A Conceptually Rich Model of Business Process Compliance

Guido Governatori *   **Antonino Rotolo** **

* NICTA, Australia PO Box 6020, St Lucia, Queensland 4067, Australia
** CIRSFID and Law Faculty, University of Bologna, Via Galliera 3, 40122, Bologna, Italy
Email: `guido.governatori@nicta.com.au, anotnino.rotolo@unibo.it`

Abstract

In this paper we extend the preliminary work developed elsewhere and investigate how to characterise many aspects of the compliance problem in business process modeling. We first define a formal and conceptually rich language able to represent, and reason about, chains of reparational obligations of various types. Second, we devise a mechanism for normalising a system of legal norms. Third, we specify a suitable language for business process modeling able to automate and optimise business procedures and to embed normative constraints. Fourth, we develop an algorithm for compliance checking and discuss some computational issues regarding the possibility of checking compliance runtime or of enforcing it at design time.

Keywords: Business Process, Regulatory Compliance, Obligations

1 Introduction

1.1 Background

Business processes specify the activities a business does to achieve its business objectives. Businesses are typically regulated. The requirements businesses have to comply with may stem from legislation and regulatory bodies, standards and codes of practice, and also business partner contracts. *Business Process Compliance* is the relationship between two sets of specifications: the specification for the processes/procedures adopted by a business to achieve its goal, and the specifications corresponding to the regulations relevant for a business. Essentially, compliance is aimed at ensuring that business processes, operations and practise are in accordance with a prescribed and/or agreed set of norms. Compliance requirements may stem from legislation and regulatory bodies (e.g., Sarbanes-Oxley, Basel II, HIPAA), standards and codes of practice (e.g., SCOR, ISO9000) and also business partner contracts. To align the two sets of specification a fundamental requirements is to be able to represent formally the two sets of specifications. Accordingly, any attempt to provide a formal framework to investigate whether processes comply with relevant regulation must provide a conceptually sound formalisation of the norms and the obligations process are subject to.

In this paper we extend the preliminary work developed in (Governatori & Rotolo 2008a) and further investigate how to model compliance in business processes. In (Governatori & Rotolo 2008a) we proposed (1) a method to align the language specifying the activities of a business process and the conditions set up by the norms relevant for the process and (2) an efficient algorithm to determine whether a process is compliant. The method was based on (semantic) annotations, where the annotations are written in the formal language chosen to represent the normative specifications. The idea was that business processes are annotated and the annotations provide the conditions a process has to comply with. Annotations can be at different levels; for example we can annotate a full process or a single task in a process. In addition we can have different types of annotation. Annotations can range from the full set of rules (norms) specific to a process or a single task to simple semantic annotation corresponding to one effect of a particular task (e.g., after the successful execution of task A in a process B the value of the environment variable C is D).

Checking compliance amounts to a relatively affordable operation when we have to see whether processes are compliant with respect to simple normative systems. But things are tremendously harder when we deal with processes to be tested against complex, large and articulated systems of norms such as bodies of legal provisions.

In (Governatori & Rotolo 2008a) we suggested a first solution to overcome the difficulties arising when the violation of a legal obligation activates other obligations able to compensate for this violation. In particular, we proposed to use the logic originally developed in (Governatori & Rotolo 2006). This paper adopts the same methodology, but it generalises (Governatori & Rotolo 2008a)'s approach to reparational obligations and it addresses other two significant sources of complexities. As we will see, the proposed framework will be helpful to address and analyse some still unsolved research issues in business process modeling. The following subsection summarises the complexities which we will discuss in this paper.

1.2 Motivation

A *first source of complexities* was already (partially) addressed in (Governatori & Rotolo 2008a). It resides in the fact that legal norms regulate processes by usually specifying actions to be taken in case of breaches of some of the norms, actions which can vary from (pecuniary) penalties to the termination of an interaction itself. These constructions, i.e., obligations in force after some other obligations have been violated, are known in the deontic literature as contrary-to-duty obligations (CTDs) or reparational obligations (because they are meant to 'repair' or 'compensate' violations of primary obligations (Carmo & Jones 2002)). Thus a CTD is a conditional obligation arising in response to a violation, where a violation is signalled by an unfulfilled obligation. These constructions identify situations that are not ideal for the interaction but still acceptable. The ability to deal with violations and the reparational obligations generated from them is an essential requirement for processes where, due to the nature of the environment where they are deployed, some failures can occur, but it does not necessarily mean that the

whole interaction has to fail. However, the main problem with these constructions is that they can give rise to very complex rule dependencies, because we can have that the violation of a single rule can activate other (reparational) rules, which in turn, in case of their violation, refer to other rules, and so forth.

A *second source of complexities* depends on the fact that processes may be regulated by different types of obligations (see Section 2.1). In (Governatori et al. 2007) we proposed a classification of obligations, according to which different obligations may require distinct compliance conditions. We may have obligations requiring (1) to be always fulfilled during the execution of the entire process or of some subpaths of it, (2) that a certain condition must occur at least once before the execution of a certain task *A* of the process and such that the obligations may, or may not, persist after *A* if they are not complied with, (3) that something be done in a single task. Clearly, the peculiarities of these types of obligation make things more complex when we deal with the compliance of a process with respect to chains of reparational obligations. For example, if the primary obligation is persistent and states to pay before task A, and the secondary (reparational) obligation is to pay a fine in the task B successive to A, the process is compliant not only when we pay before A, but also when we do not meet this deadline, pay later and pay the fine at B. If the secondary obligation rather requires to be always fulfilled during the execution of all tasks successive to A, compliance conditions will change. In addition, other types of obligation can be considered: for instance, we may have provisions stating that some *A* is obligatory and which are fulfilled even if *A* was obtained before the provision was in force, whereas other provisions state that *A* is obligatory but they are complied with only when *A* holds after they are in force.

A *third source of complexities* depends on whether our aim is to check compliance at *runtime* or to model and enforce it at *design time*. Indeed, ensuring automated detection of compliance of business processes with normative documents is a complex problem, involving a number of alternatives.

Currently there are two main approaches towards achieving runtime compliance[1]: *auditing* and *monitoring*. Both methods detect compliance by means of *retrospective reporting*. *Auditing* is conducted for "after-the-fact" detection. Much of the existing software solutions for compliance follow this approach. The proposed solutions hook into variety of enterprise system components (e.g. SAP HR, LDAP Directory, Groupware etc.) and generate audit reports against hard-coded checks performed on the requisite system. These solutions often specialise in certain class of checks, for example the widely supported checks that relate to Segregation of Duty violations in role management and user provisioning systems. This approach resides in the space of "after-the-fact" detection, as compliance is checked at the end of the process execution. *Monitoring* is based on the same principle, but is performed by checking compliance step by step (after each task is executed, for example).

The main limit of monitoring and auditing is that they do not offer any guarantee to correctly fix the processes when they are not compliant. Such methods are basically able to answer the question of whether a given process fulfilled all regulations applying to it. If the process is not compliant we would rather need a method which provides a means of visualizing the impact of compliance controls on process models, assist in compliance checking and analysis and also feedback for subsequent (re)design of the process models.

Hence, a sustainable approach for achieving compliance should fundamentally have a *preventative* focus. As we will see, we describe an approach that provides the capability to capture compliance requirements through a generic requirements modelling framework, and subsequently facilitate the propagation of these requirements into business process models and enterprise applications, thus achieving *compliance by design*.[2]

However, we will see that, when all types of obligations mentioned above are used to regulate business processes, the enforcement of compliance at design time can rise serious computational problems.

