Capturing Variability of Law with *Nòmos* 2

Alberto Siena[1], Ivan Jureta[2], Silvia Ingolfo[1], Angelo Susi[3], Anna Perini[3], John Mylopoulos[1]

[1] University of Trento, via Sommarive 14, Trento, Italy
{a.siena, silvia.ingolfo, jm}@unitn.it
[2] University of Namur, 8, rempart de la vierge, 5000 Namur, Belgium
ivan.jureta@fundp.ac.be
[3] FBK-Irst, via Sommarive 18, Trento, Italy
{susi, perini}@fbk.eu

Abstract. Regulatory compliance is increasingly viewed as an essential element of requirements engineering. Laws, but also regulations and policies, frame their provisions through complex structures made of conditions, derogations, exceptions, which together generate a high number of alternative compliance solutions. This paper addresses the problem of modeling, exploring and selecting among alternatives in a variability space defined by laws. Our proposal includes a conceptual modeling framework for laws and reasoning techniques, called *Nòmos* 2. The proposal is evaluated with a fragment of the Health Insurance Portability and Accountability Act (HIPAA).

Keywords: requirement engineering, variability, regulatory compliance

1 Introduction

Socio-technical systems – complex systems consisting of software, human and organizational actors as well as business processes, running on an open network of hardware nodes – are increasingly gaining attention of governmental agencies for their tremendous potential to fulfill social functions but also the risks they introduce in case of mishap. Legislators are challenged to lay down comprehensive laws, abstract enough to avoid dependance on specific technological configurations, but able to foresee conceivable adverse conditions and prevent possible workarounds. The challenge for organizations is to understand and analyze the various ways their business goals can be achieved, while complying with applicable laws. The cost to ensure compliance is high: in the financial domain the U.S. Government Accountability Office estimates a $2.9 billion expense over five years[1] to implement the Wall Street Reform and Consumer Protection Act; in the Healthcare domain, organizations have spent $17.6 billion over a number of years[2] to align their systems and procedures with a single law, the Health Insurance Portability and Accountability Act (HIPAA).

The cost of ensuring compliance can be mitigated by understanding better variability in laws (including regulations and policies). For an organization, this means understanding that there are various ways to comply with the same law, and each of them can result

[1] http://blogs.wsj.com/economics/2011/03/28/gao-implementing-dodd-frank-could-cost-2-9-billion/

[2] Medical privacy - national standards to protect the privacy of personal health information. Office for Civil Rights, US Department of Health and Human Services, 2000.

in different costs. The purpose of this paper is to provide a language for modeling and analyzing variability in laws. A law consists of atomic elements (clauses) together with *conditions for their applicability*. A clause generally applies to a legal subject *if and only if the legal subject finds herself in a situation that satisfies the applicability conditions of the clause*. The legal subject *complies* with a clause if that clause applies to that legal subject and the subject satisfies that clause. The observation that clauses are conditional has an important consequence on the design of methods for assuring the compliance of requirements to laws. Namely, *by choosing the requirements that the system-to-be should satisfy, the requirements engineer chooses the conditions that the system-to-be will satisfy, and therefore also the clauses that the system-to-be should comply to*. Not all systems-to-be are therefore subject to the same clauses. And this is not only because they have different purposes, but also because the same purpose can be achieved in different ways. Each of these alternative sets of requirements that a system-to-be should satisfy will result in different properties and behavior of the system-to-be, and thus a different set of clauses which the system-to-be must comply to. Separating the identification, analysis and selection of requirements from the analysis of laws is therefore a mistake.

Choosing requirements should be done together with the identification and evaluation of the consequences of these choices on the *applicability* of clauses (i.e., whether the system-to-be should comply with the clause) and *satisfiability* of clauses (whether the system-to-be satisfies applicable clauses). In other words, the problem and solution space to explore during Requirements Engineering (RE) should also include representations of antecedents (i.e., preconditions) and consequents (postconditions) of clauses and analysis of requirements and laws should help evaluate both applicability and satisfiability of clauses. This is done by answering questions such as:

1. Given a law, containing a set of clauses articulated though conditions, exceptions and so on, which of them apply to a given requirements model?
2. Given a requirement, how to select a compliant way to achieve it, that is, a way which satisfies all applicable clauses?

