certainly provide further insights into the state–of–the–affairs as well as their shortcomings.

Another line of interest can be gaining a better understanding of the concerns on the usability and the generalisation of norms modelling patterns evaluated in this work—in particular, the balance between the semiotic clarity of the concepts and the proposed patterns. A first step in this direction can be a detailed usability study with the non-technical experts and industry professionals that can provide valuable insights on these issues. A theoretical framework proposed by Figl et al. (2009) can be used as a guiding framework to gain more understanding on the balance between the complexity and the expressiveness of the modelling constructs, and logic formulas to carry out such usability studies.

With respect to the deontic extension to EC, we included new deontic predicates and events to the calculus. The deontic extension addresses the issues with EC, and provides a theoretical contribution and it has not been implemented. Gaining a detailed understanding on the effects of the extended predicates and events remains a posible future work.

On the same note, the deontic extension proved to be expressive in capturing the nuances of all types of normative requirements. However, we are unaware of any complexities that might arise as the result of the inclusion of new predicates to the EC semantics. A detailed analysis of the expressiveness of the extended calculus is thus required to gain a deeper understanding of the potential complexities. The techniques proposed in Cervesato et al. (2000) can be used to analyse the complexity of various extensions to EC increasing its expressiveness.

A comparative analysis of the expressiveness of the proposed deontic extension to EC with other formalisms notably: linear temporal logic, deontic and defeasible logic, first–order–logic, features and fluents calculus, and situation calculus could certainly provide more insights into the *state–of–the–affairs* of formal modelling languages for modelling and representing the legal knowledge. Most importantly, it would highlight the deficiencies of these languages with respect to automated business process compliance checking.

Appendices

APPENDIX A

SYNTHETIC BUSINESS CONTRACT

This Deed of Agreement is made between 'ABC', an independent body operating in the Education Sector, which has its office at place PEL Tce APZ, and the legal entity company 'UBIX' Inc., which has its office at Place Liberty Avenue, MPL.

WHEREAS the independent **ABC** (hereafter *Principal*) requires the professional engineering, equipments, and other services as defined in the expression of interests proposal.

WHEREAS , The **UBIX Inc.** (hereafter known as *Contractor*) can provide the services (as per the terms of services) to the *Principal*, and is in the business of discharging —and has the capabilities, resources, and experience to discharge—the obligations set forth in the terms and conditions of this Agreement.

1. **DEFINITIONS**

   a) **Agreement:** This agreement comprises (i) this document; (ii) the request for proposal dated: XYZ; (iii) the qualifications and proposal dated XYZI; and (iv) all other schedules attached hereto or incorporated herein by reference.

   b) **Services:** 'Services' include all the obligations (as per terms of services) to perform the professional engineering services, to supply the equipment, and other services defined in the program and as described in the proposal.

c) **Obligation:** 'Obligation' refers to a course of action corresponding to the duties or services to be rendered under the conditions set forth in this Agreement.

d) **Effectiveness of This Agreement:** This Agreement shall become effective when signed by both the *Principal* and the *Contractor*.

e) **Execution of This Agreement:**

   i. **Start Date:** The date when this Agreement is signed.

   ii. **End Date:** 90 days from the signing date of this Agreement.

2. **TERMS OF SERVICES**

2.1 **Scope of the work**: The services including their related general and special terms and conditions as described in Section 3 of this Agreement.

a) **Changes in the Scope of the Work:** The *Principal* can request major or minor changes in the scope of the work at any time during the period of this Agreement. Under such circumstances, the *Contractor* shall extend full co-operation and accordingly make/implement such changes as per the change request. The change request shall be within the scope of the work agreed upon in this Agreement, and shall be subject to the following conditions.

   (i) *Major changes:* A major change, if requested, might require new requirements or changes in the specifications of the tendered equipment. These requirements or changes might include (but might not be limited to) the required model, make, or manufacturer of the equipment; technical (and/or configuration) specifications; the number of required items; the acquisition of new services etc., or

   (ii) *Minor changes:* A minor change, if requested, might require the renewed requirements or changes in the specifications of the tendered equipment. These might include (but might not be limited to) changes to the agreed upon configuration, change to location of the supply of the equipment, alteration (or alternative) in cabling, and/or its installation etc., thereof.

b) If the *Principal* requests a major change in the services as described in Section 3 (subject to sub-clauses) per sub-clause 2.1.a(i) hereof, the

Contractor reserves the right either to (i) accept, or (ii) reject, any such major change request, and

c) Pursuant to clause 2.1.b(i), the *Contractor* shall make any such acceptance only by a separate amendment.

d) The *Contractor* has the right to draw up new conditions for a major change request in addition to the conditions of services defined in this Agreement.

e) Pursuant to clause 2.1(d), the new conditions drawn up for the major change shall be sent to the *Principal* for approval.

f) Subject to clauses 2.1(d) – 2.1(e), the contract shall be amended to include approved new changes for an accepted major request.

g) Pursuant to sub-clause 2.1.a(ii), for a minor change request, the *Contractor* shall accept such minor changes as an amendment to this Agreement, without drawing up any new conditions.

h) The *Principal* shall pay the *Contractor* the extra costs of the requested changes per the provisions set forth in Section 5 (sub-clauses 5a-5d).

3. **PERFORMANCE OF SERVICES:** The CONTRACTOR (pursuant to Sub-section 2.1, if applicable) shall be responsible for the following:

   a) **Delivery of Equipments:** The conditions for the delivery of equipments (as listed in Annexure X) are:

      i. The *Contractor* must deliver the equipment within 7 working days to the *Principal*'s designated location, within 7 working days of the date this Agreement is signed.

      ii. All the supplied equipment must be completely new, and conform to the requirements set forth by the *Principal.*

      iii. The *Contractor* must also provide adequate certification that the supplied equipment is in alignment with the state-of-the art technology.

      iv. The *Contractor* must also provide certification that the supplied equipment comply (and conform) to all applicable regulations, and shall indemnify the *Principal* from any breaches committed by the *Contractor.*

v. The *Contractor* shall be responsible for all transportation, labour, taxes, and freight charges related to the supply of the equipment.

vi. The *Contractor* must prepare an Equipment Delivery Report (EDR) for submission no later than 3 working days from the actual delivery.

b) **Installation&Testing of Equipment**

i. The *Contractor* must install, as per the provided specifications/configurations, the supplied equipment not later than 5 working from the submission of an EDR.

ii. If needed, the *Contractor* can alter the provided specifications, if needed so, to best fit the *Principal*'s requirements, or to enable the equipment to function properly.

iii. Subject to sub-clause 3b(ii), the *Contractor* must inform the *Principal* of any alternations made to the agreed upon configurations, and provide complete documentation of such alterations including drawings and other relevant information.

iv. The *Principal* shall not be responsible for any extra cost(s) arising as the result of any alterations made, such as the costs of spare parts or any new equipment required. Such expenses will be the sole responsibility of the *Contractor*.

v. The *Contractor* must prepare and provide an installation report within 3 working days of equipment installation.

vi. The *Contractor* must test the installation and functioning of the installed equipment, under the supervision of the *Principal*'s representative.

vii. The testing of equipment must begin as soon as practicable, and not longer than 5 working days after its installation of equipments.

viii. The *Contractor* must document all the testing phases in consultation with the *Principal*'s representative and submit a testing report (TR) within 7 working days.

c) **Staff Training**

The *Contractor* shall be responsible for training the staff who will operate the newly installed system. This will be conducted as detailed below:

i. **The Users:** The *Contractor* must train the staff members who are the users of the newly installed system in the general use of the system, and in the use of its various components in particular.

ii. **The Maintenance Staff:** If the staff members are responsible for the maintenance of the new system, the *Contractor* must train them to maintain the system in a manner that ensures its maximum performance.

iii. The staff training can be provided either:

   A. during the testing of the equipment, OR

   B. once the system is fully commissioned and operational,

iv. The *Contractor* must provide any material (e.g., user manuals, systems manuals, equipment manuals or any other relevant documents) required for training purposes.

v. The *Contractor* shall be responsible for all expenses incurred in providing the staff training (e.g., hiring of training staff, or training material etc.

vi. The training must be completed before the delivery of the operating system (see, clause 3[d]).