1.3 Layout of the Paper

Ensuring automated detection and/or enforcement of compliance requires to address the following related research tasks:

(1) Define a formal language able to represent, and reason about, chains of reparational obligations of the types recalled above; we address this in Section 2.2;
(2) Devise a mechanism for normalising a system of legal norms, namely, identify formal loopholes, deadlocks and inconsistencies in it, and to make hidden conditions explicit; without this task, we do not have any guarantee that a given process is compliant, because we do not know if all relevant norms have been considered; this task will be addressed in Section 2.3;
(3) Specify a suitable language for business process modeling (in our case based on annotations) able to automate and optimise business procedures and to embed normative constraints; this task will be addressed in Section 3;
(4) Develop a (possibly efficient) algorithm for compliance checking; the procedure will also require to develop a reasoning mechanism able to establish what chains of reparational obligations are active for each step in the process; this task will be addressed is Section 4.

A further section (Section 5) will be devoted to discussing the problem of computing runtime or design time compliance when different types of obligations regulate a given process. A summary and related work will conclude the paper.

2 Normative Constraints

2.1 Violations and Types of Obligations

Achievement, Maintenance and Punctual Obligations
We can distinguish *achievement obligations* from *maintenance obligations* (Governatori et al. 2007).

For an *achievement obligation*, a certain condition must occur at least once before the deadline:

Example 1 *Customers must pay before the delivery of the good, after receiving the invoice*

In this example, the deadline (before the delivery of the good) refers to an obligation triggered by receipt of the invoice: such an obligation is persistent. After that the customer is obliged to pay. The obligation terminates only when it is complied with. Note that, in this example, the obligation itself obviously persists after the deadline, until it is achieved. But we may have cases where achievement obligations do not persist after the deadline:

Example 2 *Once the submissions to APCCM are made available to APCCM PC members, the reviewers must send their reports before the notifications are delivered to the authors*

Indeed, the obligation to deliver a review does not persist after the deadline, since after the review result has been notified to the authors, the paper has been accepted or rejected on the basis of the other reports delivered in time.

---

[1]For a comprehensive exposition of compliance for business process models, see (Governatori & Sadiq 2009, Sadiq & Governatori 2009).

[2]See also (Lu et al. 2007) for the compliance by design methodology.

In general, a deadline signals that a violation of the obligation has occurred. This may trigger an explicit sanction (see below).

For *maintenance obligations*, a certain condition must obtain during all instants before the deadline:

Example 3 *After opening a bank account, customers must keep a positive balance until bank charges are taken out.*

By definition, maintenance obligations do not persist after the deadline. In Example 3, the deadline only signals that the obligation is terminated. A violation occurs when the obliged state does not obtain at some point before the deadline.

Finally, we may have *punctual obligations*, which only apply to single tasks or instants:

Example 4 *When banks proceed with any wire transfer, they must transmit a message, via SWIFT, to the receiving bank requesting that the payment is made according to the instructions given.*

Many norms can be associated with an explicit sanction. Consider, for instance, the obligations in Examples 1, 2, and 3:

Example 5 *Customers must pay before the delivery of the good, after receiving the invoice. Otherwise, an additional fine must be paid.*

Example 6 *After opening a bank account, customers must keep a positive balance until bank charges are taken out. Otherwise, their account is blocked.*

Example 7 *Once the submissions to APCCM are made available to APCCM PC members, the reviewers must send their reports before the notifications are delivered to the authors. Otherwise, they will be blacklisted for inclusions in future APCCM PCs.*

An explicit sanction is often implemented through a separate obligation, which is triggered by a detected violation. In this setting, legislators may need further deadlines to enforce the sanctions, leading to a chain of obligations. For instance, the payment of a fine mentioned in Example 5 could be due before the execution of a subsequent task.

Preemptive or Non-preemptive Obligations We can also distinguish *preemptive obligations* from *non-preemptive obligations*.

Consider again the obligation in Example 1 and suppose that the price to be paid by a customer is 200\$. Suppose now that this customer, by mistake, had transferred an amount of 200\$ to the bank account of the seller before the delivery of the invoice. In this case, the early transfer may count as a payment and the customer could claim that her obligation to pay the seller is already fulfilled. This is an example of *preemptive* obligation.

Non-preemptive obligations do not work as above. Consider this example:

Example 8 *Executors and administrators of a decedent's estate will be required to give notice to each beneficiary named in the Will within 60 days after the date X of an order admitting a will to probate has been signed.*

If an executor gives a notice to the beneficiaries before X, she will not comply with the above obligation and will have to resend the notification after that. Note that, in general, the distinction between preemptive and non-preemptive obligations applies only to achievement obligations, while it does not make sense with the maintenance and punctual ones.

Violations The expression of violation conditions and the reparation obligations is an important requirement for designing subsequent processes to minimise or deal with such violations and also to determine the compliance of a process with the relevant norms. The violation expression consists of the primary obligation, its violation conditions, an obligation generated upon the violation condition occurs, and this can recursively be iterated, until the final condition is reached. We introduce the non-boolean connective $\otimes$, whose interpretation is such that $OA \otimes OB$ is read as "*OB* is the reparation of the violation of *OA*". In other words, the interpretation of $OA \otimes OB$, is that *A* is obligatory, but if the obligation *OA* is not fulfilled (i.e., when $\neg A$ is the case), then the obligation *OB* is activated and becomes in force until it is satisfied or violated. In the latter case a new obligation may be activated, followed by others in chain, as appropriate.

However, the violation condition of an obligation varies depending on whether it is an achievement or a maintenance obligation, or a preemptive or a non-preemptive one. In the next section, we will extend the approach of (Governatori & Rotolo 2008*a*, 2006) to cover these cases.

2.2 Process Compliance Language (PCL)

A conceptually sound formalisation of norms (for assessing the compliance of a process) should take into account all the aspects mentioned in Section 2.1. Thus, we now provide here a formal account of the ideas presented above. Our formalism, called Process Compliance Language (PCL), is a combination of an efficient non-monotonic formalism (Defeasible Logic (Antoniou et al. 2001)) and a deontic logic of violations (Governatori & Rotolo 2006). The current version of PCL significantly extends the logic of (Governatori & Rotolo 2008*a*) by working on all types of obligations discussed in Section 2.1. Hence, this particular combination allows us to represent exceptions as well as the ability to capture violations and any types of obligations resulting from the violations; in addition our framework has good computational properties: the extension of a theory (i.e., the set of conclusions/normative positions following from a set of facts) can be computed in time linear to the size of the theory.

The ability to handle violation is very important for compliance of agents' processes. Often businesses operate in dynamic and somehow unpredictable environments. As a consequence in some cases, maybe due to external circumstances, it is not possible to operate in the way specified by the norms, but the norms prescribe how to recover from the resulting violations. In other cases, the prescribed behaviours are subject to exceptions. Finally, in other cases, one might not have a complete description of the environment. Accordingly the process has to operate based on the available input (this is typically the case of the *due diligence* prescription), but if more information were available, then the task to be performed could be a different one.

PCL is sound in this respect given the combinations of the deontic component (able to represent the fundamental normative positions and chains of violations/reparations) and the defeasible component that takes care of the issue about partial information and possibly conflicting prescriptions.

PCL formal language consists of the following set of atomic symbols: a numerable set of propositional letters $p, q, r, \ldots$, intended to represent the state variables and the tasks of a process. Formulas of the logic are constructed using the negation $\neg$, the non-boolean connective $\otimes$ (for the Contrary-To-Duty (CTD) operator), and the deontic operators O_y^x (for obligation), where y can be empty. The deontic operators have subscripts and superscripts to specify whether they denote achievement, maintenance or punctual obligations, or whether they are or not preemptive. Table 1 summarises all possible combinations. The formulas of PCL will be constructed in two steps according to the following formation rules:

- every propositional letter is a literal;

Obligation operators	Intuitive reading
$O_{pr}^{a,\pi}$	achievement, persistent, preemptive
$O_{n-pr}^{a,\pi}$	achievement, persistent, non-preemptive
$O_{pr}^{a,\tau}$	achievement, non-persistent, preemptive
$O_{n-pr}^{a,\tau}$	achievement, non-persistent, non-preemptive
O^m	maintenance
O^p	punctual

Table 1: Types of obligations

- the negation of a literal is a literal;
- if X is a deontic operator and l is a literal then Xl and $\neg Xl$ are deontic literals.

