Legal Knowledge and Information Systems (Jurix 2008)
E. Francesconi et al. (eds)
IOS Press, 2008, 2003

An Algorithm for Business Process Compliance

Guido Governatori [a] and Antonino Rotolo [b]
[a] *NICTA, Queensland Research Laboratory*
[b] *CIRSFID, University of Bologna*

Abstract. This paper provides a novel mechanism to check whether business processes are compliant with business rules regulating them. The key point is that compliance is a relationship between two sets of specifications: the specifications for executing a business process and the specifications regulating it.

1. Introduction

By business process compliance we mean the adherence or consistence of a set of specifications modelling a business process and a set of specifications modelling the norms for a particular business [3]. Business 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 to how to align the language to specify the activities to be performed to complete a business process and the conditions set up by the norms for the process.

2. Modelling Norms

We represent norms as rules in FCL [1], where an FCL rule is an expression $r : A_1, \dots, A_n \Rightarrow B$, where r is the (unique) name of the rule, $A_1, \dots, A_n$ are the *premises* (propositions in the logic), and B is the *conclusion* (also a proposition of the logic). The propositions of the logic are built from a finite set of atomic propositions, and the following operators: $\neg$(negation), O (obligation), P (permission), and $\otimes$ (violation/reparation). 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 p is an atomic proposition, then $\neg p$ is a proposition. Given a proposition p we use $\sim p$ to denote the complement of p: i.e., if $p = l$, then $\sim p = \neg l$ and if $p = \neg l$, then $\sim p = l$. If p is a proposition, then Op is an *obligation proposition* and Pp is a *permission proposition*; both are called *deontic propositions*. If $p_1, \dots, p_n$ are obligation propositions and q is a deontic proposition, then $p_1 \otimes \cdots \otimes p_n \otimes q$ is a *reparation chain*. Given a reparation chain C, we use $\pi_i(C)$ to denote the i-th element of the chain. The meaning of an expression like $Op \otimes Oq$ is that p is obligatory, but if it is violated (i.e., we have $\neg p$) then q is obligatory.

FCL 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).

3. Process Modelling

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

3.1. Execution Semantics

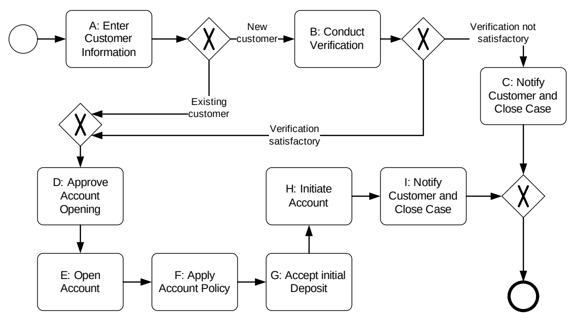
The basic execution semantics of the control flow aspect of a business process model is defined using token-passing mechanisms, as in Petri Nets. The definitions used here extend the execution semantics for process models given by [5] with semantic annotations in the form of effects and their meaning.

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 the process and (ii) the rules describing the obligations (and other normative positions) relevant for the process [4].

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 the annotation table in Figure 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


A: Enter Customer Information
New customer
B: Conduct Verification
Verification not satisfactory
C: Notify Customer and Close Case
Existing customer
Verification satisfactory
D: Approve Account Opening
H: Initiate Account
I: Notify Customer and Close Case
E: Open Account
F: Apply Account Policy
G: Accept initial Deposit

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	$notify(x,y)$

Figure 1. Example account opening process in private banking

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 annotated process in Figure 1 models an account opening process in private banking. In 2006 a new legislative framework, i.e., *Anti-Money Laundering and Counter-Terrorism Financing Act 2006* (AML/CTF), was introduced in Australia for anti-money laundering. The AML/CTF act imposes a number of obligations, which include: customer due diligence, reporting and record keeping. AML/CTF is a principles or risk based regulation and hence businesses need to determine the exact manner in which they will fulfil the obligations. Suppose the follwing rules have been put in place by a business to implement the act.

- All new customers must be scanned against provided databases for identity checks.
$$r_1 : newCustomer(x) \Rightarrow OcheckIdentity(x)$$
- Retain history of identity checks performed.
$$r_2 : checkIdentity(x) \Rightarrow OrecordIdentity(x)$$
- Accounts must maintain a positive balance, unless approved by a bank manager, or for VIP customers.
$$r_3 : account(y) \Rightarrow OpositiveBalance(y) \otimes OapproveManager(y)$$
$$r_4 : account(x), owner(x,y), accountType(x,VIP) \Rightarrow P\neg positiveBalance(x)$$

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, FCL constraint expressions for a normative document define a behavioural and state space which can be used to analyse how well different behaviour execution paths of a business process comply with the FCL constraints. Our aim is to use this analysis as a basis for deciding whether execution paths of a business process are compliant with the FCL and thus with the normative document modelled by the FCL specifications. To this end we use the following procedure:

1. We traverse the graph describing the business 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 theory resulting from the effects and the FCL 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 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 FCL to compute the set of conclusions (obligations) if 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 $A \Rightarrow OB \otimes OC$ and the effects are A and $\neg B$. The only obligation in force for this scenario is OC. Since we have a violation of the rule, 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 business process. What we have to do is to identify the chain for the ideal situation for the task at hand. To deal with this issue we have to identify the *active* reparation chains.

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 is 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 has two steps (see [1]). In the first step, given a set of literals S, corresponding to effects of a task T in a process model, we identify the set of active chains for the process *Current*. 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 ($C = A_1 \otimes A_2$) 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 is 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[1]. 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 define the state of a process based on the *CheckCompliance* algorithm. A process is compliant if there are no violations or there are violations but these have been compensated for; a process is fully compliant if there are no violations.

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

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

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

relative to the annotated process of Figure 1. After task E we have, among others, the effect $\mathit{account}(y)$. This means that after task E we have the chain

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

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.

References

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

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

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

[4] S. Sadiq, G. Governatori, and K. Naimiri. Modelling of control objectives for business process compliance. In *Proc. BPM 2007*, 2007.

[5] 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.

[1] [2] 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 a violation. The above algorithm can be easily modified to deal with the different types of obligations examined by [2].