d) **Delivery of the Operating System**

i. The *Contractor* shall be responsible for delivering the operating system within 90 days (see, sub-clause 1e.ii) from the time of the signing of this contract.

ii. The *Contractor* is obliged to provide all mandatory and ancillary documentation related to the operating system.

e) **Penalties for Delaying in Delivering the Operating System**

i. In a case where the *Contractor* fails to deliver the up and operating system within the agreed upon time, the *Contractor* is obliged to pay the *Principal*, as per payment conditions (see, Section 5a-5d), a penalty amounting to 0.25% of the sum of the contract amount for each calendar day of default. However, the penalty amount shall not exceed 10% of the total amount of the contract.

ii. Subject to Section 3d(i), if the default causes damage to the *Principal*, the latter can charge the *Contractor* the cost of damage incurred as well as the penalty cost, as per the payment policies listed in Section 5.

iii. Any penalty under sub Section 3d(i)–3d(ii) shall be charged on a separate invoice to be issued by the *Principal* to the *Contractor*.

iv. The *Principal* shall not take any action or impose a penalty for any matter subject to Section 7, unless it is resolved.

v. Pursuant to Section 8 (in part or full), the *Contractor* shall not suspend and/or delay the performance of their duties under this Agreement.

vi. Subject to clause (c) Section 7, the *Contractor* shall not suspend/terminate its services pending resolution of on any conflict, and shall continue its operation, as agreed upon in this Agreement.

4. **RESPONSIBILITIES OF THE PRINCIPAL**

The *Principal*:

a) Shall provide timely access to the places/sites where the equipment will be delivered and installed.

b) Must assist the *Contractor* in undertaking their duties.

c) Must not hinder or cause anything that might hinders/limits the *Contractor*'s ability to discharge their duties defined in this Agreement.

d) Shall pay all the agreed upon payments/compensations as per the Clause 5 of this Agreement.

e) Shall issue a Release Notice (RN) upon successful completion of the work and receipt of a full commissioning report from the Head of the relevant division.

5. **TERMS OF PAYMENTS**

The conditions of payment are as the follows:

a) The *Contractor* shall issue an invoice(s) for making payment claims against the provided services.

b) The *Principal* is obliged to pay (in full) all the payments (and/or penalties) to the *Contractor* after receiving an invoice for the performed services.

c) Any payment(s) must be paid within 15 days from the date of the receipt of the invoice.

d) Pursuant to clause 5(c), if the *Principal* fails to pay the invoice, the total invoice amount is subject to 3% default surcharge per calendar day.

e) If payment is not made within 7 days of the default, another 0.25% interest per day shall be applied to the compound amount of the invoice, and the payment must be made within the next 10 working days.

f) Pursuant to sub-clause 7(c), the *Principal* may suspend or delay any payments until any conflict(s) are resolved to the satisfaction of both parties, The *Principal* shall then make any suspended / outstanding payments within 3 working days of such resolution.

g) Subject to Section 8 (sub-clauses a–b), the *Principal* shall not be held responsible for any payments in relation to (any) duties, defined in this Agreement, to any party outside this Agreement.

h) Subject to Section 12, sub-clauses 12a and 12b, the *Principal* shall not be held responsible for any payments after the termination of this Agreement.

RESPONSIBILITIES OF THE CONTRACTOR

6. **INSURANCE**

Under the terms of, and during the term of, this Agreement:

a) The *Contractor* shall maintain, during the term of this agreement, the insurance coverage for the items listed in Exhibit C.

b) The *Contractor* shall furnish insurance certificates showing the types and amounts of coverage, and the expiration dates of such policies, and

c) The *Contractor* shall furnish a statement that no insurance under such policies will be terminated/cancelled by the insurer(s) without 30 days' prior notice to the *Principal.*

7. **DISPUTE HANDLING PROCESS**

In the event of any dispute or complaints (arising for whatever reason[s]), both the parties shall resolve the dispute in accordance with the sub-clauses 7(a)–7(c).

a) **Internal Complaints Resolution:** All complaints pertaining to this contract herein shall be:

   a Making Complaints: Complaints can be made in person, by phone, or by email.

      A. **Acknowledgment**

         A. Immediately acknowledged, if received in person or by phone, or

         B. Acknowledge within 2 working days if received by email or letter.

   b All received complaints shall be resolved within 7 working days.

b) **Escalation of Complaints**

   (i) Subject to sub-clause 7.a.(ii), if a complaint is not resolved within the stipulated time frame, it must be escalated to the Head of the relevant Division Head.

   (ii) An escalated complaint must be resolved within 3 days of being escalated.

c) **Dispute Arbitration**

   Under this Agreement, in the event of any conflict, both the parties reserve the right to seek independent arbitration from an external entity (including a legal entity, such as a court of law), if the complaint is not resolved under the internal complaint handling procedure pursuant to clauses 7.a and 7.b.

8. **DELEGATION AND SUBCONTRACTING**

The delegation and conditions for subcontracting are as given below:

a) In principle, the *Contractor* shall be responsible for the provision of assigned professional services, deliveries and any other services, as defined in the terms of services, in its entirety.

b) The *Contractor* shall not delegate (or subcontract) the performance of the work, or any portion thereof which is, by this Agreement, to be undertaken by the *Contractor*.

c) The *Principal* reserves the right to terminate this Agreement without any further notice, if the *Contractor* fails to discharge contractual duties under sub-clauses 8(a)–8(b), and if there is reason to believe that the *Contractor* is in breach of this Agreement.

9. **PRIVACY AND CONFIDENTIALITY**

   The protection and safeguard of the private and confidential information of the *Principal* will be strictly observed. Throughout the duration of this Agreement, the *Contractor*:

   a) Shall retain and maintain all personal and highly-sensitive information in strict confidence, using such a degree of care as is appropriate to avoid unauthorised access, use, or disclosure.
   b) Use the provided information solely and exclusively for the purposes for which the information, or access to it, is provided pursuant to the clause-9(a) of the Agreement.
   c) Unless instructed by a court, shall not distribute, make available, or otherwise disclose the provided information to a third party(ies).
   d) Pursuant to Section 9(a–c), this Agreement is terminated by default without any further notice if the *Contractor* has breached confidentiality, or if there is reason to believe that the privacy of the confidential information has been compromised.

10. **CREDIT RATING**

    a) It is the *Contractor*'s responsibility to maintain a positive credit balance with their credit institution for the whole duration of the contract.
    b) The credit rating must not drop below the acceptable minimum rating level of B^+ for the whole duration of the contract.

11. **FORCE MAJEURE**

    In no event shall either party be responsible or liable for any failure or delay in performance that results, directly or indirectly, in whole or in part, from any cause or circumstances beyond their control. Such causes and circumstances shall include, but not limited to: fire; floods; strikes; riots; sabotage; explosion; adverse weather conditions; unavoidable causalities; unavailability of labour; acts of God, or a government agency; or loss of permits that do not arise from the actions or responsibilities of either party.

Work stoppages or interruptions in the delivery of services under this agreement that are caused by any of the above circumstances might result in additional costs beyond outlined by the *Principal*in the proposal. This occurrence shall entitle the *Contractor* to an adjustment in the charges and fees for services under this Agreement.

12. **TERMINATION OF THE AGREEMENT**
    The termination conditions for the contract are:

    a) **Termination by Fulfillment:** The said contract automatically ceases to exist under the following conditions:

       i. When all the required services rendered under this contract have been fulfilled to the *Principal*'s satisfaction; and
       ii. When neither party liable pursuant to any clause (or sub-clauses) of this contract; or involved in any court proceedings at the time of termination.

    b) **Termination by Cause:** The *Principal* and the *Contractor* each reserves the right, in their reasonable discretion, to terminate this agreement at any time, and without any liability to the other party, under the following clauses:

       i. If the *Principal* enters in 3 invoice defaults against payment for any penalties or payment for materials, equipment or services rendered the contractual obligations. In this case, the *Principal* must pay the *Contractor* the whole amount of the contract plus the agreed penalties, as per the procedure defined in this contract.
       ii. In the event of the *Contractor*'s non-performance of services, as described in the terms and conditions of the contract.
       iii. On the receipt of an unfavorable credit report, or some other reasonable indicator(s), that the other party will not be able to discharge their obligations under this Agreement.
       iv. Pursuant to Section 9(a–c), in the event of failure to protect and safeguard the *Principal*'s personal and confidential information. In this case, this Agreement is terminated by default.

    c) **Termination without Cause:**

i neither party may terminate this agreement without any cause.

ii if the contract is terminated without any reasonable cause, subject to Section 5, the terminating party shall pay the damages caused to the other party (This can be equivalent to the full amount of the contractual value).