After we have defined the notions of literal and deontic literal we can use the following set of formation rules to introduce $\otimes$-expressions, i.e., the formulas used to encode chains of obligations and violations.

- every deontic literal is an $\otimes$-expression;
- if $Xl_1, \ldots, Xl_n$ are deontic literals, then $Xl_1 \otimes \ldots \otimes Xl_n$ is an $\otimes$-expression.

The connective $\otimes$ permits combining primary and CTD obligations into unique regulations. The meaning of an expression like $O_{pr}^{a,\pi}A \otimes O^pB \otimes O^mC$ is that the primary provision is an achievement, persistent, preemptive obligation to do A, but if A is not done, then we have a punctual obligation to do B. If B fails to be realised, then we obtain a maintenance obligation to do C. Thus B is the reparation of the violation of the obligation $O_{pr}^{a,\pi}A$. Similarly C is the reparation of the obligation O^pB, which is in force when the violation of A occurs.

Each norm is represented by a rule in PCL, where a rule is an expression $r : A_1, \ldots, A_n \Rightarrow C$, where r is the name/id of the norm, $A_1, \ldots, A_n$, the *antecedent* of the rule, is the set of the premises of the rule (alternatively it can be understood as the conjunction of all the literals in it) and C is the conclusion of the rule. Each A_i is either a literal or a deontic literal and C is an $\otimes$-expression.

The meaning of a rule is that the normative position (obligation, permission, prohibition) represented by the conclusion of the rule is in force when all the premises of the rule hold. PCL is also equipped with another type of rules, called defeaters (marked with arrow $\rightsquigarrow$) and a superiority relation (a binary relation) over the rule set.

In Defeasible Logic, the superiority relation ($\prec$) determines the relative strength of two rules, and it is used when rules have potentially conflicting conclusions. For example, given the rules $r_1 : a \Rightarrow O^mb \otimes O_{n-pr}^{a,\pi}c$ and $r_2 : d \Rightarrow \neg O_{pr}^{a,\pi}c$, $r_1 \prec r_2$ means that rule r_1 prevails over rule r_2 in situations where both fire and they are in conflict.

Defeaters play in Defeasible Logic a peculiar role, as they cannot lead to any conclusion but are used to defeat some rules by producing evidence to the contrary. In this sense, defeaters are suitable to model the termination of the persistence of obligations (Governatori et al. 2005, Governatori & Rotolo 2008b). Consider Example 5 (we assume that the reparational obligation is a punctual one):

$$\text{inv}_{init} \quad \text{invoice} \Rightarrow O_{pr}^{a,\pi}\text{pay} \otimes O^p\text{pay\_fine}$$
$$\text{inv}_{term} \quad \text{pay} \rightsquigarrow \neg O_{pr}^{a,\pi}\text{pay}$$

Here, compliance is the only condition that terminates the obligation to pay: the obligation in fact persists beyond the deadline.

Example 6 is modeled as follows:

$$\text{pos}_{init} \quad \text{open\_account} \Rightarrow O^m\text{positive} \otimes O^p\text{blocked}$$
$$\text{pos}_{term} \quad \text{bank\_charges} \rightsquigarrow \neg O^m\text{positive}$$

On account of the nature of maintenance obligations, the termination of the primary obligation occurs only when bank charges are taken out.

Finally, the termination of the primary obligation in Example 7 is captured as follows:

$$\text{rev}_{init} \quad \text{papers\_available} \Rightarrow O_{n-pr}^{a,\tau}\text{review} \otimes O_{pr}^{a,\pi}\text{blacklist}$$
$$\text{rev}_{term_1} \quad \text{notification} \rightsquigarrow \neg O_{n-pr}^{a,\tau}\text{review}$$
$$\text{rev}_{term_2} \quad \text{review} \rightsquigarrow \neg O_{n-pr}^{a,\tau}\text{review}$$

Note that we have two termination rules: one stating that the obligation to send the review no longer holds when the reports are notified to the authors, another establishing that such an obligation is terminated if it is complied with.

2.3 Normal Forms

We introduce transformations of an PCL representation of a normative system to produce a normal form of the same (NPCL). A normal form is a representation of a normative system based on an PCL specification containing all conditions that can be generated/derived from the given PCL specification. The purpose of a normal form is to "clean up" the PCL representation of a normative system, that is to identify formal loopholes, deadlocks and inconsistencies in it, and to make hidden conditions explicit.

In the rest of this section we introduce the procedures to generate normal forms. First (Section 2.3.1) we describe a mechanism, based on (Governatori & Rotolo 2006), to derive new conditions by merging together existing normative clauses. In particular we link an obligation and the obligations triggered in response to violations of the obligation. Then, in Section 2.3.2, we examine the problem of redundancies, and we give a condition to identify and remove redundancies from the formal normative specification. Section 2.3.3 discusses how to solve possible conflicts between deontic provisions.

2.3.1 Merging Norms

One of the features of the logic of violations is to take two rules, or norms, and merge them into a new clause.

Consider a norm like (Γ and Δ are sets of premises)

$$\Gamma \Rightarrow O^mA.$$

Given an obligation like this, if we have that the violation of O^mA is part of the premises of another norm, for example,

$$\Delta, \neg A \Rightarrow O^pC,$$

then the latter must be a good candidate as reparational obligation of the former. This idea is formalised as follows:

$$\frac{\Gamma \Rightarrow O^mA \qquad \Delta, \neg A \Rightarrow O^pC}{\Gamma, \Delta \Rightarrow O^mA \otimes O^pC}$$

This reads as follows: given two policies such that one is a conditional obligation ($\Gamma \Rightarrow O^mA$) and the antecedent of second contains the negation of the propositional content of the consequent of the first ($\Delta, \neg A \Rightarrow O^pC$), then the latter is a reparational obligation of the former. Their reciprocal interplay makes them two related norms so that they cannot be viewed anymore as independent obligations. Therefore we can combine them to obtain an expression (i.e., $\Gamma, \Delta \Rightarrow O^mA \otimes O^pC$) that exhibits the *explicit reparational obligation* of the second norm with respect to the first. Notice that the subject of the primary obligation and the subject of its reparation can be different, even if very often they are the same.

Let X, Y, Z be deontic operators. The following is the general rule for merging norms based on (Governatori & Rotolo 2006, Governatori 2005):

$$\frac{\Gamma \Rightarrow Xa \otimes (\bigotimes_{i=1}^n Yb_i) \otimes Zc \qquad \Delta, \neg b_1, \ldots, \neg b_n \Rightarrow Zd}{\Gamma, \Delta \Rightarrow Xa \otimes (\bigotimes_{i=1}^n Yb_i) \otimes Zd}$$

$$(1)$$

2.3.2 Removing Redundancies

Given the structure of the inference mechanism it is possible to combine rules in slightly different ways, and in some cases the meaning of the rules resulting from such operations is already covered by other rules. In other cases the rules resulting from the merging operation are generalisations of the rules used to produce them, consequently, the original rules are no longer needed in the specifications. To deal with this issue we introduce the notion of subsumption between rules. Intuitively a rule subsumes a second rule when the behaviour of the second rule is implied by the first rule. We first introduce the idea with the help of an example and then we show how to give a formal definition of the notion of subsumption appropriate for PCL.

