How Do Agents Comply with Norms?

Guido Governatori[1] and Antonino Rotolo[2]

[1] NICTA, Queensland Research Laboratory
guido.governatori@nicta.com.au
[2] CIRSFID and Law Faculty, University of Bologna
antonino.rotolo@unibo.it

Abstract. The import of the notion of institution in the design of MASs requires to develop formal and efficient methods for modeling the interaction between agents' behaviour and normative systems. This paper discusses how to check whether agents' behaviour is compliant with the rules regulating them. The key point of our approach is that compliance is a relationship between two sets of specifications: the specifications for executing a process and the specifications regulating it. We propose a logic-based formalism for describing both the semantics of normative specifications and the semantics of compliance checking procedures.

1 Introduction

Recent developments in MAS have pointed out that normative concepts can play a crucial role in modeling agents' interaction [29]. In fact, while the main objective is to design systems of autonomous agents, it is likewise important that agent systems may exhibit global desirable properties. Like in human societies, such properties are ensured if the interaction of artificial agents, too, adopts institutional and organizational models whose goal is to regiment agents' behaviour through normative systems in supporting coordination, cooperation and decision-making. However, to keep agents autonomous it is often suggested that norms should not simply work as hard constraints, but rather as soft constraints [6]. In this sense, norms should not limit in advance agents' behaviour, but would instead provide standards which can be violated, even though any violations should result in sanctions or other normative effects applying to non-compliant agents.

If normative systems for MAS are designed as mentioned above, it is of paramount importance to develop mechanisms to characterizing and detecting agents' *norm compliance*. To our knowledge, no systematic investigation has been devoted so far to this research issue in MAS theory, whereas its importance has increased over the last few years in other related fields such as in business modeling. In this perspective, compliance is essentially ensuring that business processes, operations and practise are in accordance with a prescribed and/or agreed set of norms. Compliance is often used to denote adherence of one set of rules (we refer to them as 'source rules' hereafter) against other set of rules (we refer to them as 'target rules' hereafter).

In this paper we apply this interpretation of compliance to discuss adherence or consistence of a set of rules specifying a process against a set of "normative" rules regulating it. Of course, agents' compliance could be tested by directly focusing on plan

Dagstuhl Seminar Proceedings 09121
Normative Multi-Agent Systems
http://drops.dagstuhl.de/opus/volltexte/2009/1909

design and execution. The choice of working on processes is motivated by two reasons. First, modelling agents' behaviour in terms of processes has been proven useful in developing agent-oriented systems for business management (for a recent proposal see, e.g., [5]). The correspondence of business processes and agent plans makes business services flexible and adaptable. Second, while it is far from obvious that complex plan's actions can be always viewed as processes (for the pros and cons of this view, see [9]), in institutional settings agents usually instantiate roles, which consist of a specification of an agent's internal and external behavior. In this sense, taking roles as specific processes (or procedures) allows for obtaining a flexible team agent structure [28]. Under this working hypothesis, the problem of norm compliance can be framed as the relationship between the specifications for process execution and those regulating it.

Process specifications describe how a process is executed while norms state what can be done and what cannot be done by a process. The problem is how to align the language to specify the activities to be performed to complete a process and the conditions set up by the norms relevant for the process. The solution of such a problem is not trivial matter. The detection of violations and the design of agents' compliance amount to relatively affordable operations when we have to check whether processes are compliant with respect to simple normative systems. But things are tremendously harder when we deal with processes to be tested against realistic, large and articulated systems of norms.

What do we mean by a "complex" normative system? Among other things, the complexities of normative systems reside in the fact that they regulate agents's behaviour 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 [7]). 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 agents 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.

In this paper, we take inspiration from an approach originally designed for modeling business process compliance[3]. This approach is based on (semantic) annotations, where the annotations are written in the formal language chosen to represent the normative specifications. The idea is that 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

[3] For a comprehensive exposition of compliance for business process models, see [21, 27]

(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.

The layout of the paper is as follows. Section 2 provides a reasoning mechanism to deal with reparational constructions and to reframe the normative system in such a way as it is possible to detect agent compliance. Section 3 briefly outlines how to represent processes and to annotate them. Section 4 provides a semantics of compliance checking procedures on account of what proposed in the previous sections. A section on related work ends the paper.