END OF AGREEMENT

Bibliography

Abate, F. and Jewell, E. J., editors (2001). *New Oxford American Dictionary*. Oxford University Press.

Abdullah, N. S., Sadiq, S., and Indulska, M. (2010). Emerging Challenges in Information Systems Research for Regulatory Compliance Management. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering*, CAiSE'10, pages 251–265. Springer-Verlag.

Accorsi, R., Lowis, L., and Sato, Y. (2011). Automated Certification for Compliant Cloud-based Business Processes. *Business & Information Systems Engineering*, 3(3):145–154.

Ågotnes, T., van der Hoek, W., Rodríguez-Aguilar, J. A., Sierra, C., and Wooldridge, M. (2007). On the logic of normative systems. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1175–1180.

Ågotnes, T., Van der Hoek, W., and Wooldridge, M. (2010). Robust Normative Systems and a Logic of Norm Compliance. *Logic Journal of IGPL*, 18(1):4–30.

Agrawal, R., Johnson, C., Kiernan, J., and Leymann, F. (2006). Taming Compliance with Sarbanes-Oxley Internal Controls Using Database Technology. In *Proceedings of the 22nd International IEEE Conference on Data Engineering*, page 92.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., and Torroni, P. (2007). Expressing and Verifying Business Contracts with Abductive Logic Programming. In Boella, G., van der Torre, L., and Verhagen, H., editors, *Normative Multi-agent Systems*, number 7122 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany.

Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., and Torroni, P. (2006). Compliance Verification of Agent Interaction: A Logic-based Software Tool. *Applied Artificial Intelligence*, 20(2-4):133–157.

Allaire, M. and Governatori, G. (2014). On the Equivalence of Defeasible Deontic Logic and Temporal Defeasible Logic. In Dam, H., Pitt, J., Xu, Y., Governatori, G., and Ito, T., editors, *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, volume 8861 of *LNCS*, pages 74–90. Springer International Publishing.

Alrawagfeh, W. (2013). Norm Representation and Reasoning: A Formalization in Event Calculus. In Boella, G., Elkind, E., Savarimuthu, B., Dignum, F., and Purvis, M., editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *LNCS*, pages 5–20. Springer.

Arbab, F. (2004). REO: A Channel-based Coordination Model for Component Composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366.

Arbab, F., Kokash, N., and Meng, S. (2009). Towards Using REO for Compliance-Aware Business Process Modeling. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 108–123. Springer.

Arend, S. (2001). Pluralism and Law. In *Proceedings of the 20th IVR World Congress of the International Association of Philosophy of Law and Social Philosophy*, volume 4, page 104.

Artikis, A., Kamara, L., Pitt, J., and Sergot, M. (2005). A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. In Leite, J., Omicini, A., Torroni, P., and Yolum, P., editors, *Declarative Agent Languages and Technologies II*, volume 3476 of *LNCS*, pages 221–238. Springer.

Arya, A., van Dongen, B., and van der Aalst, W. (2010). Towards Robust Conformance Checking. In *Business Process Management Workshops'10*, pages 122–133.

Ashby, S. (2008). Operational Risk: Lessons from Non-Financial Organisations. *Journal of Risk Management in Financial Institutions*, 1:406–415.

Awad, A. (2007). BPMN-Q: A Language to Query Business Processes. In *Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*, pages 115–128.

Awad, A. (2010). *A Compliance Management Framework for Business Process Models.* PhD thesis, Hasso Plattner Institut, Potsdam University, Germany.

Awad, A., Decker, G., and Weske, M. (2008a). Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM*, LNCS, pages 326–341. Springer.

Awad, A., Polyvyanyy, A., and Weske, M. (2008b). Semantic Querying of Business Process Models. In *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*, pages 85–94.

Awad, A., Weidlich, M., and Weske, M. (2009). Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. In Baresi, L., Chi, C.-H., and Suzuki, J., editors, *Service-Oriented Computing*, volume 5900 of *Lecture Notes in Computer Science*, pages 500–515. Springer Berlin / Heidelberg.

Awad, A., Weidlich, M., and Weske, M. (2011). Visually Specifying Compliance Rules and Explaining their Violations for Business Processes. *Journal of Visual Languages & Computing*, 22(1):30–55.

Awad, A. and Weske, M. (2009). Visualisation of Compliance Violations in Business Process Models. In *5th Workshop on Business Process Intelligence*, volume 9, pages 182–193.

Bai, X., Liu, Y., Wang, L., Tsai, W.-T., and Zhong, P. (2009). Model-Based Monitoring and Policy Enforcement of Services. In *proceedings of the 2009 World Conference on Services–I*, pages 789 –796.

Baier, C. and Katoen, J.-P. (2007). *Principles of Modeling Checking.* MIT Press, Cambridge.

Bandara, A., Lupu, E., and Russo, A. (2003). Using Event Calculus to Formalise Policy Specification and Analysis. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)*, pages 26–39.

Bandara, W., Miskon, S., and Fielt, E. (2011). A Systematic, Tool-supported Method for Conducting Literature Reviews in Information Systems. In Virpi, T., Joe, N., Matti, R., and Wael, S., editors, *Proceedings of 19th European Conference on Information Systems, ECIS 2011, Helsinki,Finland.*

Baral, C. and Zhao, J. (2007). Non-monotonic Temporal Logics for Goal Specification. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 236–242. Morgan Kaufmann Publishers Inc.

BCBS (2013). Basel III: The Liquidity Coverage Ratio and Liquidity Risk Monitoring Tools.

Becker, J., Delfmann, P., Eggert, M., and Schwittay, S. (2012). Generalizability and Applicability of Model-Based Business Process Compliance-Checking Approaches – A State-of-the-Art Analysis and Research Roadmap. *BuR - Business Research Journal*, 5(2):221–247.

Becker, M. and Laue, R. (2012). A Comparative Survey of Business Process Similarity Measures. *Computers in Industry*, 63(2):148–167.

Bench-Capon, T. and Gordon, T. F. (2009). Isomorphism and Argumentation. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law*, ICAIL '09, pages 11–20, New York, USA. ACM.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2001). *System and Software Verification - Model Checking Techniques and Tools*. Springer.

Birukou, A., D'Andrea, V., Leymann, F., Serafinski, J., Silveira, P., Strauch, S., and Tluczek, M. (2010). An Integrated Solution for Runtime Compliance Governance in SOA. In *ICSOC*, pages 122–136.

Bonatti, P. A., Shahmehri, N., Duma, C., Olmedilla, D., Nejdl, W., Baldoni, M., Baroglio, C., Martelli, A., Coraggio, P., Antoniou, G., Peer, J., and Fuchs, N. E. (August 2004). Rule-based Policy Specification: State of the Art and Future Work, REWERSE Project Report-i2-D1. Report, Universitá di Napoli Fedrecio II.

Bonazzi, R. and Pigneur, Y. (2009). Compliance Management in Multi-Actor Contexts. *Proceedings of International Workshop on Governance, Risk and Compliance (GRCIS), An ancillary meeting of CAISE*.

Broersen, J. (2006). Strategic Deontic Temporal Logic as a Reduction to ATL, with an Application to Chisholm's Scenario. In Goble, L. and Meyer, J.-J., editors, *Deontic*

Logic and Artificial Normative Systems, volume 4048 of *Lecture Notes in Computer Science*, pages 53–68. Springer Berlin Heidelberg.

Cabanillas, C., Resinas, M., and Ruiz-Cortés, A. (2010). On the Identification of Data-related Compliance Problems in Business Processes. In *VI Jornadas Científico-Técnias En Servicios Web Y SOA (JSWEB'10)*, volume 1, pages 89–102, Valencia, España.

Cabannilas, C., Resinas, M., and Ruiz-Cortes, A. (2010). Hints on How to Face Business Process Compliance. In *III Taller de Procesos de Negocio e Ingenieria de Servicios PNIS10 in JISBD10*, volume 4, pages 26–32.