Let us consider the rules

$$r : Invoice \Rightarrow O_{pr}^{a,\pi} PayWithin7Days \otimes O^p PayWithInterest$$
$$r' : Invoice, \neg PayWithin7Days \Rightarrow O_{n-pr}^{a,\pi} PayWithInterest.$$

The first rule says that after the seller sends the invoice the buyer has the achievement, persistent and preemptive obligation to pay within one week, otherwise immediately after the violation the buyer has to pay the principal plus the interest. Thus we have the primary obligation $O_{pr}^{a,\pi} PayWithin7Days$, whose violation is repaired by the secondary obligation $O^p PayWithInterest$. According to the second rule, given the same set of circumstances $Invoice$ and $\neg PayWithin7Days$ we have the achievement, persistent and non-preemptive obligation $O_{n-pr}^{a,\pi} PayWithInterest$. However,

- the primary obligation of r' obtains when we have a violation of the primary obligation of r;
- after the obligation $O_{pr}^{a,\pi} PayWithin7Days$ is violated, complying with the secondary obligation $O^p PayWithInterest$ of r entails complying with the primary obligation $O_{n-pr}^{a,\pi} PayWithInterest$ of r' (but not vice versa);
- hence, r is more general than r', and so the latter can be discarded from the formal representation of the specifications.

The intuitions we have just exemplified is captured by the following definitions.

Definition 1 *Let X, Y be two deontic operators in $\{O_{pr}^{a,\pi}, O_{n-pr}^{a,\pi}, O_{pr}^{a,\tau}, O_{n-pr}^{a,\tau}, O^m, O^p\}$. Then, $Y \sqsubseteq X$ iff*

(i) *if $Y = O_{pr}^{a,\pi}$, then*
   $X \in \{O_{pr}^{a,\pi}, O_{n-pr}^{a,\pi}, O_{pr}^{a,\tau}, O_{n-pr}^{a,\tau}, O^m, O^p\}$;
(ii) *if $Y = O_{n-pr}^{a,\pi}$, then $X \in \{O_{n-pr}^{a,\pi}, O_{n-pr}^{a,\tau}, O^m, O^p\}$;*
(iii) *if $Y = O_{pr}^{a,\tau}$, then*
   $X \in \{O_{pr}^{a,\pi}, O_{n-pr}^{a,\pi}, O_{pr}^{a,\tau}, O_{n-pr}^{a,\tau}, O^m, O^p\}$;
(iv) *if $Y = O_{n-pr}^{a,\tau}$, then $X \in \{O_{n-pr}^{a,\pi}, O_{n-pr}^{a,\tau}, O^m, O^p\}$;*
(v) *if $Y = O^m$, then $X = O^m$;*
(vi) *if $Y = O^p$, then $X \in \{O^p, O^m\}$.*

Definition 2 *Let Xa be a deontic literal and Y any deontic operator. If $X = \neg Y$, X is a* negative *operator; if $X = Y$, it is a* positive *operator.*

Definition 3 *Let $A = \bigotimes_{i=1}^m Xa_i$ and $B = \bigotimes_{i=1}^n Yb_i$ be two $\otimes$-expressions. Then, A deontically includes B iff $m = n$, and for each Xa_i, Yb_i*

- *$a_i = b_i$, and*
- *if X and Y are positive operators, then $Y \sqsubseteq X$.*

Definition 4 *Let $r_1 : \Gamma \Rightarrow A \otimes B \otimes C$ and $r_2 : \Delta \Rightarrow D$ be two rules, where $A = \bigotimes_{i=1}^m Xa_i$, $B = \bigotimes_{i=1}^n Yb_i$ and $C = \bigotimes_{i=1}^p Zc_i$. Then r_1 subsumes r_2 iff*

1. *$\Gamma = \Delta$ and A deontically includes D; or*
2. *$\Gamma \cup \{\neg a_1, \ldots, \neg a_m\} = \Delta$ and B deontically includes D; or*

3. *$\Gamma \cup \{\neg b_1, \ldots, \neg b_n\} = \Delta$ and $A \otimes \bigotimes_{i=0}^{k \leq p} Zc_i$ deontically includes D.*

The intuition behind subsumption is that the normative content of r_2 is fully included in r_1. Thus r_2 does not add anything new to the system and it can be safely discarded. To show this, some auxiliary notions should be used. In fact, Definition 3 (together with Definitions 1 and 2) establishes when the compliance conditions for an $\otimes$-expression cover the compliance conditions of another $\otimes$-expression[3]. For the sake of simplicity, take, for example, the single obligation $B = O_{n-pr}^{a,\tau} b$. Then, if another obligation A is equal to B, compliance conditions for both are trivially the same. If A is either $O_{n-pr}^{a,\pi} b$, $O^m b$, or $O^p b$, A deontically includes B, because, if both are in force, the compliance of A implies the compliance of B. However, if A is a preemptive achievement obligation, we have no guarantee that its compliance supports the compliance of B: indeed, b could have been obtained before A and B were in force, which is sufficient for fulfilling only A.

2.3.3 Solving Conflicts

Conflicts often arise in normative systems. Since PCL is based on Defeasible Logic, our framework allows us to solve them. In this regard, however, we have to determine whether we have genuine conflicts between $\otimes$-expressions or whether such $\otimes$-expressions admit states where all can be complied with.

Suppose, for example, that $A = O^p a \otimes O^m b$ and $B = O_{pr}^{a,\pi} \neg a \otimes O^m \neg b$ are both in force. The secondary obligations of A and B are clearly in contradiction but their primary obligations do not necessarily lead to a joint non-compliance. Indeed, if it is now forbidden to pay, and it is obligatory to pay by tomorrow, I can comply with both obligations by simply paying tomorrow.

Therefore, we have first to identify what $\otimes$-expressions do conflict with one another. First of all, let us define when two single obligations are in conflict:

Definition 5 *Let l, Xl, and Y be a literal, a deontic literal, and a positive operator, respectively. The complement $\sim l$ is $\neg p$ if $l = p$, and p if $l = \neg p$. The complement $\sim Xl$ is defined as follows:*

- *If $Xl = Yl$, $\sim Xl = \{Zp|Z$ is positive, $p = \sim l$, either $Z \sqsubseteq Y$ or $Y \sqsubseteq Z\} \cup \{\neg Zq|Z = Y, q = l\}$;*
- *If $Xl = \neg Yp$, $\sim Xl = \{Zq|Z$ is positive, $Z = Y, q = l\}$.*

The following definition states under what conditions two $\otimes$-expressions are in conflict.

Definition 6 *Let $A = \bigotimes_{i=1}^m Xa_i$ be an $\otimes$-expression. Then, $\sim A = \{B = \bigotimes_{i=1}^n Yb_i|m = n, \forall Xa_i, Yb_i : Xa_i = \sim Yb_i\}$.*

Given a theory consisting of a set of rules R, a set S of facts (literals and deontic literals), and a superiority relation, we can use the inference mechanism of Defeasible Logic to compute, in time linear to the size of the theory, the set of its conclusions. This implies to solve genuine conflicts by resorting to the superiority relation over the rules. Once we have defined when two $\otimes$-expressions are in conflict (Definition 6), we can simply use the same reasoning mechanism described in (Governatori 2005).

2.3.4 Normalisation Process

We now describe how to use the machinery presented in Section 2.3.1 and Section 2.3.2 to obtain PCL normal forms. The PCL normal form of a normative system provides a logical representation of normative specifications in a format that can be used to check the compliance of a process. This consists of the following steps:

---

[3]Notice that, in Definition 3, we do not care about the types of obligation when the deontic literals to be compared are negative. In fact, an expression like $\neg Ob$ is always complied with.

1. Starting from a formal representation of the explicit clauses of a set of normative specifications we generate all the implicit conditions that can be derived from the normative system by applying the merging mechanism of PCL.
2. We can clean the resulting representation by throwing away all redundant rules according to the notion of subsumption.
3. Finally we detect and solve normative conflicts.

