Law, Logic and Business Processes

Guido Governatori
NICTA, Queensland Research Laboratory,
St Lucia, QLD, 4072, Australia
guido.governatori@nicta.com.au

***Abstract*—Since its inception one of the aims of legal informatics has been to provide tools to support and improve the day to day activities of legal and normative practice and a better understanding of legal reasoning. The internet revolutions, where more and more daily activities are routinely performed with the support of ITC tools, offers new opportunities to legal informatics. We argue that the current technology begins to be mature enough to embrace in the challenge to make intelligent ICT support widespread in the legal and normative domain. In this paper we examine a logical model to encode norms and we use the formalisation of relevant law and regulations for regulatory compliance for business processes.**

***Keywords*-Regulatory Compliance, Business Process Compliance, Law, Deontic Logic, Defeasible Logic**

I. Introduction

A general research aim in the AI&Law field is to design computer systems whose performance is constrained by suitable sets of legal norms: in this sense, norms establish what legality criteria should apply to their functioning. However, the general idea of regulating computer systems can be modeled in different ways. As [1] pointed out in the field of multi-agent systems, norms may work either as hard or soft constraints. In the first case, computer systems are designed in such a way as to avoid legal violations, and so norms in fact limit in advance systems' behaviour. In the second case, norms rather provide standards which can be violated, even though any violations should result in sanctions or other normative effects applying to non-compliant systems. Thus, we have to monitor the behavior of systems and enforce the sanctions. In both perspectives, it is of paramount importance to develop mechanisms for enforcing and detecting *norm compliance*.

Compliance (or regulatory compliance, as it is often termed) is aimed at ensuring that business processes, operations and practise are in accordance with a prescribed and/or agreed set of norms. Accordingly the term *compliance* is used to denote adherence of one set of rules (the 'source rules') against other set of rules ('target rules'). This means that the specifications for a business process (target rules) do not violate the relevant laws and regulations (source rules). 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.[1]

[1]For an overview for a comprehensive methodological approach to business process compliance, see [2], [3].

Compliance is a relationship between two set of specifications, where one the target does not result in violation of the source. This means that to determine whether a business process is compliant with relevant laws or regulations, one has to have:

1) a formal specification of the business process;
2) a formal specification of the (relevant) laws or regulation;
3) a common framework to interface the two sets of specifications.

A further concern is that the set of specifications typically have different purposes. Business processes describe *how* to achieve core business objectives, while norms, in general, specify *what* activities are permitted, prohibited of required for particular classes of businesses. Thus, the common framework should take into account and reconcile these two aspects.

Despite the relevance of compliance, to our knowledge no systematic investigation has been devoted to it in the field of AI&Law community, whereas its importance has increased over the last few years in other related research fields such as in business modeling.

Deontic logic and AI&Law disciplines have a long history in modelling and formalising normative concepts and normative reasoning. However, by most, they ignored the business process aspects. On the other hand, research on business process management and modelling, has provided a wealth of techniques for handling, modelling and managing business processes, and as we have just said, they are now investigating regulatory compliance for business processes. The majority of the proposed methodologies neglects the normative aspects.

The aim of research in business process management and modelling is to provide practical tools for enterprises. Nowadays more and more businesses in all sectors relay and heavily depends on their process aware information systems. These systems are now essential to control, administer and enact all core business activities, and business processes are increasingly constrained by the legislative and regulatory framework in which they operate. Furthermore, failure to comply is no longer an option. This means that research in business processes must now address the issue of how to incorporate techniques to handle normative requirements, and these techniques should be conceptual, in the sense that they should provide notions and construction as close as possible to the concepts they intend to models. Without this practitioners are left with no methodology to tackle this problem, and ad-hoc solutions appear as the only option for otherwise routine situations. This leads to increased cost, and this is

one of the reasons why compliance is still seen as burden on an enterprise, instead of an opportunity to improve the performance [2].

II. Requirements for Normative Languages

The previous discussion points out that the ability to model normative requirements, and then reasoning with (and about) them is essential to have a successful approach to business process compliance. This raises the question about what are the requirements for a normative language for compliance of business processes?

The law is a complex phenomenon, which can be analyzed into different branches according to the authority who produces legal norms and according to the circumstances and procedures under which norms are created. But, independently of these aspects, it is possible to identify some general features that norms should enjoy.