Cervesato, I., Franceschet, M., and Montanari, A. (2000). A Hierarchy of Modal Event Calculi: Expressiveness and Complexity. In Barringer, H., Fisher, M., Gabbay, D., and Gough, G., editors, *Advances in Temporal Logic*, volume 16 of *Applied Logic Series*, pages 1–20. Springer Netherlands.

Chesani, F., Mello, P., Montali, M., and Torroni, P. (2013). Representing and Monitoring Social Commitments using the Event Calculus. *Autonomous Agents and Multi-Agent Systems*, 27(1):85–130.

Clark, A. and Dawson, R. (1999). *Evaluation Research: An Introduction to Principles, Methods and Practice*. SAGE Inc., European Foundation Centre, 1060 Brussels, Belgium.

COBIT (2007). Control Objectives for Information Related Technology - COBIT 4.1.

COSO (1994). Internal control –integrated framework.

Croitoru, M., Oren, N., Miles, S., and Luck, M. (2012). Graphical Norms via Conceptual Graphs. *Knowledge Based Systems*, 29:31–43.

Cunningham, H., Maynard, D., Tablan, V., Ursu, C., and Bontcheva, K. (2001). *Developing Language Processing Components with GATE: A User Guide.*

Daniel, F., Casati, F., D'Andrea, V., Mulo, E., Zdun, U., Dustdar, S., Strauch, S., Schumm, D., Leymann, F., Sebahi, S., de Marchi, F., and Hacid, M.-S. (2009). Business Compliance Governance in Service-Oriented Architectures. In *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on*, pages 113 –120.

D'Aprile, D., Giordano, L., Gliozzi, V., Martelli, A., Pozzato, G., and Theseider Dupré, D. (2010). Verifying Business Process Compliance by Reasoning about Actions. In Dix, J., Leite, J. a., Governatori, G., and Jamroga, W., editors, *Computational Logic in Multi-Agent Systems*, volume 6245 of *LNCS*, pages 99–116. Springer.

de Maat, E. and Winkels, R. (2007). Categorisation of Norms. In *Legal Knowledge and Information Systems - JURIX 2007, The Netherlands*, pages 79–88.

de Maat, E. and Winkels, R. (2010). Automated Classification of Norms in Sources of Law. In Francesconi, E., Montemagni, S., Peters, W., and Tiscornia, D., editors, *Semantic Processing of Legal Texts*, volume 6036 of *Lecture Notes in Computer Science*, pages 170–191. Springer Berlin Heidelberg.

de Medeiros, A. K. A. and van der Aalst, W. (2005). Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. *Electronice Notes in Theoratical Computer Science.*, pages 3–21.

de Moura Araujo, B., Schmitz, E. A., Correa, A. L., and Alencar, A. J. (2010). A Method for Validating the Compliance of Business Processes to Business Rules. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10)*, pages 145–149, New York, USA. ACM.

DECLARE (2010). Declarative Process Models, http://www.win.tue.nl/declare/.

Dehnert, J. and Rittgen, P. (2001). Relaxed Soundness of Business Processes. In Dittrich, K., Geppert, A., and Norrie, M., editors, *Advanced Information Systems Engineering*, volume 2068 of *LNCS*, pages 157–170. Springer Berlin / Heidelberg.

Dijkman, R. M., Dumas, M., and Ouyang, C. (2008). Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology*, 50(12):1281 – 1294.

Doganata, Y. and Curbera, F. (2009). Effect of Using Automated Auditing Tools on Detecting Compliance Failures in Unmanaged Processes. In *Business Process Management*, pages 310–326.

Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Managment.* Springer Berlin Heidelberg.

Dwyer, M., Avrunin, G., and Corbett, J. (1999). Patterns in Property Specifications for Finite-state Verification. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 411–420.

El Kharbili, M. (2012). Business Process Regulatory Compliance Management Solution Frameworks: A Comparative Evaluation. In *APCCM 2012*, CRPIT 130, pages 23–32.

El Kharbili, M. and Stein, S. (2008). Policy-Based Semantic Compliance Checking for Business Process Management. In *MobIS Workshops*, CEUR Workshops 420, pages 178–192.

El Kharbili, M., Stein, S., Markovic, I., and Pulvermüller, E. (2008). Towards a Framework for Semantic Business Process Compliance Management. *Banking*, 08(i):1–15.

Elgammal, A. (2012). *Towards A Comprehensive FramewCompliacBusiness Process Compliance*. PhD thesis, Tiburg Universtity.

Elgammal, A., Turetken, O., van den Heuvel, W.-J., and Papazoglou, M. (2011a). On the Formal Specification of Regulatory Compliance: A Comparative Analysis. In *Proceedings of ICSOC'10*, pages 27–38.

Elgammal, A., Turetken, O., van den Heuvel, W.-J., and Papazoglou, M. (2011b). On the Formal Specification of Regulatory Compliance: A Comparative Analysis. In *Proceedings of ICSOC'10*, pages 27–38.

Elgammal, A., Turetken, O., van den Heuvel, W.-J., and Papazoglou, M. (2014). Formalizing and Applying Compliance Patterns for Business Process Compliance. *Software & Systems Modeling*, pages 1–28.

Elgammal, A., Türetken, O., van den Heuvel, W.-J., and Papazoglou, M. P. (2010). Root-Cause Analysis of Design-Time Compliance Violations on the Basis of Property Patterns. In *ICSOC*, pages 17–31.

Evans, D. and Eyers, D. M. (2008). Deontic Logic for Modelling Data Flow and Use Compliance. In *Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-hoc Computing*, MPAC '08, pages 19–24. ACM.

Evans, G. P. (2014). Managing Risk with an End-to-End Process View: Adopting a Process-based Approach to Risk Management. *BPTrends Article*.

Fellmann, M. and Zasada, A. (2014). State-of-the-Art of Business Process Compliance Approaches. In *22nd European Conference on Information Systems, ECIS 2014, Tel Aviv, Israel, June 9-11, 2014*.

Figl, K., Mandling, J., and Strembeck, M. (2009). Towards a Usability Assessment of Process Modelling Languages. volume 554 of *CEUR Workshop Proceedings*, pages 138–156.

Flores, R. A. and Kremer, R. C. (2002). To Commit or Not to Commit: Modeling Agent Conversations for Action. *Computational Intelligence*, 18(2):120–173.

Fongon, P. and Grillo, K. (2004). Corporate Implications of Sarbanes Oxley Act: A Public Polcy.

Fornara, N. and Colombetti, M. (2009). *Specifying Artificial Institutions in the Event Calculus*, pages 335–366. IGI Global.

Förster, A., Engels, G., and Schattkowsky, T. (2005). Activity Diagram Patterns for Modeling Quality Constraints in Business Processes. In *MoDELS'05*, pages 2–16.

Förster, A., Engels, G., Schattkowsky, T., and Straeten, R. V. D. (2006). A Pattern-Driven Development Process for Quality Standard-conforming Business Process Models. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006)*, pages 135 –142.

Frank, D., Jan, B., Virginia, D., and John-Jules, C. M. (2004). Meeting the Deadline: Why, When and How. In *Formal Approaches to Agent-Based Systems, Third InternationalWorkshop, FAABS 2004, Greenbelt, MD, USA, April 26-27, 2004, Revised Selected Papers*, pages 30–40.

Gambini, M., Rosa, M., Migliorini, S., and Hofstede, A. (2011). Automated Error Correction of Business Process Models. In Rinderle-Ma, S., Toumani, F., and Wolf, K., editors, *Business Process Management*, volume 6896 of *LNCS*, pages 148–165. Springer Berlin Heidelberg.

Ghanavati, S., Amyot, D., and Peyton, L. (2007). Towards a Framework for Tracking Legal Compliance in Healthcare. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, pages 218–232, Berlin, Heidelberg. Springer-Verlag.

Ghose, A. and Koliadis, G. (2007). Auditing Business Process Compliance. In Krämer, B., Lin, K.-J., and Narasimhan, P., editors, *Service-Oriented Computing (ICSOC 2007)*, volume 4749 of *LNCS*, pages 169–180. Springer.