2 Normative Constraints for MAS

The expression of violation conditions and the reparation obligations is an important requirement for formalising norms, design 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. This final condition is one which cannot be violated and this it is to be a permission. 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.

2.1 Process Compliance Language (PCL)

We now provide a formal account of the idea presented above. Our formalism, called Process Compliance Language (PCL), is a combination of an efficient non-monotonic formalism (defeasible logic [3, 4]) and a deontic logic of violations [18]. This particular combination allows us to represent exceptions as well as the ability to capture violations and the 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 agents 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. A conceptually sound

formalisation of norms (for assessing the compliance of a process) should take into account all the aspects mentioned above. 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.

Our 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 deontic operators O (for obligation), P (for permission), negation $\neg$ and the non-boolean connective $\otimes$ (for the Contrary-To-Duty (CTD) operator). 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 $Ol_1, \ldots, Ol_n$ are deontic literals and l_{n+1} is a literal, then $Ol_1 \otimes \ldots \otimes Ol_n$ and $Ol_1 \otimes \ldots \otimes Ol_n \otimes Pl_{n+1}$ are $\otimes$-expressions.

The connective $\otimes$ permits combining primary and CTD obligations into unique regulations. The meaning of an expression like $O_sA \otimes O_sB \otimes O_sC$ is that the primary obligation for agent s is A, but if A is not done, then s has the obligation to do B. But if event B fails to be realised, then s has the obligation to do C. Thus B is the reparation of the violation of the obligation O_sA ($\neg A$ holds). Similarly C is the reparation of the obligation O_sB, which is in force when the violation of A occurs.

The formation rules for $\otimes$-expressions allow a permission to occur only at the end of such expressions. This is due to the fact that a permission can be used as a reparation of a violation, but it is not possible to violate a permission, thus it makes no sense to have reparations to permissions.

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 equipped with a superiority relation (a binary relation) over the rule set. 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 $r_1 : A \Rightarrow B \otimes C$ and $r_2 : D \Rightarrow \neg C$. $r_1 \prec r_2$ means that rule r_1 prevails over rule r_2 in situations

where both fire and they are in conflict (i.e., rule r_1 fires for the secondary obligation C). For example let us consider the following two contract rules[4]:

$$r : PremiumCustomer \Rightarrow O_s Discount$$
$$r' : SpecialOrder \Rightarrow O_s \neg Discount$$

saying that Premium Customers are entitled to a discount (r), but there is no discount for goods bought with a special order (r'). Is a Premium customer entitled to a discount when she places a special order? If we only have the two rules above there is no way to solve the conflict just using the contract and there is the need of a domain expert to advise the knowledge engineer about what to do in such case. The logic can only point out that there is a conflict in the contract. On the other hand, if we have an additional provision

$$r'' : PremiumCustomer, \neg Discount \Rightarrow O_s Rebate$$

specifying that if for some reasons a premium customer did not receive a discount then the customer is entitled to a rebate on the next order, then it is possible to solve the conflict, because the contract allows a violation of rule r to be amended by r'', using the merging mechanism we analyse in Section 2.2.

2.2 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 2.2) we describe a mechanism 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 2.2, we examine the problem of redundancies, and we give a condition to identify and remove redundancies from the formal normative specification.

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. In what follows we will first examine some common patterns of this kind of construction and then we will show how to generalise them.

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

$$\Gamma \Rightarrow O_s A.$$

[4] In what follows we will use O_S and P_S for the obligation and permission operators relative to the *Supplier*, and O_P and P_P for the *Purchaser*. O_s and P_s will be used for a generic subject.

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

$$\Delta, \neg A \Rightarrow O_{s'}C,$$

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

$$\frac{\Gamma \Rightarrow O_sA \qquad \Delta, \neg A \Rightarrow O_{s'}C}{\Gamma, \Delta \Rightarrow O_sA \otimes O_{s'}C}$$

This reads as follows: given two policies such that one is a conditional obligation ($\Gamma \Rightarrow O_sA$) and the antecedent of second contains the negation of the propositional content of the consequent of the first ($\Delta, \neg A \Rightarrow O_{s'}C$), 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., $\Gamma, \Delta \Rightarrow O_sA \otimes O_{s'}C$) 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.