This paper extends the *Nòmos* framework [1] for representing legal knowledge in requirements engineering, to (i) include antecedents and consequents of clauses in models of law, and (ii) use this information to evaluate the applicability and satisfiability of clauses for given sets of requirements, and thereby the compliance of these requirements. This new framework is called *Nòmos* 2.

The rest of the paper is organized as follows. We introduce the concepts and relations used in *Nòmos* 2, i.e., the ontology of the framework (§2). We define a modeling language used to represent the instances of the concepts and relations, i.e., for creating models of norms (§3). We define the algorithms for deciding about the applicability and satisfiability of norms (§4). We illustrate models using HIPAA norms (§5). We end with a discussion of related work (§6) and a summary of limitations and conclusions (§7).

2 Primitive Concepts and Relationships

The ontology of our proposed framework is founded on the assumption that *legal clauses are not requirements*. Stakeholders may or may not want compliance with a law, but they

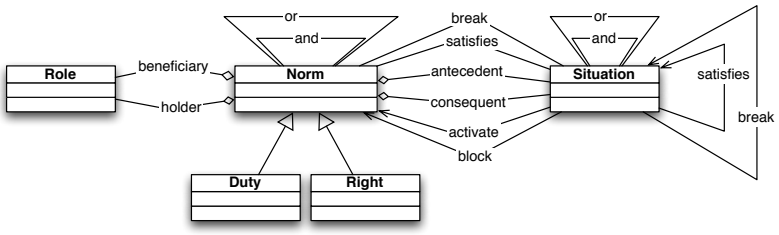


Figure 1. A meta-model for the *Nòmos* 2 modeling language.

certainly want their requirements to be fulfilled by the system-to-be. Figure 1 presents the metamodel of our proposal in the form of a UML class diagram. The rest of this section defines the concepts and relations, providing a description of all the concepts presented in the meta-model.

Norm. A *norm* is an atomic fragment of law with deontic status. Each norm consists of a 5-tuple (t, h, b, a, c), where:

- t is the *type* of the norm;
- h is the *holder* of the norm, the role having to satisfy the norm, if that norm applies;
- b is the *beneficiary*, the role whose interests are helped if the norm is satisfied;
- a is the *antecedent*, the conditions to satisfy for the norm to apply;
- c is the *consequent*, the conditions to satisfy for the norm to be satisfied.

A norm only takes effect under some condition. The condition, under which the norm takes effect, is its *antecedent*. If the norm takes effect, it is said to be **applicable**. E.g., "Taxi drivers must exhibit their taxi driving license while driving the taxi" – In this norm, "driving the taxi" is the antecedent and, when it holds, the prescription of exhibiting the driving license also holds (i.e., it must be satisfied). If the norm has a antecedent, but the antecedent does not hold, then the norm is not applicable. In the taxi example, when not driving, there is no need to exhibit the license. If the norm has no antecedent, it is always applicable (wrt the holder role). A norm is **satisfied** under some conditions. The condition under which the norm is satisfied is its *consequent*. In the taxi driving example, "exhibiting the taxi driving license" is the norm's consequent. Satisfying a norm is different from complying with it. If the taxi driver exhibits the license while not actually driving the taxi, there is no compliance The norm's *holder* specifies the one(s) in charge of complying with the norms, whilst the *beneficiary* specifies the one(s) in the interests of whom the norm applies. A norm's *type* allows us to specialize the norm concept according to a legal ontology of choice.

We assume that at least two specializations of a norm are provided by the ontology: duty and right. A duty entails that if the antecedent holds, then the holder *must* bring about the consequent. A right states that if the antecedent holds, then the holder *is entitled to, but can choose not to* bring about the consequent of the right. A duty thus requires that the consequent necessarily holds if the antecedent does, while the right allows the consequent to hold when the antecedent does. Rights are used to exclude the need to comply with a duty.

The ultimate purpose of our proposed language is to evaluate *compliance*. In natural language, we say that duties are *complied with*, whereas rights can be *exercised*. In this paper, *we use the phrase comply with a norm to mean both "comply with a duty", if the norm is a duty, and "exercise a right", if the norm is a right*. A duty can be violated by its holder; a right can only be violated by those who are in charge of guaranteeing the right, but not by its holder.