Giblin, C., Liu, A. Y., Müller, S., Pfitzmann, B., and Zhou, X. (2005). Regulations Expressed As Logical Models (REALM). In *Proceeding of the 18th Annual Conference on Legal Knowledge and Information Systems (JURIX 2005)*, pages 37–48. IOS Press.

Gilliot, M. and Accorsi, R. (2009). Runtime Predictions of Policy Violations in Automated Buisness Processes. Extended Abstract: presented at Prime Life/IFIP Summer School Program, 7-11 September, Nice/France.

Goedertier, S. and Vanthienen, J. (2006a). Business Rules for Compliant Business Process Models. In *Proceeding of the 9th International Conference on Business Information Systems (BIS 2006)*, volume P-85 of *Lecture Notes in Informatics*, pages 558–579. Gesellschaft für Informatik.

Goedertier, S. and Vanthienen, J. (2006b). Compliant and flexible business processes with business rules. In *BPMDS. CEUR Workshop Proceedings, CEUR-WS.org, vol 236*.

Goedertier, S. and Vanthienen, J. (2006c). Designing Compliant Business Processes with Obligations and Permissions. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops 2006*, LNCS 4103, pages 5–14. Springer.

Gordon, T. (1993). The Pleadings Game. *Artificial Intelligence and Law*, 2(4):239–292.

Gordon, T. F., Governatori, G., and Rotolo, A. (2009). Rules and Norms: Requirements for Rule Interchange Languages in the Legal Domain. In *RuleML 2009*, LNCS 5858, pages 282–296. Springer.

Governatori, G. (2005). Representing Business Contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216.

Governatori, G. (2015). Thou Shalt is not You Will. In Atkinson, K., editor, *Proceedings of the Fiftheenth International Conference on Artificial Intelligence and Law*, New York: ACM.

Governatori, G. and Hashmi, M. (2015a). No Time for Compliance. In *Proceedings of 19th IEEE the Enterprise Computing Conference (EDOC'15), Adelaide Australia.*

Governatori, G. and Hashmi, M. (2015b). Permissions in Deontic Event-Calculus. In *Proceedings of the 28th International Conference on Legal Knowledge and Information Systems (Jurix' 15), Braga Portugal.* [Short Paper].

Governatori, G., Hoffmann, J., Sadiq, S. W., and Weber, I. (2008a). Detecting Regulatory Compliance for Business Process Models Through Semantic Annotations. In *Business Process Management Workshops'08*, pages 5–17.

Governatori, G., Hulstijn, J., Riveret, R., and Rotolo, A. (2007a). Characterising Deadlines in Temporal Modal Defeasible Logic. In *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*, AI'07, pages 486–496, Berlin, Heidelberg. Springer-Verlag.

Governatori, G. and Milosevic, Z. (2005). Dealing with Contract Violations: Formalism and Domain Specific Language. In *9th International Enterprise Distributed Object Computing Conference (EDOC 2005)*, pages 46–57. IEEE Computer Society.

Governatori, G., Milosevic, Z., and Sadiq, S. (2006a). Compliance Checking between Business Processes and Business Contracts. In *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, pages 221–232.

Governatori, G., Milosevic, Z., and Sadiq, S. (2006b). Compliance Checking between Business Processes and Business Contracts. In *10th International Enterprise Distributed Object Computing Conference (EDOC 2006)*, pages 221–232. IEEE Computing Society.

Governatori, G., Olivieri, F., Scannapieco, S., and Cristani, M. (2011). Designing for Compliance: Norms and Goals. In *RuleML 2011*, Ft Lauderdale.

Governatori, G. and Rotolo, A. (2006). Logic of Violations:A Gentzen System for Reasoning with Contrary-To-Duty Obligation. *Australasian Journal of Logic*, 4:193–215.

Governatori, G. and Rotolo, A. (2008a). An Algorithm for Business Process Compliance. In *Legal Knowledge and Information Systems*, pages 186–191. IOS Press.

Governatori, G. and Rotolo, A. (2008b). Changing Legal Systems: Abrogation and Annulment. Part II: Temporalised Defeasible Logic. In *Proceedings of Normative Multi Agent Systems (NorMAS 2008)*.

Governatori, G. and Rotolo, A. (2010a). A Conceptually Rich Model of Business Process Compliance. In *Proceedings of APCCM '10*, volume 110, pages 3–12.

Governatori, G. and Rotolo, A. (2010b). Norm Compliance in Business Process Modeling. In *RuleML 2010: 4th International Web Rule Symposium*, pages 194–209. Springer.

Governatori, G., Rotolo, A., Riveret, R., Palmirani, M., and Sartor, G. (2007b). Variants of Temporal Defeasible Logics for Modelling Norm Modifications. In *Proceedings of the 11th International Conference on Artificial Intelligence and Law*, ICAIL '07, pages 155–159, New York, USA. ACM.

Governatori, G., Rotolo, A., and Sartor, G. (2005). Temporalised Normative Positions in Defeasible Logic. In *Proceedings of the 10th international conference on Artificial intelligence and law*, ICAIL '05, pages 25–34, New York. ACM.

Governatori, G. and Sadiq, S. (2009). The Journey to Business Process Compliance. In *Handbook of Research on Business Process Management*, pages 426–454. IGI Global.

Governatori, G. and Shek, S. (2013). Regorous: A Business Process Compliance Checker. In *International Conference on Artificial Intelligence and Law (ICAIL) 2013*, pages 245–246, Rome. ACM.

Governatori, G., Thakur, S., and Pham, D. H. (2008b). A Compliance Model of Trust. In *Proceedings of the 2008 conference on Legal Knowledge and Information Systems: JURIX 2008: The Twenty-First Annual Conference*, pages 118–127, Amsterdam, The Netherlands. IOS Press.

Grosof, B. N., Labrou, Y., and Chan, H. Y. (1999). A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proceedings of the 1st*

ACM conference on Electronic commerce, EC ’99, pages 68–77, New York, USA. ACM.

Han, J., Jin, Y., Li, Z., Phan, T., and Yu, J. (2007). Guiding the Service Composition Process with Temporal Business Rules. In *Web Services 2007*.

Hashmi, M. (2015). A Methodolgy for Extracting Legal Norms from Regulatory Documents. In *Proceedings of 8th International Workshop on Evolutionary Business Processes (EVL-BP 2-15), co-located with EDOC 2015, Adelaide Australia*.

Hashmi, M. and Governatori, G. (2013). A Methodological Evaluation of Business Process Compliance Management Frameworks. In Song, M., Wynn, M. T., and Liu, J., editors, *Asia Pacific Business Process Management*, volume 159 of *LNBIP*, pages 106–115. Springer, Switzerland.

Hashmi, M., Governatori, G., and Wynn, M. (2015a). Normative Requirements for Regulatory Compliance: An Abstract Formal Framework. *Information Systems Frontiers*, pages 1–27. [Online First].

Hashmi, M., Governatori, G., and Wynn, M. T. (2012). Business Process Data Compliance. In *Proceedings of 6th International Symposium, RuleML 2012, Montpellier, France*, pages 32–46.

Hashmi, M., Governatori, G., and Wynn, M. T. (2013). Normative Requirements for Business Process Compliance. In *Proceedings of 3rd Symposium (ASSRI’13) on Service Research and Innovation, Sydney, Australia*, pages 100–116.

Hashmi, M., Governatori, G., and Wynn, M. T. (2014). Modeling Obligations with Event-Calculus. In *Proceedings of 8th International Symposium, RuleML 2014, Prague, Czech Republic*, pages 296–310.

Hashmi, M., Governatori, G., and Wynn, M. T. (2015b). Norms Modelling Constructs of Business Process Compliance Management Frameworks: A Conceptual Evaluation. *Enterprise Information Systems Journal*. [Submitted].

Hee, K., Hidders, J., Houben, G.-J., Paredaens, J., and Thiran, P. (2010). On-the-Fly Auditing of Business Processes. In Jensen, K., Donatelli, S., and Koutny, M., editors, *Transactions on Petri Nets and Other Models of Concurrency IV*, volume 6550 of *LNCS*, pages 144–173. Springer.

Herrestad, H. (1991). Norms and Formalization. In *proceedings of ICAIL 1991*, pages 175–184.

Hilty, M., Basin, D. A., and Pretschner, A. (2005). On Obligations. In *ESORICS*, pages 98–117.