Suppose we have the following rules

$$r : \mathit{Invoice} \Rightarrow O_P\mathit{PayWithin7Days}$$
$$r' : \neg\mathit{PayWithin7Days} \Rightarrow O_P\mathit{PayWithInterest}.$$

From these we obtain

$$r'' : \mathit{Invoice} \Rightarrow O_P\mathit{PayWithin7Days} \otimes O_P\mathit{PayWithInterest}.$$

We can also generate chains of CTDs in order to deal iteratively with violations of reparational obligations. The following case is just an example of this process.

$$\frac{\Gamma \Rightarrow O_sA \otimes O_sB \qquad \neg A, \neg B \Rightarrow O_sC}{\Gamma \Rightarrow O_sA \otimes O_sB \otimes O_sC}$$

For example, from the rules

$$r : \mathit{Invoice} \Rightarrow O_S\mathit{QualityOfService} \otimes O_S\mathit{Replace3days}$$
$$r' : \neg\mathit{QualityOfService}, \neg\mathit{Replace3days} \Rightarrow O_S\mathit{Refund\&Penalty}$$

we derive the new rule

$$r'' : \mathit{Invoice} \Rightarrow O_S\mathit{QualityOfService} \otimes O_S\mathit{Replace3days} \otimes O_S\mathit{Refund\&Penalty}.$$

The above patterns are just special instances of the general mechanism described in details in [18, 15].

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 some examples and then we show how to give a formal definition of the notion of subsumption appropriate for PCL.

Let us consider the rules

$$r : \mathit{Service} \Rightarrow O_S\mathit{QualityOfService} \otimes O_S\mathit{Replace3days} \otimes O_S\mathit{Refund\&Penalty},$$
$$r' : \mathit{Service} \Rightarrow O_S\mathit{QualityOfService} \otimes O_S\mathit{Replace3days}.$$

The first rule, r, subsumes the second r'. Both rules state that after the supplier has provided the service she has the obligation to provide the service according to the published standards, if she violates such an obligation, then the violation of *QualityOfService* can be repaired by replacing the faulty service within three days (O_S*Replace3days*). In other words O_S*Replace3days* is a secondary obligation arising from the violation of the primary obligation O_S*QualityOfService*. In addition r prescribes that the violation of the secondary obligation O_S*Replace3days* can be repaired by O_S*Refund&Penalty*, i.e., the seller has to refund the buyer and in addition she has to pay a penalty.

The conditions of a normative system cannot be taken in isolation insofar as they exist in it. Consequently the whole normative system determines the meaning of each single clause (norm). In agreement with this holistic view of norms we have that the normative content of r' is included in that of r. Accordingly r' does not add any new piece of information, it is redundant and can be dispensed from the explicit formulation of the norms.

Another common case is exemplified by the rules:

$$r : \mathit{Invoice} \Rightarrow O_P\mathit{PayWithin7Days} \otimes O_P\mathit{PayWithInterest}$$
$$r' : \mathit{Invoice}, \neg\mathit{PayWithin7Days} \Rightarrow O_P\mathit{PayWithInterest}.$$

The first rule says that after the seller sends the invoice the buyer has one week to pay, otherwise the buyer has to pay the principal plus the interest. Thus we have the primary obligation O_P*PayWithin7Days*, whose violation is repaired by the secondary obligation O_P*PayWithInterest*, while, according to the second rule, given the same set of circumstances *Invoice* and $\neg$*PayWithin7Days* we have the primary obligation O_P*PayWithInterest*. However, the primary obligation of r' obtains when we have a violation of the primary obligation of r. Thus the condition of applicability of the second rule includes that of the first rule, which then is more general than the second and we can discard r' from the formal representation of the specifications.

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

Definition 1. *Let* $r_1 : \Gamma \Rightarrow A \otimes B \otimes C$ *and* $r_2 : \Delta \Rightarrow D$ *be two rules, where* $A = \bigotimes_{i=1}^{m} A_i$, $B = \bigotimes_{i=1}^{n} B_i$ *and* $C = \bigotimes_{i=1}^{p} C_i$. *Then* r_1 subsumes r_2 *iff (1)* $\Gamma = \Delta$ *and* $D = A$; *or (2)* $\Gamma \cup \{\neg A_1, \ldots, \neg A_m\} = \Delta$ *and* $D = B$; *or (3)* $\Gamma \cup \{\neg B_1, \ldots, \neg B_n\} = \Delta$ *and* $D = A \otimes \bigotimes_{i=0}^{k \leq p} C_i$.

The intuitions 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.