First of all, it is widely acknowledged in legal theory and AI & Law that norms have basically a conditional structure like [4], [5]

$$\text{if } A_1,\ldots,A_n \text{ then } B \tag{1}$$

where $A_1,\ldots,A_n$ are the applicability conditions of the norm and B denotes the legal effect which ought to follow when those applicability conditions hold[2].

This very general view highlights an immediate link between the concepts of norm and rule. However, there are many types of rules. The common sense, dictionary meaning of rule is "One of a set of explicit or understood regulations or principles governing conduct within a particular sphere of activity." [6]. In classical logic, rules can be inference rules or material implications. In computer science, rules can be production rules, grammar rules, or rewrite rules.

When we use the term 'rule' in the legal field, we usually mean rule in the regulatory sense. But rules express not only regulations about how to act. For example, von Wright [7] classified norms into the following main types (among others):

1) determinative rules, which define concepts or constitute activities that cannot exist without such rules. These rules are also called in the literature 'constitutive rules'.
2) technical rules, which state that something has to be done in order for something else to be attained;
3) prescriptions, which regulate actions by making them obligatory, permitted, or prohibited. These norms, to be complete, should indicate
   - who (the norm-subjects)
   - does what (the action-theme)
   - in what circumstances (the condition of application)
   - the nature of their guidance (the mode).

Notice that the notion of norm proposed by von Wright is very general and extends well over the notion of norm in legal reasoning.

[2]Indeed, norms can be also unconditioned, that is their effects may not depend upon any antecedent condition. Consider, for example, the norm "everyone has the right to express his or her opinion". Usually, however, norms are conditioned. In addition, unconditioned norms can formally be reconstructed in terms of (1) with no antecedent conditions.

In [8] a comprehensive list of requirements for (rule) languages for modelling norms is given. Some of the features listed are not relevant for typical business processes (for example, features related to legal procedure such modelling different types of burden of proof), while other are just marginal for the domain at hand (e.g., the notion of jurisdiction and authority). However, three features, we believe are particularly important for modelling compliance: defeasibility, the normative (deontic) effect of a norm, and whether the effects of a norm are persistent.

Defeasibility, the notion that a conclusion can be withdrawn, in case of additional information that allow to conclude the opposite, is very important in modelling normative reasoning where we can combine different sources for norms. In addition it is an efficient mechanism to capture and reasoning with exceptions, and when we are in scenario where we have incomplete information. This is particularly suitable for processes, where we can derive plausible conclusion based on the information provided, but when more date becomes available the conclusion can be changed.

The deontic effects (obligations, permissions, prohibitions) are essential to capture compliance. They specify what is permitted, forbidden and required to fulfill the normative requirements.

Finally the features about the persistency of normative effects allow us to define different types of obligations, with different conditions for their fulfillment of violation. This features has the effect that we are able to model different types of common scenarios (and some not so common as well).

III. A Compliance Framework and Challenges

In this paper we extend the preliminary work developed in [9] and further investigate how to model compliance in business processes. In [9] 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 [9] 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 [10]. This paper adopts the same methodology, but it generalises [9]'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.

A *first source of complexities* was already (partially) addressed in [9]. 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 [11]). 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 IV-A). In [12] 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.

IV. Normative Constraints

A. Violations and Types of Obligations

Achievement, Maintenance and Punctual Obligations We can distinguish *achievement obligations* from *maintenance obligations* [12].

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 RELaw are made available to RELaw 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.

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 *puctual 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 are 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 RELaw are made available to RELaw 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 RELaw 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-premptive 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 premptive or a non-preemptive one. In the next section, we will extend the approach of [9], [10] to cover these cases.

B. 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 IV-A. 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 [13]) and a deontic logic of violations [10]. The current version of PCL significantly extends the logic of [9] by working on all types of obligations discussed in Section IV-A. 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 processes. Often processes 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^x_y (for obligation), where y can be empty. Based on the discussion in Section IV-A we have three main classes of deontic operators: punctual obligations (O^p), maintenance obligations (O^m) and achievement obligations (O^a); achievement obligations in turn can be classified based on two orthogonal distinctions: persistent ($O^{a,\pi}$) vs non-persistent ($O^{a,\tau}$), and preemptive ($O^{a,x}_{pr}$) vs non-preemptive ($O^{a,x}_{n-pr}$). The formulas of PCL will be constructed in two steps according to the following formation rules:

- every propositional letter is a literal;