Hinge, K., Ghose, A., and Koliadis, G. (2009). Process SEER: A Tool for Semantic Effect Annotation of Business Process Models. In *EDOC '09. IEEE International*, pages 54–63.

HIPAA, T. U. G. (1996). The US Health Insurance Portability and Accountability Act of 1996.

Hoffmann, J., Weber, I., and Governatori, G. (2009). On Compliance Checking for Clausal Constraints in Annotated Process Models. *Information Systems Frontieres*, 14(2):155–177.

IFRS (2014). IFRS 7 International Financial Reporting Standards: Financial Instruments Disclosures.

James, E. and Jonathan, S. (2011). The Benefits of Static Compliance Testing for SCA Next. In *Proceedings of the SDR'11 Technical Conference and Product Exposition*. The Wireless Innovation Forum, Inc.

Jiang, J., Aldewereld, H., Dignum, V., Wang, S., and Baida, Z. (2014). Regulatory Compliance of Business Processes. *AI & SOCIETY*, pages 1–10.

Jiang, J., Virginia, D., Huib, A., Frank, D., and Yao-Hua, T. (2013). Norm compliance checking. In *International conference on Autonomous Agents and Multi-Agent Systems,AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1121–1122.

Johansson, L.-O., Wärja, M., and Carlsson, S. (2012). An Evaluation of Business Process Model Techniques, using Moody's Quality Criterion for a Good Diagram. In *BIR12. CEUR Workshop Proceedings, CEUR-WS.org, vol 963.*

Johnson, C. and Grandison, T. (2007). Compliance with Data Protection Laws using Hippocratic Database Active Enforcement and Auditing. *IBM Systems Journal*, 46(2):255 –264.

Kabilan, V., Johannesson, P., and Rugaimukamu, D. (2003a). Business Contract Obligation Monitoring through Use of Multi-Tier Contract Ontology. In Meersman, R. and Tari, Z., editors, *On The Move (OTM) Workshops to Meaningful Internet Systems*, volume 2889 of *LNCS*, pages 690–702. Springer Verlag Berlin Heidelberg.

Kabilan, V., Johannesson, P., and Rugaimukamu, D. M. (2003b). An Ontological Approach to Unified Contract Management. In *proceedings of 13th European Jananese Conference (EJC) on Information Modelling and Knowlege Bases*, pages 106–110.

Kähmer, M., Gilliot, M., and Müller, G. (2008). Automating Privacy Compliance with ExPDT. In *Proceedings of the 10th IEEE Conference on E-Commerce Technology and 5th Conference on Enterprise Computing*, pages 87 –94.

Karagiannis, D. (2008). A Business Process-Based Modeling Extension for Regulatory Compliance. In *Multikonferenz Wirtschaftsinformatik*.

Karagiannis, D., Mylopoulos, J., and Schwab, M. (2007). Business Process-Based Regulation Compliance: The case of the Sarbanes-Oxley Act. *15th IEEE International Requirements Engineering Conference RE 2007*, pages 315–321.

Kazmierczak, P., Pedersen, T., and Ågotnes, T. (2012). NORMC: A Norm Compliance Temporal Logic Model Checker. In *STAIRS*, volume 241 of *Frontiers in Artificial Intelligence and Applications*, pages 168–179.

Keller, A. and Ludwig, K. (2002). Defining and Monitoring Service-Level Agreements for Dynamic e-Business. In *Proceedings of the 16th USENIX conference on System administration*, pages 189–204, Berkeley, USA. USENIX Association.

Kelsen, H. (1991). *General Theory of Norms*. Clarendon, Oxford.

Kharbili, M. E., Medeiros, A. K. A. D., Stein, S., and van der Aalst, W. (2008). Business Process Compliance Checking:Current State and Future Challenges. In *Modellierung Betrieblicher Informationssyteme, MobIS*, pages 107–113.

Kharbili, M. E. and Stein, S. (2008). Policy-Based Semantic Compliance Checking for Business Process Management. In *MobIS Workshops*, pages 178–192.

Kowalski, R. and Sergot, M. (1989). A Logic-Based Calculus of Events. In Schmidt, J. and Thanos, C., editors, *Foundations of Knowledge Base Management*, Topics in Information Systems, pages 23–55. Springer.

Küster, J. M., Ryndina, K., and Gall, H. (2007). Generation of Business Process Models for Object Life Cycle Compliance. In *Business Prcess Management*, pages 165–181.

Lam, H.-P. and Governatori, G. (2009). The Making of SPINdle. In Governatori, G., Hall, J., and Paschke, A., editors, *Rule Interchange and Applications*, volume 5858 of *Lecture Notes in Computer Science*, pages 315–322. Springer Berlin Heidelberg.

Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *Proceedings of IEEE International Conference on Web Services (ICWS'10)*, pages 369 –376.

Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., and Leymann, F. (2009). Runtime Prediction of Service Level Agreement Violations for Composite Services. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service Oriented Computing co-located with ICSOC 2009*, pages 176–186. Springer,Germany.

Letia, I. A. and Groza, A. (2013). Compliance Checking of Integrated Business Processes. *Data & Knowledge Engineering*, 87(0):1 – 18.

Lin, Y. (2008). *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability*. PhD thesis, Department of Computer and Information Science, Nowegian University of Science and Techology (NTNU).

Liu, Y., Müller, S., and Xu, K. (2007). A Static Compliance-Checking Framework for Business Process Models. *IBM Systems Journal*, 46(2):335–361.

Lohmann, N. (2012). Compliance by Design for Artifact-centric Business Processes. *Information Systems*, 38(4):606–618.

Lomuscio, A., Qu, H., and Solanki, M. (2008). Towards Verifying Contract Regulated Service Composition. In *Proceedings of IEEE International Conference on Web Services (ICWS'08)*, pages 254 –261.

Lu, R. and Sadiq, S. (2007). A Survey of Comparative Business Process Modeling Approaches. In Abramowicz, W., editor, *Business Information Systems*, volume 4439 of *LNCS*, pages 82–94. Springer Berlin / Heidelberg.

Lu, R., Sadiq, S., and Governatori, G. (2007). Compliance Aware Business Process Design. In *3rd International Workshop on Business Process Design (BPD'07)*, pages 120–131. Springer.

Lu, R., Sadiq, S., and Governatori, G. (2008). Measurement of Compliance Distance in Business Processes. *Information Systems Management*, 25(4):344–355.

Ly, L., Rinderle-Ma, S., and Dadam, P. (2010a). Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems. volume 6051, pages 9–23. Springer Berlin Heidelberg.

Ly, L., Rinderle-Ma, S., Knuplesch, D., and Dadam, P. (2011). Monitoring Business Process Compliance Using Compliance Rule Graphs. In Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D., White, J., Hauswirth, M., Hitzler, P., and Mohania, M., editors, *On the Move to Meaningful Internet Systems: OTM 2011*, volume 7044 of *LNCS*, pages 82–99. Springer Berlin Heidelberg.

Ly, L. T., Knuplesch, D., Rinderle-Ma, S., Goeser, K., Reichert, M., and Dadam, P. (2010b). SeaFlows Toolset - Compliance Verification Made Easy. In *CAiSE'10 Demos*.

Ly, L. T., Maggi, F. M., Montali, M., Rinderle, S., and van der Aalst, W. (2013). A Framework for the Systematic Comparison and Evaluation of Compliance Monitoring Approaches. In *Proceeding of EDOC*.

Ly, L. T., Rinderle-Ma, S., Göser, K., and Dadam, P. (2012). On Enabling Integrated Process Compliance with Semantic Constraints in Process Management Systems. *Information Systems Frontiers*, 14(2):195–219.

Maggi, F., Montali, M., Westergaard, M., and van der Aalst, W. (2011a). Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *BPM*, LNCS 6896, pages 132–147. Springer.

Maggi, F., Westergaard, M., Montali, M., and van der Aalst, W. (2011b). Runtime Verification of LTL-Based Declarative Process Models. In *Proc. of RV*, LNCS. Springer-Verlag.

Makinson, D. and van der Torre, L. (2003). Permission from an input/output perspective. *J. Philosophical Logic*, 32(4):391–416.