Conflicts often arise in normative systems. What we have to determine is whether we have genuine conflicts, i.e., the norms are in some way flawed or whether we have *prima-facie* conflicts. A prima-facie conflict is an apparent conflict that can be resolved when we consider it in the context where it occurs and if we add more information the conflict disappears.

The following rule is devised for making explicit conflicting norms (contradictory norms) within the system:

$$\frac{\Gamma \Rightarrow A \qquad \Delta \Rightarrow \neg A}{\Gamma, \Delta \Rightarrow \bot} \tag{1}$$

where

1. there is no rule $\Gamma' \Rightarrow X$ such that either $\neg A \in \Gamma'$ or $X = A \otimes B$;
2. there is no conditional rules $\Delta' \Rightarrow X$ such that either $A \in \Delta'$ or $X = \neg A \otimes B$;
3. for any formula B, $\{B, \neg B\} \not\subseteq \Gamma \cup \Delta$.

The meaning of these three conditions is that given two rules, we have a conflict if the normative content of the two rules is opposite, such that none of them can be repaired, and the states of affairs/preconditions they require are consistent.

Once conflicts have been detected there are several ways to deal with them. The first thing to do is to determine whether we have a *prima-facie* conflict or a genuine conflict. As we have seen we have a conflict when we have two rules with opposite conclusions. Thus a possible way to solve the conflict is to create a superiority relation over the rules and to use it do "defeat" the weaker rule. In Section 2.3 we will examine how to reason with norms, and we will see how to use the superiority relation to solve conflicts.

Normalisation Process We now describe how to use the machinery presented in Section 2.2 and Section 2.2 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:

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 use the conflict identification rule to label and detect 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 [18], thus according to standard set theory results the fixed-point exists and it is unique. However, we have to be careful since merging first and doing subsumption after produces different results from the opposite order (i.e., subsumption first and merging after), or by interleaving the two operations.

2.3 Reasoning with Norms

In the previous section we have examined the mechanism to obtain a set of rules covering all possible (explicit) norms for obligations, permissions and prohibitions that can arise from an initial set of norms. In this section we focus on the issue of how to determine what obligations are in force for a specific situation, thus taking the well known distinction between schema and instance. The previous section defines the procedure to obtain the full (normalised) schema corresponding to a normative system. Here we study how to get the normative positions active for a specific instance of a process. The reasoning mechanism of PCL is an extension of Defeasible Logic.

Defeasible logic [24] is a simple and efficient rule based non-monotonic formalism. Over the year the logic has been developed and extended, and several variants have been proposed to model different aspects of normative reasoning and encompassed other formalisms to for normative reasoning.

The main intuition of the logic is to be able to derive "plausible" conclusions from partial and sometimes conflicting information. Conclusions are *tentative* conclusions, in the sense that a conclusion can be withdrawn when we have new pieces of information.

The knowledge in a Defeasible Theory is organised in *facts* and *rules* and *superiority relation*. Facts are indisputable statements. Defeasible rules are rules that can be defeated by contrary evidence. The superiority relation is a binary relation defined over the set of rules. The superiority relation determines the relative strength of two (conflicting) rules. The meaning of a defeasible rule, like $A_1,\ldots,A_n \Rightarrow C$, is that normally we are allowed to derive C given $A_1,\ldots,A_n$, unless we have some reasons to support the opposite conclusion (i.e., we have a rule like $B_1,\ldots,B_m \Rightarrow \neg C$).

Defeasible Logic is a "skeptical" non-monotonic logic, meaning that it does not support contradictory conclusions. Instead Defeasible Logic seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding $\neg A$, Defeasible Logic does not conclude either of them (thus the name skeptical). If the support for A has priority over the support for $\neg A$ then A is concluded.

A defeasible conclusion is a tentative conclusion that might be withdrawn by new pieces of information, or in other terms it is the 'best' conclusion we can reach with the given information. In addition the logic is able to tell whether a conclusion is or is not provable. Thus it is possible to have the following types of conclusions: (a) Positive defeasible conclusions, meaning that the conclusions can be defeasible proved; (b) Negative defeasible conclusions, meaning that one can show that the conclusion is not even defeasibly provable. A defeasible conclusion A can be derived if there is a rule whose conclusion is A, whose prerequisites (antecedent) have either already been proved or given in the case at hand (i.e., facts), and any stronger rule whose conclusion is $\neg A$ (the negation of A) has prerequisites that fail to be derived. In other words, a conclusion A is (defeasibly) derivable when: (1) A is a fact; or (2) there is an applicable defeasible rule for A, and either (2.1) all the rules for $\neg A$ are discarded (i.e., not applicable) or (2.2) every applicable rule for $\neg A$ is weaker than an applicable strict or defeasible rule for A. A rule is applicable if all elements in the body of the rule are derivable (i.e., all the premises are positively provable), and a rule is discarded if at least one of the elements of the body is not provable (or it is a negative defeasible conclusion).