Situation. A situation is a state-of-affairs where some propositions are true, others false, and some neither true nor false. For example, when the situation "Christmas season" occurs, propositions such as "Falls within December" are true, while "Falls within summer" is false, and "Paolo likes Marta" is neither true nor false. Different norms are applicable in different situations, depending on whether the situation satisfies the norm's antecedent. A norm is satisfied in a given situation, if the situation satisfies the consequent of the norm. The concept of situation allows us to relate models of law to models of requirements independently from the specific language used for modeling requirements: consistent sets of requirements satisfy one or more situations, which make certain norms applicable. It is then unimportant if one sees requirements as use cases, goals, tasks, or otherwise, as long as one logically relates these requirements to situations.

Role. Roles define the subjects of a norm. Instances of the role concept appear as the holder and beneficiary as parts of instances of the norm concept. Both holder and beneficiary can be a single role, a set of roles who together must comply with the norm, or a set of roles of which at least one must comply with the norm. In the present paper we focus on compliance problems addressing a single role, simplifying the reasoning to only situations and norm type.

Relations. We define six types of relations between norms and situations:

- **Activate** from situations to norms: if situation is satisfied, then norm is applicable;
- **Block** from situations to norms: if situation is satisfied, norm does not apply;
- **Satisfy**:
  - From situations to situations: if the former is satisfied, the latter is too;
  - From situations to norms: if the former is satisfied, the latter is too;
  - From norms to situations: if the former is satisfied, the latter is too;
- **Break**:
  - From situations to situations: if the former is satisfied, the latter is not;
  - From situations to norms: if the former is satisfied, the latter is not;
  - From norms to situations: if the former is satisfied, the latter is not;
- **Conjunction** (And), between two or more norms to be satisfied together, or two or more situation instances to be satisfied together;
- **Disjunction** (Or), between two or more norms of which at least one ought to be satisfied, or two or more situation instances of which at least one ought to be satisfied.

The applicability and satisfaction of a norm, as well as the satisfaction of a situation, are decided based on the relations that they participate in. The rules for deciding applicability and satisfaction are defined in the next section. Applicability and satisfaction values let us check if we need to comply with a particular norm, if we do comply with it, and in which situations we comply or fail to do so.

$$S ::= p \mid S \wedge q \mid S \vee r \tag{1}$$
$$R ::= R \mid \wedge_{i=1..n,n>1}\ R_i \mid \vee_{i=1..n,n>1}\ R_i \tag{2}$$
$$N ::= (T, R_h, R_b, S_a, S_c) \tag{3}$$
$$\textit{And-S} ::= S \mid \wedge_{i=1..n,n>1}\ S_i \tag{4}$$
$$\textit{Or-S} ::= S \mid \vee_{i=1..n,n>1}\ S_i \tag{5}$$
$$\textit{And-N} ::= N \mid \wedge_{i=1..n,n>1}\ N_i \tag{6}$$
$$\textit{Or-N} ::= N \mid \vee_{i=1..n,n>1}\ N_i \tag{7}$$
$$\textit{X-S} ::= \textit{And-S} \mid \textit{Or-S} \tag{8}$$
$$\textit{X-N} ::= \textit{And-N} \mid \textit{Or-N} \tag{9}$$
$$\phi ::= S \mid N \mid \textit{X-S} \mid \textit{X-N} \tag{10}$$
$$\begin{aligned}\textit{Rel} ::= {} & \textit{X-S} \xrightarrow{\text{activate}} N \mid \textit{X-S} \xrightarrow{\text{block}} N \mid \textit{X-S} \xrightarrow{\text{satisfy}} S \mid \textit{X-S} \xrightarrow{\text{satisfy}} N \\ & \mid \textit{X-N} \xrightarrow{\text{satisfy}} S \mid \textit{X-S} \xrightarrow{\text{break}} S \mid \textit{X-S} \xrightarrow{\text{break}} N \mid \textit{X-N} \xrightarrow{\text{break}} S\end{aligned} \tag{11}$$

Figure 2. BNF rules defining the grammar of the language.

3 Syntax and Semantics

We describe below the modeling language through its syntax – symbols and grammar – and semantics – semantic domain and semantic mapping. The modeling language has a symbolic and a corresponding graphical syntax.