- 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 $O^{a,\pi}_{pr} A \otimes O^{p} B \otimes O^{m} C$ 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 $O^{a,\pi}_{pr} A$. Similarly C is the reparation of the obligation $O^{p} B$, 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 $r_1 : a \Rightarrow O^{m} b \otimes O^{a,\pi}_{n-pr} c$ and $r_2 : d \Rightarrow \neg O^{a,\pi}_{pr} c$, $r_1 \prec r_2$ 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 [14], [15]. Consider Example 5 (we assume that the reparational obligation is a punctual one):

inv_{init} invoice $\Rightarrow O^{a,\pi}_{pr}$ pay $\otimes O^{p}$ pay_fine
inv_{term} pay $\rightsquigarrow \neg O^{a,\pi}_{pr}$ 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:

pos_{init} open_account $\Rightarrow O^{m}$positive $\otimes O^{p}$blocked
pos_{term} bank_charges $\rightsquigarrow \neg O^{m}$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:

rev_{init} papers_available $\Rightarrow O^{a,\tau}_{n-pr}$review $\otimes O^{a,\pi}_{pr}$blacklist
rev_{term_1} notification $\rightsquigarrow \neg O^{a,\tau}_{n-pr}$review
rev_{term_2} review $\rightsquigarrow \neg O^{a,\tau}_{n-pr}$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.

C. Normal Forms and Reasoning

PCL is equipped with two reasoning mechanisms: the first is used to generate normal forms, the second, given a case, determines what normative effects (permissions, prohibitions, obligations) are in force (for the full details see [16], [17]).

A normal form is a representation of a normative document based on a PCL specification containing all conditions that can generated/derived from the given PCL specification. The purpose of a normal form is to "clean up" the PCL representation of a normative document, that is to identify formal loopholes, deadlocks and inconsistencies in it, and to make hidden conditions explicit. Normal forms are used in the compliance checking procedure to determine whether a process is compliant or not. Specifically, PCL mechanisms to derive new rules from existing rules, remove redundant rules, i.e., rules subsumed by other rules, and to identify conflicts. The normalisation process, in turn

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 document by applying the merging mechanism of PCL.
2) We can clean the resulting representation of the contract by removing all redundant rules according to the notion of subsumption.
3) Finally, we use the conflict identification rule to label and detect conflicts.

In general, the above process must be repeated several times in the appropriate order. The normal form of a set of rules in PCL is the fixed-point of the above constructions. A normative document 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 [10], thus according to standard set theory results the fixed-point exists and it is unique. Notice that the computation of the fixed-point depends on the order in which the operations of subsumption and merging are performed (subsumption fixed and merging after). Changing the order of these operations, i.e. merging first and subsumption after, or interleaving the two operations does not produce the same result. Specifically, some rules might be excluded from the computation. For the full details see [17], [16]

PCL reasoning is based on Defeasible Logic. To derive a conclusion we need a rule for the conclusion we want to derive, and the premises of the rule must have already

been proved. In addition, we have to consider all possible counterarguments, i.e., rule for the complementary conclusion. For each of such arguments we have to show that either the rule does not fire, that is, at least on of the premises does not hold, or we have rebut the argument providing a stronger one. For this we use the superiority relation and we have to show that the defeating rule fires in the given cases. If the conclusion of rule is an obligation chain $OA_1 \otimes \cdots \otimes OA_n$, and we want to prove OA_i, $1 < i \leq n$ then we have to show that $\neg A_j$ holds for all $1 \leq j < i$. For the full details see [13], [17].

V. 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).

A. 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 [18] 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.

B. 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 and (ii) the rules describing the obligations (and other normative positions) relevant for the process.

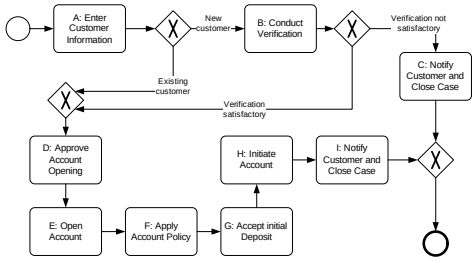

Fig. 1. Example account opening process in private banking

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 I), over which logical statements can be made, in the form of literals involving these variables. The task nodes can be annotated using *effects* (also referred to as *postconditions*) 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 I shows the semantic effect annotations of the process activities.

Task	Semantic Annotation	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 I
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.

$$r_1 : newCustomer(x) \Rightarrow O^{a,\tau}_{pr} checkIdentity(x)$$

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.

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

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, $r_4 \prec r_3$).

VI. 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 V-A.
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 execution of 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 IV and V-B). 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.

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.

A. 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 IV) 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 1: 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 $t \prec s$.