Defeasible Logic at Work We illustrate the inferential mechanism of Defeasible Logic with the help of an example. Let us assume we have a theory containing the following rules:

$$r_1 : PremiumCustomer(X) \Rightarrow Discount(X)$$
$$r_2 : SpecialOrder(X) \Rightarrow \neg Discount(X)$$
$$r_3 : Promotion(X) \Rightarrow \neg Discount(X)$$

where the superiority relation is thus defined: $r_1 \prec r_3$ and $r_2 \prec r_1$. The theory states that services in promotion are not discounted, and so are special orders with the exception of special orders placed by premium customers, who are normally entitled to a discount.

In a scenario where all we have is that we received a special order, then we can conclude that the price has to be calculated without a discount since rule r_1 is not applicable (we do not know whether the customer is a premium customer or not). In case the special order is received from a special customer for a service not in promotion, we can derive that the customer is entitled to a discount. Indeed rule r_1 is now applicable and it is stronger than rule r_2, and r_3, which is stronger than r_1, is not applicable (i.e., the service is not in promotion).

Adding Reparation Chains PCL is an extension of defeasible logic with the reparation operator ($\otimes$). Accordingly the reasoning mechanism to derive conclusion is an extension of that for defeasible logic. In defeasible logic the conclusions of a rule is a single literal and not a reparation chain. Thus the condition that OA appears in the conclusion of a rule means in defeasible logic that OA is the conclusions of the rule. For PCL have to extend the notion to accommodate reparation chain. The required change is that to prove OA, we have to consider all rules with a reparation chain for OA, where for all elements before OA in the chain, the negation of the element is already provable. Thus to prove OA given a rule $P_1, \ldots, P_n \Rightarrow OC_1 \otimes \cdots \otimes OC_m \otimes OA \otimes OD_1 \otimes \cdots \otimes OD_k$, we have that $P_1, \ldots, P_n$ must be all provable, and so must be $\neg C_1, \ldots, \neg C_m$ [15].

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

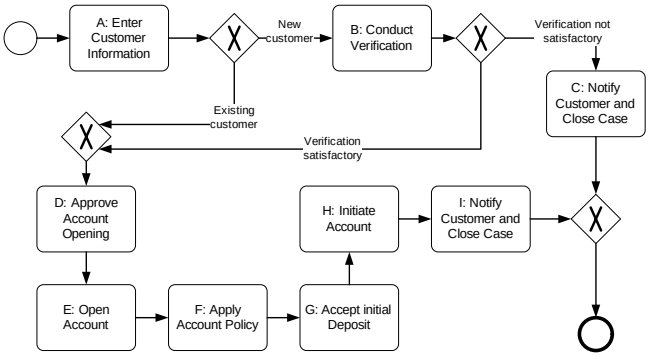

Fig. 1. Example account opening process in private banking

extend the execution semantics for process models given by [30] 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 a XOR (Parallel) split node consumes the token on its incoming edge and produces a token on one (all) of its outgoing edges, whereas a 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 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 1), 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).

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 1 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 1. 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.

$$r_1 : newCustomer(x) \Rightarrow OcheckIdentity(x)$$

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.

– Retain history of identity checks performed.

$$r_2 : checkIdentity(x) \Rightarrow OrecordIdentity(x)$$

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.

$$r_3 : account(y) \Rightarrow OpositiveBalance(y) \otimes OapproveManager(y)$$

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.

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

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 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 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.

Then, 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 $t \prec s$.

Let us examine the following example. Consider the rules

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

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.