In general the process at step 2 must be done several times in the appropriate order as described above. The normal form of a set of rules in PCL is the fixed-point of the above constructions. A normative system contains only finitely many rules and each rule has finitely many elements. In addition it is possible to show that the operation on which the construction is defined is monotonic (Governatori & Rotolo 2006), thus according to standard set theory results the fixed-point exists and it is unique.

3 Process Modelling

A business process model (BPM) describes the tasks to be executed (and the order in which they are executed) to fulfill some objectives of a business. BPMs aim to automate and optimise business procedures and are typically given in graphical languages. A language for BPM usually has two main elements: tasks and connectors. Tasks correspond to activities to be performed by actors (either human or artificial) and connectors describe the relationships between tasks: a minimal set of connectors consists of sequence (a task is performed after another task), parallel –AND-split and AND-join– (tasks are to be executed in parallel), and choice –(X)OR-split and (X)OR-join– (at least (most) one task in a set of task must be executed).

3.1 Execution Semantics

The basic execution semantics of the control flow aspect of a business process model is defined using token-passing mechanisms, as in Petri Nets. The definitions used here extend the execution semantics for process models given by (Vanhatalo et al. 2007) with semantic annotations in the form of effects and their meaning.

A process model is seen as a graph with nodes of various types –a single start and end node, task nodes, XOR split/join nodes, and parallel split/join nodes– and directed edges (expressing sequentiality in execution). The number of incoming (outgoing) edges are restricted as follows: start node 0 (1), end node 1 (0), task node 1 (1), split node 1 (>1), and join node >1 (1). The location of all tokens, referred to as a *marking*, manifests the state of a process execution. An execution of the process starts with a token on the outgoing edge of the start node and no other tokens in the process, and ends with one token on the incoming edge of the end node and no tokens elsewhere. Task nodes are executed when a token on the incoming link is consumed and a token on the outgoing link is produced. The execution of an XOR (Parallel) split node consumes the token on its incoming edge and produces a token on one (all) of its outgoing edges, whereas an XOR (Parallel) join node consumes a token on one (all) of its incoming edges and produces a token on its outgoing edge.

3.2 Annotation of Processes

A process model is then extended with a set of annotations, where the annotations describe (i) the artifacts or effects of executing the taks and (ii) the rules describing the obligations (and other normative positions) relevant for the process.

As for the semantic annotations, the vocabulary is presented as a set of predicates P. There is a set of process variables (x and y in Table 2), over which logical statements can be made, in the form of literals involving these
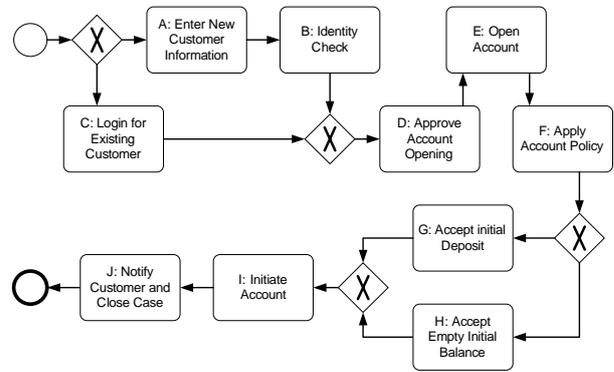

Figure 1: Example account opening process in private banking

variables. The task nodes can be annotated using *effects* which are conjunctions of literals using the process variables. The meaning is that, if executed, a task changes the state of the world according to its effect: every literal mentioned by the effect is true in the resulting world; if a literal l was true before, and is not contradicted by the effect, then it is still true (i.e., the world does not change of its own accord). We further assume that effects in parallel tasks do not contradict each other.

The obligations for this example are motivated by the following scenario: A new legislative framework has recently been put in place in Australia for anti-money laundering. The first phase of reforms for the *Anti-Money Laundering and Counter-Terrorism Financing Act 2006* (AML/CTF), covers the financial sector including banks, credit unions, building societies and trustees and extends to casinos, wagering service providers and bullion dealers. The act namely AML/CTF imposes a number of obligations, which include: customer due diligence (identification, verification of identity and ongoing monitoring of transactions); reporting (suspicious matters, threshold transactions and international funds transfer instructions); and record keeping. Table 2 shows the semantic effect annotations of the process activities.

Task	Semantic Annotation
A	$newCustomer(x)$
B	$checkIdentity(x)$
C	$checkIdentity(x), recordIdentity(x)$
D	$accountApproved(x)$
E	$owner(x,y), account(y)$
F	$accountType(y,type)$
G	$positiveBalance(y)$
H	$\neg positiveBalance(y)$
I	$accountActive(y)$
J	$notify(x,y)$

Table 2: Annotations for the process in Fig 1.

Here we give the norms governing this particular class of processes.

- All new customers must be scanned against provided databases for identity checks.

$$r_1 : newCustomer(x) \Rightarrow O_{pr}^{a;\tau} checkIdentity(x)$$

The meaning of the predicate *newCustomer(x)* is that the input data with $Id = x$ is a new customer, for which we have the obligation to check the provided data against provided databases *checkIdentity(x)*. The obligation resulting from this rule is a non-persistent obligation, i.e., as soon as a check has been performed, the obligation is no longer in force. In addition the obligation is preemptive, this means that if for some reasons the check was already previously performed there is no need to perform it again, e.g., if

the customer were an existing customer of a company recently acquired.

- Retain history of identity checks performed.

$$r_2 : checkIdentity(x) \Rightarrow O^m recordIdentity(x)$$

This rule establishes that there is a permanent obligation to keep record of the identity corresponding to the (new) customer identified by x. In addition this obligation is not fulfilled by the achievement of the activity (for example, by storing it in a database). We have a violation of the condition, if for example, the record x is deleted from the database.

- Accounts must maintain a positive balance, unless approved by a bank manager, or for VIP customers.

$$r_3 : account(y) \Rightarrow O^m positiveBalance(y) \otimes$$
$$O^{a,\pi}_{n-pr} approveManager(y)$$

The primary obligation is that each account has to maintain a positive balance *positiveBalance*; if this condition is violated (for any reason the account is not positive), then we still are in an acceptable situation if a bank manager approve the account not to be positive. In this case the obligation of approving persists until a manager approves the situation; after the approval the obligation is no longer in force.

$$r_4 : account(x), accountType(x, VIP) \Rightarrow$$
$$P \neg positiveBalance(x)$$

This rule creates an exception to rule r_3. Accounts of type VIP are allowed to have a non positive balance and no approval is required for this type of accounts (this is achieved by imposing that rule r_4 is stronger than rule r_3, $r_4 \prec r_3$).

4 Compliance Checking

Our aim in the compliance checking is to figure out (a) which obligations will definitely appear when executing the process, and (b) which of those obligations may not be fulfilled. In a way, PCL constraint expressions for a normative system define a behavioural and state space which can be used to analyse how well different behaviour execution paths of a process comply with the PCL constraints. Our aim is to use this analysis as a basis for deciding whether execution paths of a process are compliant with the PCL and thus with the normative system modelled by the PCL specifications. To this end we use the following procedure:

1. We traverse the graph describing the process and we identify the sets of effects (sets of literals) for all the tasks (nodes) in the process according to the execution semantics outlined in Section 3.1.
2. For each task we use the set of effects for that particular task to determine the normative positions (obligations, permissions, prohibitions) triggered by the task. This means that effects of a task are used as a set of facts, and we compute the conclusions of the defeasible theory resulting from the effects and the PCL rules annotating the process (see Sections 2 and 3.2). In the same way we accumulate effects, we also accumulate (undischarged) obligations from one task in the process to the task following it in the process.
3. For each task we compare the effects of the tasks and the obligations accumulated up to the task. If an obligation is fulfilled by a task, we discharge the obligation, if it is violated we signal this violation. Finally if an obligation is not fulfilled nor violated, we keep the obligation in the stack of obligations and propagate the obligation to the successive tasks.