Marín, R. H. and Sartor, G. (1999). Time and Norms: A Formalisation in the Event-Calculus. In *ICAIL*, pages 90–99.

MASTER (2008). Managing Assurance, Security, and Trust for Services. FP7-ICT Integrated Project for Secure, Dependable, and Trusted Infrastructures.

Mateescu, R. and Sighireanu, M. (2003). Efficient On-the-Fly Model-Checking Ror regular Alternation-Free Mu-Calculus. *Science of Computer Programming*, 46(3):255 – 281. Special issue on Formal Methods for Industrial Critical Systems.

McIntyre, S. R. (2008). Integrated Governance, Risk and Compliance: Improve Performance and Enhance Productivity in Federal Agencies. Technical report, PricewaterhouseCoopers.

Mili, H., Tremblay, G., Jaoude, G. B., Lefebvre, E., Elabed, L., and Boussaidi, G. E. (2010). Business Process Modeling Languages:Sorting Through the Alphabet Soup. *ACM Computing Surveys*, 43(1):1–56.

Miller, R. and Shanahan, M. (1999). The Event Calculus in Classical Logic - Alternative Axiomatisations. *Electron. Trans. Artif. Intell.*, 3(A):77–105.

Miller, R. and Shanahan, M. (2002). Some Alternative Formulations of the Event-Calculus. In Kakas, A. and Sadri, F., editors, *Computational Logic: Logic Programming and Beyond*, volume 2408 of *LNCS*, pages 452–490. Springer.

Milosevic, Z. (2005). Towards Integrating Business Policies with Business Processes. *Journal of Business Process Management*, 3649:404 – 409.

Milosevic, Z., Jösang, A., Dimitrakos, T., and Patton, M. A. (2002). Discretionary Enforcement of Electronic Contracts. In *Proceedings of the 6th International Enterprise Distributed Object Computing Conference (EDOC'02)*, EDOC '02, pages 39–, Washington, DC, USA. IEEE Computer Society.

Milosevic, Z., Sadiq, S., and Orlowska, M. (2006a). Towards a Methodology for Deriving Contract-Compliant Business Processes. In Dustdar, S., Fiadeiro, J., and Sheth, A., editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 395–400. Springer Berlin / Heidelberg.

Milosevic, Z., Sadiq, S., and Orlowska, M. (2006b). Translating Business Contract into Compliant Business Processes. In *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, pages 211 –220.

Montali, M. (2010). *Specification and Verification of Declarative Open Interaction Models: A logic-Based Approach*, volume 56 of *LNBIP*. Springer-Verlag Berlin Heidelberg.

Montali, M., Pesic, M., Aalst, W. M. P. v. d., Chesani, F., Mello, P., and Storari, S. (2010). Declarative Specification and Verification of Service Choreographiess. *ACM Trans. Web*, 4(1):3:1–3:62.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580.

Namiri, K. and Stojanovic, N. (2007a). Pattern-Based Design and Validation of Business Process Compliance. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems*, pages 59–76, Berlin, Heidelberg. Springer-Verlag.

Namiri, K. and Stojanovic, N. (2007b). Using Control Patterns in Business Processes Compliance. In *Proceedings of the 2007 international conference on Web information systems engineering*, (WISE'07), pages 178–190, Berlin, Heidelberg. Springer-Verlag.

Namiri, K. and Stojanovic, N. (2008a). Towards a Formal Framework for Business Process Compliance. In *Multikonferenz Wirtschaftsinformatik'08*, pages –1–1.

Namiri, K. and Stojanovic, N. (2008b). Towards A Formal Framework for Business Process Compliance. In *Proceedings of Multikonferenz Wirtschaftsinformatik, MKWI 2008, München*.

Nishizaki, S.-y. and Ohata, T. (2013). Real-Time Model Checking for Regulatory Compliance. In Das, V. and Chaba, Y., editors, *Mobile Communication and Power*

Engineering, volume 296 of *Communications in Computer and Information Science*, pages 70–77. Springer Berlin Heidelberg.

Nute, D. (2003). Defeasible Logic. In Bartenstein, O., Geske, U., Hannebauer, M., and Yoshie, O., editors, *Web Knowledge Management and Decision Support*, volume 2543 of *LNCS*, pages 151–169. Springer Berlin Heidelberg.

OCEG (2012). Governance,Risk and Compliance (GRC) Capability Model, version 2.1.

Ochsenschläger, P., Repp, J., Rieke, R., and Nitsche, U. (1998). The SH-Verification Tool - Abstraction-based Verification of Co-operating Systems. *The Journal of Formal Aspects of Computing*, 10(4):381–404.

Olivieri, F. (2014). *Compliance by Design. Synthesis of Business Processes by Declarative Specifications.* Phd, Dipartimento di Informatica, Università digli Studi di Verona, Italy and Institute for Integrated and Intelligent Systems, Griffith University, Australia.

OMG (2010). Business Process Model Notation (BPMN), Version 2.0. Standard.

OMG (2011). Unified modeing language (uml 2.0).

Oro, E. and Ruffolo, M. (2012). *Advances in Knowledge Representation*, chapter A Knowledge Representation Formalism for Semantic Business Process Management. InTech Europe.

Otto, P. and Anton, A. (2007). Addressing Legal Requirements in Requirements Engineering. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 5–14.

Ouyang, C., Dumas, M., Breutel, S., and ter Hofstede, A. H. M. (2006). Translating Standard Process Models to BPEL. In *Proceedings of Advanced Information Systems Engineering, 18th International Conference (CAiSE 2006), June 5-9,Luxembourg*, pages 417–432.

Ouyang, C., Dumas, M., van der Aalst, W., ter Hofstede, A. H. M., and Mendling, J. (2009). From Business Process Models to Process-oriented Software Systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1).

Palmirani, M., Governatori, G., and Contissa, G. (2011). Modelling Temporal Legal Rules. In *ICAIL*, pages 131–135.

Paschke, A. and Bichler, M. (2005). SLA Representation, Management and Enforcement. In *proceedings of The IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE'05*, pages 158–163.

Pattersson, P. and Larson, K. (2000). UPPAAL 2K. *Bulletin of the European Association of Theoratical Computer Science*, 70:40–44.

Pershkow, B. (2002). Sarbanes-Oxley: Investment Company Compliance. *Journal of Investment Compliance*, 3(4):16 – 30.

Pesic, M., Schonenberg, H., and van der Aalst, W. (2007). DECLARE: Full Support for Loosely-Structured Processes. In *Procedings of 11th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'07)*, pages 287–287.

Pesic, M. and van der Aalst, W. (2006). A Declarative Approach for Flexible Business Processes Management. In *BPM Workshops*, volume 4103 of *LNCS*, pages 169–180. Springer.

Pnueli, A. (1977). The Temporal Logic of Programs*. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57.

Prakken, H. and Sartor, G. (1996). A Dialectical Model of Assessing Conflicting Arguments in Legal Reasoning. *Artificial Intelligence and Law*, 4(3-4):331–368.

Ramezani, E., Fahland, D., and van der Aalst, W. (2012a). Where Did I Misbehave? Diagnostic Information in Compliance Checking. In *Proceedings of Business Process Management*, pages 262–278.

Ramezani, E., Fahland, D., van der Werf, J., and Mattheis, P. (2012b). Separating Compliance Management and Business Process Management. In Daniel, F., Barkaoui, K., and Dustdar, S., editors, *Business Process Management Workshops*, volume 100 of *LNBIP*, pages 459–464. Springer Berlin Heidelberg.

Ramezani, E., Fahland, D., van Dongen, B. F., and van der Aalst, W. (2013). Diagnostic Information for Compliance Checking of Temporal Compliance Requirements. In *CAiSE*, pages 304–320.

Rangan, R. M., Rohde, S. M., Peak, R., Chadha, B., and Bliznakov, P. (2005). Streamlining Product Lifecycle Processes: A Survey of Product Lifecycle Management Implementations, Directions, and Challenges. *Journal of Computing and Information Science in Engineering*, 5(3):227–237.

Rieke, R., Repp, J., Zhdanova, M., and Eichler, J. (2014). Monitoring Security Compliance of Critical Processes. In *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'14)*, pages 552–560.