Symbolic syntax. The concepts from the ontology give sorts for the language. For simplicity (to avoid decorating symbols with sort symbols), we use N for instances of norm, T for instances of a concept specializing norm (e.g., for duty instances, we can write T_D), S for instances of situation, and R for instances of role. Expressions in the language, the well-formulas, are denoted by Greek lowercase letters. Symbols for relations are $\wedge$ for Conjunction, $\vee$ for Disjunction, $\xrightarrow{\text{activate}}$ for Activate, $\xrightarrow{\text{block}}$ for Block, $\xrightarrow{\text{satisfy}}$ for Satisfy, and $\xrightarrow{\text{break}}$ for Break. Expressions of the language are generated via the BNF rules in Figure 2.

Semantics. The semantic domain includes two triples of values:

- **Applicability** values: AT reads *Applicable*, AF reads *Not applicable*, and AU reads *Applicability undefined*; applicability values *are assigned to norms;*
- **Satisfiability** values: ST reads *Satisfied*, SF reads *Not satisfied*, and SU reads *Satisfiability undefined*; satisfiability values *are assigned to situations and to norms*.

Conjunction and Disjunction obtain Applicability and Satisfiability value as shown in Table 1. The Applicability and Satisfiability values are interesting in the following sense. Given a set of expressions in the language of the present framework, we want to make assumptions about the value of some of these formulas, and compute from there the value of others. For example, we want some situations to be ST and others SF, and we want to know what Applicability and Satisfiability values this propagates across other formulas.

Activate, Block, Satisfy, and Break relations do not have a value. We see them as propagating a value to their target. The value they propagate depends on whether the origin of the relation is a norm or situation, and on the relation type. Table 2 defines how Applicability and Satisfiability values propagate values over the Activate, Block, Satisfy, and Break relations. If there are several Activate, Block, Satisfy, Break relations targeting the same situation or norm, the values that they propagate to that target are treated as being in disjunction, and the following rules apply; let ψ be the target:

- If ψ receives both ST and SF, then value of ψ is ST;
- If ψ receives both ST and SU, then value of ψ is ST;
- If ψ is receives both SF and SU, then value of ψ is SU;
- If ψ receives both AT and AF, then value of ψ is AT;
- If ψ receives both AT and AU, then value of ψ is AT;
- If ψ is receives both AF and AU, then value of ψ is AU.

Compliance value. Applicability and Satisfiability values are relevant to the extent that they tell us whether requirements, through situations, satisfy applicable norms, that is, are compliant. We call this *Compliance value*, and compute it as shown in Table 3. The informal meaning for Compliance values is the following:

- **Compliance**: The norm instance applies and is satisfied;
- **Non-compliance**: The norm instance applies, but is not satisfied;
- **Tolerance**: The norm instance does not apply, making it unnecessary to satisfy it;
- **Inconclusiveness**: It is not known if the norm instance applies. This can intuitively be interpreted as there being a need to go back to the model to determine if the norm applies, and if yes, how to satisfy it in order to comply.

Shortcuts. What we call a *shortcut* is a template defining a frequently used combination of primitive relations. Shortcuts allow us to define relations which are useful for the modeling of norms, but are themselves not primitive. Rather, such relations are defined from primitive relations. For example, the Derogate relation from a norm N_1 to another norm N_2 is intended to convey that complying with N_1 (i.e., having N_1 applicable and

Table 1. Satisfiability and Applicability tables for Conjunction and Disjunction.

Satisfiability values				Applicability values			
ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$	ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$
ST	ST	ST	ST	AT	AT	AT	AT
ST	SF	SF	ST	AT	AF	AF	AT
ST	SU	SU	ST	AT	AU	AU	AT
SF	ST	SF	ST	AF	AT	AF	AT
SF	SF	SF	SF	AF	AF	AF	AF
SF	SU	SF	SU	AF	AU	AF	AU
SU	ST	SU	ST	AU	AT	AU	AT
SU	SF	SF	SU	AU	AF	AF	AU
SU	SU	SU	SU	AU	AU	AU	AU

Table 2. Propagation of satisfiability values across Activate, Block, Satisfy, and Break relations. The table reads as follows: If a situation ϕ has value ST and there is an Activate relation from ϕ to ψ, then that relation assigns AT to ψ.