The procedure for compliance checking is based on two algorithms, *ComputeObligations* and *CheckCompliance*. *ComputeObligations* is the procedure given in the Section 4.1 to compute the set of active chains. Given a set of literals S, corresponding to effects of a task T in a process model, we use the algorithm *ComputeObligations* to determine the current set of active chains for the process *Current*. The set of the current active chains includes the new chains triggered by the task, as well as the chains carried out from previous tasks. The algorithm *CheckCompliance* scans all elements of *Current* against the set of literals S, and determines the state of each reparation chain ($C = A_1 \otimes A_2$) in *Current*. *CheckCompliance* operates as follows:

if $A_1 = OB$, 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 $[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2] \in$ *Violated* then
      add($[T, R, B_1 \otimes B_2 \otimes A_1 \otimes A_2]$, *Compensated*)
  if $\neg B \in S$, then
    add($[T, R, A_1 \otimes A_2]$, *Violated*)
    add($[T, R, A_2]$, *Current*)
  else
    add($[T, R, A_1 \otimes A_2]$, *Unfulfilled*).

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 OB, 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 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*). In case of a violation, we add the information about it in the system. This is done by inserting 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 does not discharge an obligation, thus we do not remove the chain from the set of active chains[5]. Finally in the last case, the set of effects does not tell us if the obligation has been

[5] [16] propose a more fine grained classification of obligations, accordingly it is possible to have obligations that are discharged when are violated, as well as obligations that persist in case of

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.

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.

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

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

$$r_3 : account(y) \Rightarrow OpositiveBalance(y) \otimes OapproveManager(y)$$

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

$$[E, r_4, OpositiveBalance(y) \otimes OapproveManager(y)]$$

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.[6]

5 Related Work

This paper provides a means of investigate the impact of compliance controls on agents' process and of assisting in compliance checking, analysis and feedback for subsequent

a violation. The above algorithm can be easily modified to deal with the different types of obligations examined by [16].

[6] What about a situation where, after task F, we have a task producing the annotation $approveManager(x)$ but no task with effect $positiveBalance(x)$? Is the resulting process compliant? In this case we have the reparation of the violation, but not the violation. The issue here is that we could have that a sanction is enforced before the occurrence of the violation which the sanction was supposed to compensate. Thus we are in a situation similar to that described in footnote 5 where the way to address the issue depends on the types of the obligations we have to deal with. Anyway, (i) it is easy to modify algorithm *CheckCompliance* to account for this type of cases, (ii) if one accepts preemptive reparations one can change the definition that classifies a process as compliant by replacing the condition that $Unfulfilled = \emptyset$ with the condition: let S be the set of effects for the end task of a process, $\forall [T,R,OA_1 \otimes \cdots \otimes OA_n] \in Unfulfilled$, $\exists A_i \in S$.

(re)design of the processes. The procedure is based on efficient algorithms and is able to deal with reparation chains of deontic statements.

Research on compliance has carried out in the field of autonomous agents [2], but the majority of works are found in related areas, in particular on control modelling. [14] presents the logical language PENELOPE, that provides the ability to verify temporal constraints arising from compliance requirements on effected business processes. [22] develops a method to check compliance between object lifecycles that provide reference models for data artifacts e.g. insurance claims and business process models. [13] 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, [1] 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.

Another line of investigation studies compliance based on the structure of business processes. [12] 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. [8], on the other hand, investigate compliance in the context of agents and multi-agent systems based on a classification of paths of tasks. [25] proposed Concurrent Transaction Logic to model the states of a workflow and presented some algorithms to determine whether the workflow is compliant. [31] proposes a polynomial time algorithm to perform compliance checking of business processes. The algorithm propagates the effects of tasks from one task to the tasks following it. Norms are represented as logical clauses. The major limitation of these approaches to compliance is that they ignore the normative aspects of compliance.

There has been some complementary work in the field. analysis of formal models representing normative notions. [11] studies the performance of business contract based on their formal representation in event calculus. [10] 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.

PCL has already been proposed to study compliance. [17] uses it to model business contracts and their compliance with BPMN process models based on the ideal semantics of [18]. In [19] we extend the work of [31] to check compliance against PCL representation of the norms a process has to comply with. The focus is on the propagation of effects and normative positions across tasks. The propagation algorithm is based on heuristic, and thus it gives an approximate compliance checking.

Also, there have been recently some efforts towards support for process modelling against compliance requirements. [32] 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. [23] 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.

Acknowledgements

This paper extends the work presented in [20].