Here, we assume that the obligations derived from a task should be fulfilled in the remaining of the process. Variations of this schema are possible: for example, one could stipulate that the obligations derived from a task should be fulfilled by the tasks immediately after the task. In another approach one could use a schema where for each task one has both preconditions and effects. Then the obligations derived from the preconditions must be fulfilled by the current task (i.e., the obligations must be fulfilled by the effects of the task), and the obligations derived from the effects are as in our basic schema.

4.1 From Tasks to Obligations

The second step to check process compliance is to determine the obligations derived by the effects of a task. Given a set of rules R and a set of literals S (plain literals and deontic literals), we can use the inference mechanism of defeasible logic (Section 2) to compute the set of conclusions (obligations) in force given the set of literals. These are the obligations an agent has to obey to in the situation described by the set of literals. However, the situation could already be sub-ideal, i.e., such that some of the obligations prescribed by the rules are already violated. Thus, given a set of literals describing a state-of-affairs one has to compute not only the current obligations, but also what reparation chains are in force given the set.

Consider a scenario where we have the rules $A \Rightarrow OB$ and $\neg B \Rightarrow OC$, and the effects are A and $\neg B$. The normal form of the rules is $A \Rightarrow OB \otimes OC$ and $\neg B \Rightarrow OC$. The only obligation in force for this scenario is OC. Since we have a violation of the first rule ($A \Rightarrow OB$ and $\neg B$), then we know that it is not possible to have an ideal situation here. Hence, computing only the current obligation does not tell us the state of the corresponding process. What we have to do is to identify the chain for the ideal situation for the task at hand. To deal with issue we have to identify the *active* reparation chains.

Some notational conventions. Given a rule r, $A(r)$ denotes the set of premises of the rules, and $C(r)$ the conclusion. For any set of rules R, $R[C]$ denotes the subset of R of rules whose conclusion is C. If $C = p_1 \otimes \cdots \otimes p_n \otimes q$ is a *reparation chain*, we use $\pi_i(C)$ to denote the i-th element of the chain.

Definition 7 *A reparation chain C is* active *given a set of literals S, if*

1. *$\exists r \in R[C] : \forall a_r \in A(r), a_r \in S$ and*
2. *$\forall s \in R[D]$ such that $\pi_1(C) \in D$, either*
    1. *$\exists a_s \in A(s) : \sim a_s \notin S$, or*
    2. *$\exists i \, \pi_i(D) = \sim \pi_1(C)$ and $\exists k, k < i, \sim \pi_k(D) \notin S$, or*
    3. *$\exists t \in R[E] : \pi_j(E) = \pi_1(C), \forall a_t \in A(t), a_t \in S, \forall m, m < j, \sim \pi_m(E) \in S$ and $t \prec s$.*

To illustrate the above definition we examine the following example.

Example 9 *Consider the rules*

$$r_1 : A_1 \Rightarrow OB \otimes OC,$$
$$r_2 : A_2 \Rightarrow O\neg B \otimes OD,$$
$$r_3 : A_3 \Rightarrow OE \otimes O\neg B.$$

The situation S is described by A_1 and A_3. In this scenario the active chains are $OB \otimes OC$ and $OE \otimes O\neg B$. The chain $OB \otimes OC$ is active since A_1 is in S and r_2 cannot be used to activate the chain $O\neg B \otimes OD$. For r_3 and the resulting chain $OE \otimes O\neg B$, we do not have the violation of the primary obligation OE of the rule (i.e., $\neg E$ is not in S), so the obligation $O\neg B$ is not entailed by r_3.

4.2 Checking Compliance

A reparation chain is in force if there are a rule of which the reparation chain is the consequent and a set of facts (effects of a task in a process) including the rule antecedents.

In addition we assume that, once in force, a reparation chain remains as such unless we can determine that it has been violated or the obligations corresponding to it have all been obeyed to (these are two cases when we can discharge an obligation or reparation chain). This means that it is not possible to have two instances at the same time of the same reparation chain. Accordingly, a reparation chain in force is uniquely determined by the combination of the task T when the chain has been derived and the rule R from which the chain has been obtained.

Definition 8 *Given a sequence of sets of literals $S_1, \ldots, S_n$ (corresponding to a sequence of tasks in a process model) a chain C is terminated by S_n if $\exists S_i$ such that C is active at S_i, $\exists s \in R[B]$ such that $B = A \otimes C$, and $\exists r \in R[E]$, $\pi_1(E) = \sim \pi_1(C)$, $A(r) \subseteq S_n$ and $s \not\succ r$.*

A terminated chain means that we have reached a deadline. First of all to terminate an active chain the chain must have been active. This means that there is a rule from which the chain as been derived (rule s above), and then we have reached a termination condition (encoded by rule r). The termination condition must be triggered ($A(r) \subseteq S_n$), and the terminating rule should not be weaker than the rule from which the chain has been derived. Given a task/set of literals S we will use *Terminated* to refer to the set of rules terminated by S.

The procedure for compliance checking has two steps. In the first step, given a set of literals S, corresponding to the effects of a task T in a process model, we identify the set (*Current*) of active chains for the process. The set of the active chains includes the new active chains triggered by the task, as well as the chains carried out from previous tasks. Then, in the second phase, the algorithm *CheckCompliance* scans all elements of *Current* against the set of literals S, and determines the state of each reparation chain ($C = A_1 \otimes A_2$, where A_2 can be $\perp$, making $A_1 \otimes A_2$ equivalent to A_2, see (Governatori & Rotolo 2006)) in *Current* and *Terminated*. *CheckCompliance* operates as follows:[4]

for $C \in$ *Current*
  if $A_1 = O^p B$, then
    if $B \in S$, then
      remove($[T, R, A_1 \otimes A_2]$, *Current*)
      if $[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2] \in$ *Violated*, then
        add($[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2]$, *Compensated*)
    else
      remove($[T, R, A_1 \otimes A_2]$, *Current*)
      add($[T, R, A_1 \otimes A_2, B]$, *Violated*)
      add($[T, R, A_2]$, *Current*)
  if $A_1 = O^{a,x} B$, $x \in \{\pi, \tau\}$, then
    if $B \in S$, then
      remove($[T, R, A_1 \otimes A_2]$, *Current*)
      remove($[T, R, A_1 \otimes A_2]$, *Unfulfilled*)
      if $[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2] \in$ *Violated*, then
        add($[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2]$, *Compensated*)
    else
      add($[T, R, A_1 \otimes A_2]$, *Unfulfilled*)
  if $A_1 = O^m B$, then
    if $B \notin S$ or $\neg B \in S$, then
      add($[T, R, A_1 \otimes A_2, B]$, *Violated*)
      add($[T, R, A_2]$, *Current*)
for $C \in$ *Terminated*
  if $[T, R, A_1 \otimes A_2] \in$ *Unfulfilled*, then
    add($[T, R, A_2]$, *Current*)
    add($[T, R, A_1 \otimes A_2, B]$, *Violated*)

if $A_1 = O^{a,\tau}$, then
    remove($T, R, A_1 \otimes A_2$, *Current*)

