2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops

How Do Agents Comply with Norms?

Guido Governatori
NICTA, Queensland Research Laboratory, Brisbane, Australia

Antonino Rotolo
CIRSFID and Law Faculty, University of Bologna, Italy

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 complies 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 formalism for describing both the semantics of normative specifications and the semantics of compliance checking procedures.**

I. Introduction

Recent developments in MAS have pointed out that normative concepts can play a crucial role in modeling agents' interaction [15], [6]. 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 [4]. 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 against other set of rules.

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 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., [3]). 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 [7]), 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 [14]. 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 a 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 [5]). 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 [10].

978-0-7695-3801-3/09 $26.00 © 2009 IEEE
DOI 10.1109/WI-IAT.2009.332

This approach is based on (semantic) annotations in the same formal language as that of the normative specifications. The idea is that processes are annotated and the annotations provide the conditions a process has to comply with.[1]

II. Normative Constraints for MAS

The expression of violation conditions and the reparation obligations are important requirements for formalising norms, designing subsequent processes to minimise or dealing with such violations and also determining 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.

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 [1], [2]) and a deontic logic of violations [11]. 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 is computational feasible: the extension of a theory can be computed in time linear to the size of the theory.

The ability to handle violations 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 PCL are constructed using the deontic operators O (for obligation), P (for permission), negation $\neg$ and the non-boolean connective $\otimes$ (for the CTD operator) 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. 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 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; e.g., given $r_1 : A \Rightarrow OB \otimes OC$ and $r_2 : D \Rightarrow O\neg C$, $r_1 \prec r_2$ means that rule r_1 prevails over rule r_2 in situations where both fire and they are in conflict (i.e., r_1 fires for the secondary obligation OC).

Given the structure of rules in PCL, it is possible that the (normative) meaning of a rule is included in another rule. Consider for example the rules $A \Rightarrow OB \otimes OC$ and $A \Rightarrow OB$. Intuitively the first rule subsumes the second one, as the behaviour (or the normative content) of the second rule is implied by the first rule, and thus the second rule does not add anything new to the system and it can be safely discarded. For a comprehensive presentation of PCL, see [8].

III. 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 BPMs usually has at least 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

[1]For a comprehensive exposition of compliance for business process models, see [13]

and (x)or-join– (at least (most) one task in a set of task must be executed). The basic execution semantics of the control flow aspect of a business process model is defined using token-passing mechanisms, as in Petri Nets [16].

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.

We extend BPMs with sets of annotations, where the annotations describe (i) the artifacts or effects of executing tasks in a process 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 over which logical statements can be made, in the form of literals involving these variables. The task nodes can be annotated using *effects* which are conjunctions of literals using the process variables. The meaning is that, if executed, a task changes the state of the world according to its effect: every literal mentioned by the effect is true in the resulting world; if a literal l was true before, and is not contradicted by the effect, then it is still true (i.e., the world does not change of its own accord).

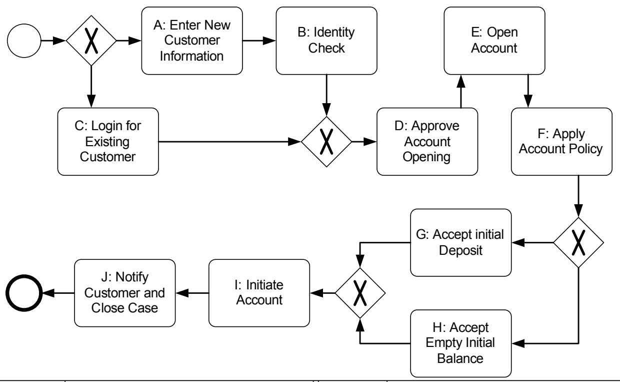
IV. Compliance Checking

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 III.
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. 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 its effects and the obligations accumulated up to it. 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 (see [13]).

A reparation chain is in force if there is a rule whose consequent is this reparation chain and a set of facts (effects of a task in a process) includes 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). Therefore it is not possible to have two instances at the same time of the same reparation chain. 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.


A: Enter New Customer Information
B: Identity Check
E: Open Account
C: Login for Existing Customer
D: Approve Account Opening
F: Apply Account Policy
G: Accept initial Deposit
J: Notify Customer and Close Case
I: Initiate Account
H: Accept Empty Initial Balance

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	$notify(x,y)$

$$r_1 : newCustomer(x) \Rightarrow OcheckIdentity(x)$$
$$r_2 : checkIdentity(x) \Rightarrow OrecordIdentity(x)$$
$$r_3 : account(y) \Rightarrow OpositiveBalance(y) \otimes OapproveManager(y)$$
$$r_4 : account(x), owner(x,y), accountType(x,VIP) \Rightarrow P\neg positiveBalance(x)$$

Fig. 1. Business Process, Annotations and Rules

The procedure for compliance checking is based on two algorithms, *ComputeObligations* and *CheckCompliance*. *ComputeObligations* is a straightforward extension of defeasible logic conditions to compute the extension of a theory [8]. Given a set of literals S, the effects of a task T, *ComputeObligations* determines the current set *Current* of active chains for the process. *Current* 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 (B and $\neg B$ are not 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[2]. Finally in the last case, 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.

The conditions below relate the state of a process based as reported by the *CheckCompliance* algorithm and the semantics for PCL expressions. 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). 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.

Acknowledgements

This paper extends the work presented in [12].

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

[2] G. Antoniou, D. Billington, G. Governatori, and M.J. Maher. Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming*, 6(6):703–735, 2006.

[3] W. Binder and et al. A multiagent system for the reliable execution of automatically composed ad-hoc processes. *JAAMAS*, 2006.

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

[5] 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.

[6] M. Dastani, Davide Grossi, John-Jules Ch. Meyer, and Nick Tinnemeier. Normative multi-agent programs and their logics. In Rafael Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems*, number 08361 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[7] F. de Jonge, N. Roos, and C. Witteveen. Primary and secondary diagnosis of multi-agent plan execution. *JAAMAS*, 2008.

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

[9] G. Governatori, J. Hulstijn, R. Riveret, and A. Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proc. Australian AI 2007*, 2007.

[10] G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *Proc. EDOC 2006*, 2006.

[11] 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.

[12] G. Governatori and A. Rotolo. An algorithm for business process compliance. In Giovani Sartor, editor, *Jurix 2008*, pages 186–191. IOS Press, 2008.

[13] G. Governatori and S. Sadiq. The journey to business process compliance. In H. Cardoso and W. van der Aalst, editors, *Handbook of Research on BPM*, pages 426–454. IGI Global, 2009.

[14] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999.

[15] L. van der Torre, G. Boella, and H. Verhagen, editors. *Normative Multi-agent Systems*, Special Issue of *JAAMAS*, vol. 17(1), 2008.

[16] 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.

[2][9] proposed 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 a violation. The algorithm can be easily modified to deal with the types of obligations examined by [9].