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. R. Agrawal, C. M. Johnson, J. Kiernan, and F. Leymann. Taming compliance with Sarbanes-Oxley internal controls using database technology. In *Proc. ICDE 2006*, 2006.
2. M. Alberti, M. Gavanelli, E. Lamma, F. Chesani, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based software tool. *Applied Artificial Intelligence*, 20(2-4):133–157, 2006.
3. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, 2001.
4. Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming*, 6(6):703–735, 2006.
5. W. Binder and et al. A multiagent system for the reliable execution of automatically composed ad-hoc processes. *JAAMAS*, 2006.
6. G. Boella and L. van der Torre. Fulfilling or violating obligations in multiagent systems. In *Procs. IAT04*, 2004.
7. J. Carmo and A.J.I. Jones. Deontic logic and contrary to duties. In D.M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, 2nd Edition*. Kluwer, 2002.
8. Amit K. Chopra and Munindar P. Sing. Producing compliant interactions: Conformance, coverage and interoperability. In *Declarative Agent Languages and Technologies IV*, pages 1–15, 2007.
9. F. de Jonge, N. Roos, and C. Witteeven. Primary and secondary diagnosis of multi-agent plan execution. *JAAMAS*, 2008.
10. N. Desai, N. C. Narendra, and M. P. Singh. Checking correctness of business contracts via commitments. In *Proc. AAMAS 2008*, 2008.
11. 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*, 14, 2005.
12. Aditya Ghose and George Koliadis. Auditing business process compliance. In *Service Oriented Computing, ISOC 2007*, LNCS, pages 169–180. Springer, 2007.
13. C. Giblin, S. Müller, and B. Pfitzmann. From regulatory policies to event monitoring rules: Towards model driven compliance automation. Technical report, IBM Zurich Research Lab., 2006.
14. S. Goedertier and J. Vanthienen. Designing compliant business processes with obligations and permissions. In *Business Process Management (BPM) Workshops*, 2006.
15. G. Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, 2005.

16. G. Governatori, J. Hulstijn, R. Riveret, and A. Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proc. Australian AI 2007*, 2007.
17. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *Proc. EDOC 2006*, 2006.
18. G. Governatori and A. Rotolo. Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215, 2006.
19. Guido Governatori, Jörg Hoffmann, Shazia Sadiq, and Ingo Weber. Detecting regulatory compliance for business process models through semantic annotations. In *4th International Workshop on Business Process Design*.
20. Guido Governatori and Antonino Rotolo. An algorithm for business process compliance. In Enrico Francesconi, Giovani Sartor, and Daniela Tiscornia, editors, *Legal Knowledge and Information Systems*, pages 186–191. IOS Press, 2008.
21. Guido Governatori and Shazia Sadiq. The journey to business process compliance. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on BPM*, chapter 20, pages 429–457. IGI Global, 2009.
22. J. M. Küster, K. Ryndina, and H. Gall. Generation of business process models for object life cycle compliance. In *Proc. BPM 2007*, 2007.
23. Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Systems Journal*, 46(2):335–362, 2007.
24. Donald Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*. 1994.
25. Dumitru Roman and Michael Kifer. Reasoning about the behaviour of semantic web services with concurrent transaction logic. In *VLDB*, pages 627–638, 2007.
26. S. Sadiq, G. Governatori, and K. Naimiri. Modelling of control objectives for business process compliance. In *Proc. BPM 2007*, 2007.
27. Shazia Sadiq and Guido Governatori. A methodological framework for aligning business processes and regulatory compliance. In Jan van Brocke and Michael Rosemann, editors, *Handbook of Business Process Management*. Springer, 2009.
28. P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999.
29. L. van der Torre, G. Boella, and H. Verhagen, editors. *Normative Multi-agent Systems*, Special Issue of *JAAMAS*, vol. 17(1), 2008.
30. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition. In *Proc. ICSOC 2007*, 2007.
31. Ingo Weber, Guido Governatori, and Jörg Hoffmann. Approximate compliance checking for annotated process models. In Marta Indulska, Shazia Sadiq, and Michael zur Muehlen, editors, *Proceedings of the 1st International Workshop on Governance, Risk and Compliance — Applications in Information Systems (GRCIS'08)*, volume 339, pages 46–60. CEUR Workshop Proceedings, 17 June 2008.
32. M. zur Muehlen and M. Rosemann. Integrating risks in business process models. In *Proc. ACIS 2005*, 2005.