Let us examine the *CheckCompliance* algorithm. Remember the algorithm scans all active reparation chains one by one, and then for each of them reports on the status of it. For each chain in *Current* (the set of all active chains), it looks for the first element of the chain and it determines the content of the obligation (so if the first element is $O^x B$, the content of the obligation in B). Then it checks whether the obligation has been fulfilled (B is in the set of effects), or violated ($\neg B$ is in the set of effects), or simply we cannot say anything about it (none of B and $\neg B$ is in the set of effects). In the first case, for punctual and achievement obligations we can discharge the obligation and we remove the chain from the set of active chains (similarly if the obligation was carried over from a previous task, i.e., it was in the set *Unfulfilled*); for maintenance obligation we have to keep the obligation among the current obligations. In case of a violation, we add the information about it in the system. Notice that we can use current to detect a violation only for punctual and maintenance obligations. To record that we have a violation we insert a tuple with the identifier of the chain and what violation we have in the set *Violated*. In addition, we know that violations can be compensated, thus if the chain has a second element we remove the violated element from the chain and put the rest of the chain in the set of active chains. Here we take the stance that a violation of a maintenance obligation does not discharge it, thus we do not remove the chain from the set of active chains; however, this is the case for a punctual obligation. In case we do not have information about whether B or $\neg B$ we do not have the information to assert that a maintenance obligation has been fulfilled, thus it has been violated. However, for achievement obligations the set of effects does not tell us if the obligation has been fulfilled or violated, so we propagate the obligation to the successive tasks by putting the chain in the set *Unfulfilled*. The algorithm also checks whether a chain/obligation was previously violated but it was then compensated. In the final part of the algorithm we consider the terminated chains, i.e., the chains for which we have reached a deadline. If the obligation has not been fulfilled we signal a violation as we have already described above. The important point to notice here is that in case of a persistent achievement obligation, the chain is keep in the set of current chains, but this is not the case for a non-persistent achievement obligation.

To check the compliance of business process against a set of normative specifications we can use the algorithm *CheckCompliance*. The algorithm given a set of literals a set of rules determines (1) the rules that have been fired (deriving thus the obligations in force for the situation described by the set of literals) (2) which obligations have been fulfilled, violated, or we do not have enough information to assert their normative state.

At run time an instance of a business process defined by a business process model is a sequence of (sets) of tasks. For each element of the sequence we generate the set of effects and then we use the *CheckCompliance* to determine the state of the obligations. Thus determining whether a business process is compliant is to determine whether the business process can be executed without end in a state where there an unrepaired violations.

The conditions below relate the state of a process based as reported by the *CheckCompliance* algorithm and the semantics for PCL expressions. In particular, a process is compliant if the situation at the end of the process is at least sub-ideal (it is possible to have violations but these have been compensated for). Similarly a process is fully compliant if it results in an ideal situation (i.e., there are no violations).

Definition 9

- *An execution path is* compliant *iff for all $[T, R, A] \in$*

---

[4]In the presentation of the algorithm we do not differentiate between preemptive and non-preemptive achievement obligations. A simple solution to handle both at the same time is that of labelling each effect with the task the effect is associated with. According to fulfil a preemptive obligation O_{pr}^a in force from a task A we have to have p^B, where B is the identified of the task the effect p is associated with, such that either task B follows task A (and this is the case also for non-preemptive achievement obligations), or task B precedes task A and there is no task C between B and A such that $\neg p^C$. We overload the inclusion operator $\in$ in the algorithm below with such functionalities.

Current, $A = OB \otimes C$, for every $[T,R,A,B] \in Violated$, $[T,R,A,B] \in Compensated$ and $Unfulfilled = \emptyset$.

- *An execution path is* fully compliant *iff for all $[T,R,A] \in Current$, $A = OB \otimes C$, $Violated = \emptyset$ and $Unfulfilled = \emptyset$.*

Accordingly, an execution path is not compliant if the set of unfulfilled obligations (*Unfulfilled*) is not empty. Consider, for example the rule

$$r_3 : account(y) \Rightarrow O^m positiveBalance(y) \otimes$$
$$O^{a,\pi}_{n-pr} approveManager(y)$$

relative to the process of Figure 1 with the annotation as in Table 2. After task E we have, among others, the effect $account(y)$. This means that after task E we have the chain

$$[E, r_4, O^m positiveBalance(y) \otimes O^{a,\pi}_{n-pr} approveManager(y)]$$

in *Current* for task F. After task F, the above entry for the chain obtained from rule r_4 is moved to the set *Unfulfilled*. Suppose now that tasks G and H do not have any annotation attached to them. In this case at the end of the process we still have the active chain, but the resulting situation is not ideal: the antecedent of the rule is a subset of the set of effects, but we do not have the first element of the chain as one of the effects. Thus, the process is not compliant.

It is possible to give two definitions of compliance for business processes.

Definition 10 *A business process is* absolutely compliant *if every execution path is compliant. A business process is* weakly compliant *if at least one execution path is compliant.*

## 5 Compliance at Runtime and Design Time: A Balance and Open Problems

The algorithm to check compliance determines, given an execution path of a business process, whether the execution path is compliant or not. The algorithm has linear time complexity. As we have seen *CheckCompliance* work on an execution path thus the same algorithm can be use to check compliance at design time and at run time. Thus checking compliance at run time (also known as conformance) can be performed in linear time. However, at run time the algorithm returns the obligations in force for every task with no look ahead features. Despite the linear time complexity of *CheckCompliance* the problem of checking compliance of a business process at design time is in general computationally hard no matter the notion of compliance one chooses (absolute or weak). This depends on the number of possible execution paths for a process.

The first source of complexity is given by (X)OR-split/(X)OR-joins. The semantics for such a construct is that for each (X)OR-split at least one of the paths is executed. Thus the number of execution paths to examine for multiple (X)OR-split/joins exhibits a tree-like growth.

As we have seen the algorithm takes the set of effects that hold after a task and it uses them to determine the active chains and the obligations in force for the successive task. Given the persistence of some types of obligations, some obligations carry over from one task to the (immediately) successive task(s).

The propagation of persistent obligation from task to task is (computationally) trivial for a linear sequence of tasks, and so is the propagation of effects. Again the propagation of effects over AND-splits and AND-joins has linear time complexity. All one has to do is to traverse the single paths, cumulate the effects, and then union the effects when one reaches the AND-join. For the accumulation of effects we assume, as is typically the case, that effects persist from one task to the successive one, and if the new task introduces an effect contradicting an effect

obtained in a previous task, this corresponds to an update, i.e., we remove the old one and we add the new one. Based on this assumption we have that for every task the set of effects cumulated for the task is consistent, and so is every path. Hence, it has been argued that parallel paths should not introduce inconsistency (inconsistent parallel paths correspond to a incorrect design of a process). It is possible to adopt such solution for weak compliance. We check the compliance of each parallel sub-process in an AND-split/AND-join if all of them are compliant we are guaranteed that there exists at least one execution path combining all the tasks to be executed in parallel is compliant. However, such a strategy is far from being satisfactory for absolute compliance apart for the case where we only have pre-emptive[5], non-terminating achievement obligations, and the triggering of the obligation either happened before the AND-split or the obligation does not admit exceptions. For the other cases the compliance of all individual parallel executions does not guarantee that the process is absolutely compliant. Consider for example a process with the following sub-process
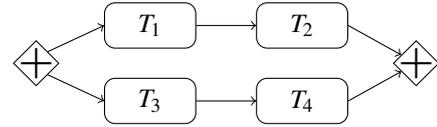
where T_1 is annotated with a, T_2 with b, T_3 with c, and T_4 with d. The rules are $r_1 : a \Rightarrow O^m c$ and $r_2 : b \Rightarrow O^m d$. It is immediate to verify that both execution paths T_1, T_2 and T_3, T_4 are compliant. At the same time it is easy to see that some sequential combinations of the four task are compliant, e.g., T_1, T_3, T_2, T_4 is not compliant since b is not an effect of T_3 and thus we have a violation of the maintenance obligation obtained from the effect of T_1 and rule r_1. At the same time the execution path $T_1, \{T_2, T_3\}, T_4$ is compliant[6]. Similar examples can be given for punctual and non-preemptive achievement obligations.