The example below illustrates the definition.

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.

B. 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 2: 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 $\bot$, making $A_1 \otimes A_2$ equivalent to A_2, see [10]) in *Current* and *Terminated*. *CheckCompliance* operates as follows:[3]

```
for C ∈ Current
    if A1 = O^p B, then
      if B ∈ S, then
        remove([T,R,A1 ⊗ A2], Current)
        if [T,R,B1 ⊗ B2 ⊗ A1 ⊗ A2,B2] ∈ Violated, then
          add([T,R,B1 ⊗ B2 ⊗ A1 ⊗ A2,B2], Compensated)
      else
        remove([T,R,A1 ⊗ A2], Current)
        add([T,R,A1 ⊗ A2,B], Violated)
        add([T,R,A2], Current)
    if A1 = O^{a,x} B, x ∈ {π,τ}, then
      if B ∈ S, then
        remove([T,R,A1 ⊗ A2], Current)
        remove([T,R,A1 ⊗ A2], Unfulfilled)
        if [T,R,B1 ⊗ B2 ⊗ A1 ⊗ A2,B2] ∈ Violated, then
          add([T,R,B1 ⊗ B2 ⊗ A1 ⊗ A2,B2], Compensated)
      else
        add([T,R,A1 ⊗ A2], Unfulfilled)
    if A1 = O^m B, then
      if B ∉ S or ¬B ∈ S, then
        add([T,R,A1 ⊗ A2,B], Violated)
        add([T,R,A2], Current)
for C ∈ Terminated
    if [T,R,A1 ⊗ A2] ∈ Unfulfilled, then
      add([T,R,A2], Current)
      add([T,R,A1 ⊗ A2,B], Violated)
      if A1 = O^{a,τ}, then
        remove(T,R,A1 ⊗ A2, 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

[3]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. Accordingly to fulfil a preemptive obligation $O^{a,x}_{pr}$ 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.

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. In this perspective the issue of 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 3:

- An execution path is *compliant* iff for all $[T,R,A] \in$ *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 I. 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 4: 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.

VII. 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. However, we identified some computational limits which, we believe, should open new lines of research in the AI&Law community with regard to the concept of compliance.

A number of works have been devoted to compliance in control modelling. [19] presents the logical language PENELOPE, that provides the ability to verify temporal constraints arising from compliance requirements on effected business processes. [20] develops a method to check compliance between object lifecycles that provide reference models for data artifacts e.g. insurance claims and business process models. [21] 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, [22] 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. [23] studies the performance of business contract based on their formal representation. [24] 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. Also, there have been recently some efforts towards support for process modelling against compliance requirements. [25] 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. [26] proposes an approach based on control tags to visualize internal controls on process models. [27] 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 [28] where, an algorithm for approximate weak compliance is presented, based on some heuristics.

Finally, in this paper we restricted ourselves just to what we call semantic compliance, the effects of tasks do not violate normative requirements. However, in some cases, normative requirements specify not only what must be done, but also how to do that. For example, they can specify to follow a sequence of steps. In [29] we have shown how to use PCL to model the standard control flows of business processes.

VIII. Acknowledgements

The work presented in this paper is the result of an ongoing development on many aspects: deontic logic, rule languages for normative reasoning and business processes. I would like to thank Antonino Rotolo, Thomas Gordon and Shzia Sadiq for their valuable contribution in the past year to the development of a framework for business process compliance.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] G. Boella and L. van der Torre, "Fulfilling or violating obligations in multiagent systems," in *Procs. IAT04*, 2004.

[2] S. Sadiq and G. Governatori, "Managing regulatory compliance in business processes," in *Handbook of Business Process Management*, J. van Brocke and M. Rosemann, Eds. Berlin: Springer, 2010, vol. 2, ch. 8, pp. 157–173.

[3] G. Governatori and S. Sadiq, "The journey to business process compliance," in *Handbook of Research on BPM*, J. Cardoso and W. van der Aalst, Eds. IGI Global, 26 October 2008 2009, ch. 20, pp. 426–454.