Rifaut, A. and Dubois, E. (2008). Using Goal-Oriented Requirements Engineering for Improving the Quality of ISO/IEC 15504 based Compliance Assessment Frameworks. In *16th IEEE International Requirements Engineering Conference (RE '08).*, pages 33 –42.

Rosemann, M. and zur Muehlen, M. (2005). Integrating Risks in Business Process Models. In *Proceedings of the 16th Australasian Conference on Information Systems (ACIS'05)*.

Rubino, R., Rotolo, A., and Sartor, G. (2006). An OWL Ontology of Fundamental Legal Concepts. In *Legal Knowledge and Information Systems - JURIX 2006, Paris, France, 7-9*, pages 101–110.

Sadiq, S. and Governatori, G. (2010). Managing Regulatory Compliance in Business Processes. In *Handbook of Business Process Management*, volume 2, pages 157–173. Springer.

Sadiq, S., Governatori, G., and Namiri, K. (2007). Modeling Control Objectives for Business Process Compliance. In *Proceedings of BPM'07*, pages 149–164. Springer.

Sadiq, S., Orlowska, M., Sadiq, W., and Foulger, C. (2004). Data Flow and Validation in Workflow Modelling. In *Proceedings of the 15th Australasian database conference - Volume 27*, pages 207–214, Darlinghurst, Australia.

Sadri, F. and Kowalski, R. (1995). Variants of the Event Calculus. In Sterling, L., editor, *Proceedings of the Twelth International Conference on Logic Programming*. MIT Cambrdige.

Sapkota, K., Aldea, A., Duce, D. A., Younas, M., and Bañares Alcántara, R. (2011). Towards Semantic Methodologies for Automatic Regulatory Compliance Support.

In *Proceedings of the 4th workshop for Ph.D. students in Information and Knowledge management*, (PIKM '11), pages 83–86, New York,USA.

Sartor, G. (2005). *Legal Reasoning: A Cognitive Approach to the Law*. Springer.

SCBS (2004). BASEL II Accord.

Schleicher, D., Anstett, T., Leymann, F., and Mietzner, R. (2009). Maintaining Compliance in Customizable Process Models. In Meersman, R., Dillon, T., and Herrero, P., editors, *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *LNCS*, pages 60–75. Springer Heidelberg.

Schleicher, D., Anstett, T., Leymann, F., and Schumm, D. (2010). Compliant Business Process Design Using Refinement Layers. In *OTM Conferences (1)*, pages 114–131.

Schmidt, R., Bartsch, C., and Oberhauser, R. (2007). Ontology-based Representation of Compliance Requirements for Service Processes. In *Proceedings of Workshop on Semantic Business Porocess and Product Lifecycle Management (SBPM'07)*, pages 28–39.

Schumm, D., Turetken, O., Kokash, N., Elgammal, A., Leymann, F., and Heuvel, W.-J. V. D. (2010). Business Process Compliance through Reusable Units of Compliant Processes. In *Proceedings of International Conference on Current Trends in Web Engineering*.

Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge, MA.

Strecker, S., Heise, D., and Frank, U. (2011). RiskM: A Multi-perspective Modeling Method for IT Risk Assessment. *Information Systems Frontiers*, 13(4):595–611.

Thomason, R. H. (1981). Deontic Logic Founded on Tense Logic. In Hilpinen, R., editor, *New Studies on Deontic Logic*, pages 165–176. Kluwer.

Türetken, O., Elgammal, A., van den Heuvel, W.-J., and Papazoglou, M. (2011). Enforcing Compliance on Business Processes Through the use of Patterns. In *Proceeding of Eurpoean Conference on Information System*.

Túretken, O., Elgammal, A., van den Heuvel, W.-J., and Papazoglou, M. (2012). Capturing compliance requirements: A pattern-based approach. *Software, IEEE*, 29(3):28 –36.

Turki, S. and Bjekovic-Obradovic, M. (2010). Compliance in e-Government Service Engineering: State-of-the-Art. In *Exploring Services Science*, LNBIP, pages 270–275. Springer.

US-Government (2002). Public Company Accountng Reforms and Investor Protection Act (Sarbanes-Oxley Act). Public Law 107-204, 116 Stat. 745.

van der Aalst, W. (1997). Verification of Workflow Nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, London, UK. Springer-Verlag.

van der Aalst, W. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

van der Aalst, W. (2000). Workflow Verification: Finding Control-Flow Errors Using Petri-Net-based Techniques. In van der Aalst, W. M. P., Desel, J., and Oberweis, A., editors, *Business Process Management: Models, Techniques, and Empirical Studies.*

van der Aalst, W. (2009). *Business Process Management, Encyclopedia of Database Systems.* Springer.

van der Aalst, W., Adriansyah, A., and van Dongen, B. (2012). Replaying History on Process Models for Conformance Checking and Performance Analysis. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 2(2):182–192.

van der Aalst, W., de Beer, H. T., and van Dongen, B. F. (2005). Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In Robert Meersman, Z. T., editor, *On the Move to Meaningful Internet Systems*, volume 3760 of *LNCS*, pages 130–147. Springer-Verlag.

van der Aalst, W., Pesic, M., and Schonenberg, H. (2009). Declarative Workflows: Balancing Between Flexibility and Support. 23:99–113.

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., and Barros., A. (2002). Workflow Patterns. QUT Technical report. FIT-TR-2002-02, Queensland University of Technology, Brisbane, Australia.

van der Aalst, W., van Hee, K., van der Werf, J. M., Kumar, A., and Verdonk, M. (2011). Conceptual model for online auditing. *Decision Support Systems*, 50(3):636 – 647. On quantitative methods for detection of financial fraud.

van der Aalst, W., van Hee, K. M., van Werf, J. M., and Verdonk, M. (2010). Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *Computer*, 43(3):90 –93.

Vicente, P. and Mira da Silva, M. (2011). A Conceptual Model for Integrated Governance, Risk and Compliance. In Mouratidis, H. and Rolland, C., editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin Heidelberg.

von Wright, G. H. (1963). *Norm and Action*. Routledge, London.

Wang, Z., ter Hofstede, A. H., Ouyang, C., Wynn, M., Wang, J., and Zhu, X. (2014). How to Guarantee Compliance Between Workflows and Product Lifecycles? *Information Systems*.

Ward, M. (1995). *Physical Electrochemistry: Principles, Methods and Applications*, chapter Principles and Applications of Electrochemical Quartz Crystal Microbalance, pages 293–338. Marcel Dekker, Inc., New York.

Weigand, H., van den Heuvel, W.-J., and Hiel, M. (2011). Business Policy Compliance in Service-Oriented Systems. *Information Systems*, 36(4):791 – 807.

Wen, L., Wang, J., van der Aalst, W., Huang, B., and Sun, J. (2010). Mining Process Models with Prime Invisible Tasks. *Data Knowl. Eng.*, 69(10):999–1021.

Wolter, C., Miseldine, P., and Meinel, C. (2009). Verification of Business Process Entailment Constraints Using SPIN. In Massacci, F., Redwine, S., and Zannone, N., editors, *Engineering Secure Software and Systems*, volume 5429 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg.

Wyner, A. Z. (2008). *Violations and Fulfillments in the Formal Representation of Contracts*. PhD thesis, School of Physical Sciences and Engineering.

Yip, F., Parameswaran, N., and Ray, P. (2007). Rules and Ontology in Compliance Management. In *Proceedings of (EDOC'07)*, pages 435–, Washington DC, USA.

Yolum, P. and Singh, M. P. (2002). Flexible Protocol Specification and Execution: Applying Event-Calculus Planning using Commitments. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, AAMAS '02, pages 527–534.

Yolum, P. and Singh, M. P. (2004). Reasoning about Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253.

Yu, J., Han, Y.-B., Han, J., Jin, Y., Falcarin, P., and Morisio, M. (2008). Synthesizing Service Composition Models on the Basis of Temporal Business Rules. *Journal of Computer Science and Technology*, 23:885–894.

Yu, J., Manh, T., Han, J., Jin, Y., Han, Y., and Wang, J. (2006). Pattern Based Property Specification and Verification for Service Composition. In Aberer, K., Peng, Z., Rundensteiner, E., Zhang, Y., and Li, X., editors, *Web Information Systems WISE 2006*, volume 4255 of *LNCS*, pages 156–168. Springer Berlin Heidelberg.