Accordingly, a process designer is left with the dilemma of choosing between weak compliance at design time with the consequence that the execution of the process might not be compliant (the designer has no control on what execution path will be performed at run time), or to face absolute compliance with a combinatorial explosion of the number of possible execution paths to be verified for compliance. Another alternative is to spend more time on the design of the process and to give additional structure to the process to introduce further synchronisation for parallel tasks. However, most of the current languages for modelling business processes have very limited capabilities to model true synchronisation. A possible solution is to extend business process language with temporal constraint language to introduce truly synchronisation constructs (see for example (Lu et al. 2006)).

6 Summary and Related Work

This paper discusses a means of visualizing the impact of compliance controls on business process and of assisting in compliance checking, analysis and feedback for subsequent (re)design of the processes. The procedure is based in principle on efficient algorithms and is able to deal with reparation chains of deontic statements of various types.

A number of works have been devoted to compliance in control modelling. (Goedertier & Vanthienen 2006) presents the logical language PENELOPE, that provides the ability to verify temporal constraints arising from compliance requirements on effected business processes.

---

[5]The meaning of a pre-emptive obligation is that something must happen in a process, but it does not matter where and when it happens in the process.

[6]Remember we have defined an execution path as a sequence of sets of tasks. The intended meaning is that each element of the sequence is a granule of time (identified with at least one task), and we group together truly concurrent tasks. To simplify the notation we drop the set theoretic notation for singletons.

(Küster et al. 2007) develops a method to check compliance between object lifecycles that provide reference models for data artifacts e.g. insurance claims and business process models. (Giblin et al. 2006) provides temporal rule patterns for regulatory policies, although the objective of this work is to facilitate event monitoring rather than the usage of the patterns for support of design time activities. Furthermore, (Agrawal et al. 2006) presented an architecture for supporting Sarbanes-Oxley Internal Controls, which include functions such as workflow modelling, active enforcement, workflow auditing, as well as anomaly detection. (Farrell et al. 2005) studies the performance of business contract based on their formal representation. (Desai et al. 2008) seeks to provide support for assessing the correctness of business contracts represented formally through a set of commitments. The reasoning is based on value of various states of commitment as perceived by cooperative agents. (Ghose & Koliadis 2007) consider an approach where the tasks of a business process model, written in BPMN, are annotated with the effects of the tasks, and a technique to propagate and cumulate the effects from a task to a successive contiguous one is proposed. The technique is designed to take into account possible conflicts between the effects of tasks and to determine the degree of compliance of a BPMN specification. Also, there have been recently some efforts towards support for process modelling against compliance requirements. (zur Muehlen & Rosemann 2005) provides a method for integrating risks in business processes. The proposed technique for "risk-aware" business process models is developed for EPCs (Event-driven Process Chains) using an extended notation. (Sadiq et al. 2007) proposes an approach based on control tags to visualize internal controls on process models. (Liu et al. 2007) takes a similar approach of annotating and checking process models against compliance rules, although the visual rule language (BPSL) does not directly address the deontic notions providing compliance requirements.

As far as we are aware of the issue of studying compliance with a rich ontology of obligations has been neglected. Indeed, the majority of work ignore the normative aspects related to the issue, and those that address the normative aspects are limited to pre-emptive achievement obligations. The only exception is (Governatori et al. 2008) where, an algorithm for approximate weak compliance is presented, based on some heuristics.

References

Agrawal, R., Johnson, C. M., Kiernan, J. & Leymann, F. (2006), Taming compliance with Sarbanes-Oxley internal controls using database technology, *in* 'Proc. ICDE 2006'.

Antoniou, G., Billington, D., Governatori, G. & Maher, M. J. (2001), 'Representation results for defeasible logic', *ACM Transactions on Computational Logic* **2**(2), 255–287.

Carmo, J. & Jones, A. (2002), Deontic logic and contrary to duties, *in* D. Gabbay & F. Guenther, eds, 'Handbook of Philosophical Logic, 2nd Edition', Kluwer.

Desai, N., Narendra, N. C. & Singh, M. P. (2008), Checking correctness of business contracts via commitments, *in* 'Proc. AAMAS 2008'.

Farrell, A. D. H., Sergot, M. J., Sallé, M. & Bartolini, C. (2005), 'Using the event calculus for tracking the normative state of contracts', *International Journal of Cooperative Information Systems* **14**.

Ghose, A. & Koliadis, G. (2007), Auditing business process compliance, *in* 'Service Oriented Computing, ISOC 2007', LNCS, Springer, pp. 169–180.

Giblin, C., Müller, S. & Pfitzmann, B. (2006), From regulatory policies to event monitoring rules: Towards model driven compliance automation, Technical report, IBM Zurich Research Lab.

Goedertier, S. & Vanthienen, J. (2006), Designing compliant business processes with obligations and permissions, *in* 'Business Process Management (BPM) Workshops'.

Governatori, G. (2005), 'Representing business contracts in RuleML', *International Journal of Cooperative Information Systems* **14**(2-3), 181–216.

Governatori, G., Hoffmann, J., Sadiq, S. W. & Weber, I. (2008), Detecting regulatory compliance for business process models through semantic annotations, *in* 'Business Process Management Workshops', pp. 5–17.

Governatori, G., Hulstijn, J., Riveret, R. & Rotolo, A. (2007), Characterising deadlines in temporal modal defeasible logic, *in* 'Proc. Australian AI 2007'.

Governatori, G. & Rotolo, A. (2006), 'Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations', *Australasian Journal of Logic* **4**, 193–215.

Governatori, G. & Rotolo, A. (2008*a*), An algorithm for business process compliance, *in* G. Sartor, ed., 'Jurix 2008', IOS Press, pp. 186–191.

Governatori, G. & Rotolo, A. (2008*b*), Changing legal systems: Abrogation and annulment. part ii: Temporalised defeasible logic, *in* 'Proceedings of Normative Multi Agent Systems (NorMAS 2008)'.

Governatori, G., Rotolo, A. & Sartor, G. (2005), Temporalised normative positions in defeasible logic, *in* '10th International Conference on Artificial Intelligence and Law (ICAIL05)', pp. 25–34.

Governatori, G. & Sadiq, S. (2009), The journey to business process compliance, *in* H. Cardoso & W. van der Aalst, eds, 'Handbook of Research on BPM', IGI Global, pp. 426–454.

Küster, J. M., Ryndina, K. & Gall, H. (2007), Generation of business process models for object life cycle compliance, *in* 'Proc. BPM 2007'.

Liu, Y., Müller, S. & Xu, K. (2007), 'A static compliance-checking framework for business process models', *IBM Systems Journal* **46**(2), 335–362.

Lu, R., Sadiq, S., Padmanabhan, V. & Governatori, G. (2006), Using a temporal constraint network for business process execution, *in* 'Seventeenth Australasian Database Conference (ADC2006)', pp. 157–166.

Lu, R., Sadiq, S. W. & Governatori, G. (2007), Compliance aware business process design, *in* 'Business Process Management Workshops', pp. 120–131.

Sadiq, S. & Governatori, G. (2009), A methodological framework for aligning business processes and regulatory compliance, *in* J. van Brocke & M. Rosemann, eds, 'Handbook of Business Process Management', Springer.

Sadiq, S., Governatori, G. & Naimiri, K. (2007), Modelling of control objectives for business process compliance, *in* 'Proc. BPM 2007'.

Vanhatalo, J., Völzer, H. & Leymann, F. (2007), Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition, *in* 'Proc. ICSOC 2007'.

zur Muehlen, M. & Rosemann, M. (2005), Integrating risks in business process models, *in* 'Proc. ACIS 2005'.