[4] H. Kelsen, *General theory of norms*. Oxford: Clarendon, 1991.

[5] G. Sartor, "Legal reasoning: A cognitive approach to the law," in *A Treatise of Legal Philosophy and General Jurisprudence*, E. Pattaro, H. Rottleuthner, R. Shiner, A. Peczenik, and G. Sartor, Eds. Springer, 2005, vol. 5.

[6] F. Abate and E. J. Jewell, Eds., *New Oxford American Dictionary*. Oxford University Press, 2001.

[7] G. H. von Wright, *Norm and Action*. London: Routledge, 1963.

[8] T. F. Gordon, G. Governatori, and A. Rotolo, "Rules and norms: Requirements for rule interchange languages in the legal domain," in *Rule Representation, Interchange and Reasoning on the Web*, ser. LNCS, G. Governatori, J. Hall, and A. Paschke, Eds., no. 5858. Berlin: Springer, 5-7 November 2009, pp. 282–296.

[9] G. Governatori and A. Rotolo, "An algorithm for business process compliance," in *Jurix 2008*, G. Sartor, Ed. IOS Press, 2008, pp. 186–191.

[10] ——, "Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations," *Australasian Journal of Logic*, vol. 4, pp. 193–215, 2006.

[11] J. Carmo and A. Jones, "Deontic logic and contrary to duties," in *Handbook of Philosophical Logic, 2nd Edition*, D. Gabbay and F. Guenther, Eds. Kluwer, 2002, pp. 265–343.

[12] G. Governatori, J. Hulstijn, R. Riveret, and A. Rotolo, "Characterising deadlines in temporal modal defeasible logic," in *Australian AI*, ser. LNCS 4830, M. A. Orgun and J. Thornton, Eds. Berlin: Springer, 2007, pp. 486–496.

[13] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, "Representation results for defeasible logic," *ACM Transactions on Computational Logic*, vol. 2, no. 2, pp. 255–287, 2001.

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

[15] G. Governatori and A. Rotolo, "Changing legal systems: Legal abrogations and annulments in defeasible logic," *The Logic Journal of IGPL*, vol. 18, no. 1, pp. 157–194, 2010.

[16] ——, "A conceptually rich model of business process compliance," in *APCCM 2010*, ser. CRPIT, S. Link and A. Ghose, Eds. ACS, 2010.

[17] G. Governatori, "Representing business contracts in RuleML," *International Journal of Cooperative Information Systems*, vol. 14, no. 2-3, pp. 181–216, 2005.

[18] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through sese decomposition," in *ICSOC*, ser. LNCS 4749, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds. Springer, 2007, pp. 43–55.

[19] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Business Process Management Workshops*, ser. LNCS 4103, J. Eder and S. Dustdar, Eds. Springer, 2006, pp. 5–14.

[20] J. M. Küster, K. Ryndina, and H. Gall, "Generation of business process models for object life cycle compliance," in *BPM*, ser. LNCS 4714, G. Alonso, P. Dadam, and M. Rosemann, Eds. Springer, 2007, pp. 165–181.

[21] C. Giblin, S. Müller, and B. Pfitzmann, "From regulatory policies to event monitoring rules: Towards model driven compliance automation," IBM Zurich Lab., Tech. Rep., 2006.

[22] R. Agrawal, C. M. Johnson, J. Kiernan, and F. Leymann, "Taming compliance with Sarbanes-Oxley internal controls using database technology," in *ICDE*, L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang, Eds. IEEE Computer Society, 2006, p. 92.

[23] A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini, "Using the event calculus for tracking the normative state of contracts," *International Journal of Cooperative Information Systems*, vol. 14, pp. 99–129, 2005.

[24] N. Desai, N. C. Narendra, and M. P. Singh, "Checking correctness of business contracts via commitments," in *Proc. AAMAS 2008*, 2008, pp. 787–794.

[25] M. zur Muehlen and M. Rosemann, "Integrating risks in business process models," in *Proc. ACIS 2005*, 2005.

[26] S. Sadiq, G. Governatori, and K. Naimiri, "Modelling of control objectives for business process compliance," in *BPM 2007*, ser. Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., no. 4714. Berlin: Springer, September 25–27 2007, pp. 149–164.

[27] Y. Liu, S. Müller, and K. Xu, "A static compliance-checking framework for business process models," *IBM Systems Journal*, vol. 46, no. 2, pp. 335–362, 2007.

[28] G. Governatori, J. Hoffmann, S. Sadiq, and I. Weber, "Detecting regulatory compliance for business process models through semantic annotations," in *4th International Workshop on Business Process Design*.

[29] G. Governatori and A. Rotolo, "Norm compliance in business process modeling," in *RuleML 2010: 4th International Web Rule Symposium*, ser. LNCS, M. Dean, J. Hall, A. Rotolo, and S. Tabet, Eds., no. 6403. Berlin: Springer, 2010, pp. 194–209.