$\phi \xrightarrow{\text{activate}} \psi$		$\phi \xrightarrow{\text{satisfy}} \psi$		$\phi \xrightarrow{\text{block}} \psi$		$\phi \xrightarrow{\text{break}} \psi$	
ϕ	ψ	ϕ	ψ	ϕ	ψ	ϕ	ψ
ST	AT	ST	ST	ST	AF	ST	SF
SF	AU	SF	SU	SF	AU	SF	SU
SU	AU	SU	SU	SU	AU	SU	SU

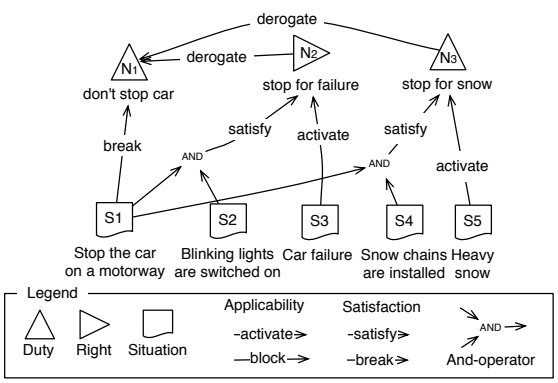
satisfied) makes it unnecessary to satisfy N_2, that is, makes N_2 non-applicable. Derogate is what we call a shortcut, because $N_1 \xrightarrow{\text{derogate}} N_2$ can be rewritten using primitive relations, as follows: (i) there is a situation S_a such that $S_a \xrightarrow{\text{activate}} N_1$, and (ii) there is a situation S_s such that $S_s \xrightarrow{\text{satisfy}} N_1$, and (iii) $S_a \wedge S_s \xrightarrow{\text{block}} N_2$. We use the following shortcuts (but others could be defined):

- **Imply**: $N_1 \xrightarrow{\text{imply}} N_2$ means there is S_a, S_b such that $S_a \xrightarrow{\text{activate}} N_1$, $S_b \xrightarrow{\text{satisfy}} N_1$ and $S_a \xrightarrow{\text{activate}} N_2$, $S_b \xrightarrow{\text{satisfy}} N_2$.
- **Derogate**: $N_1 \xrightarrow{\text{derogate}} N_2$ means there is S_a, S_b such that $S_a \xrightarrow{\text{activate}} N_1$, $S_b \xrightarrow{\text{satisfy}} N_1$ and $S_a \wedge S_b \xrightarrow{\text{block}} N_2$.
- **Endorse**: $N_1 \xrightarrow{\text{endorse}} N_2$ means there is S_a, S_b such that $S_a \xrightarrow{\text{activate}} N_1$, $S_b \xrightarrow{\text{satisfy}} N_1$ and $S_a \wedge S_b \xrightarrow{\text{activate}} N_2$.

Graphical syntax. Mapping from linear syntax to graphical syntax is shown in Figure 3. The graphical syntax is illustrated with an excerpt of the (Italian) traffic rules. The figure can be read as follows: It is forbidden (norm N_1 of type duty, represented through the symbol $\triangle$) to stop the car on a motorway (situation S_1, represented through the symbol $\square$). In case of a car failure (S_3) it is permitted (norm N_2 of type right, represented through the symbol $\triangleright$) to stop the car on the motorway (S_1), provided that blinking lights are switched on (S_2); in this case, the first prohibition is not applicable. In case of heavy snow (S_5), it becomes mandatory (N_3) to stop the car (S_1) and install snow chain on the car (S_4). Notice that shortcut relations (e.g., $N_2 \xrightarrow{\text{derogate}} N_1$) connect two norms but have the semantics described above. Norms, situations and the relations among them define the three sets N, S and Rel that form the norm model L. In the example, all the norms in L are held by a single role (say, "Driver"); since in the present paper we only reason on one-role models, the holder (and beneficiary) are not depicted here.

Table 3. Compliance values of norms.

Norm is a duty			Norm is a right		
Applicability	Satisfiability	Compliance value	Applicability	Satisfiability	Compliance value
AT	ST	Compliance	AT	ST	Compliant
AT	SF	Non-compliance	AT	SF	Tolerance
AT	SU	Not compliant	AT	SU	Tolerance
AF	ST	Tolerance	AF	ST	Tolerance
AF	SF	Tolerance	AF	SF	Tolerance
AF	SU	Tolerance	AF	SU	Tolerance
AU	ST	Inconclusiveness	AU	ST	Inconclusiveness
AU	SF	Inconclusiveness	AU	SF	Inconclusiveness
AU	SU	Inconclusiveness	AU	SU	Inconclusiveness



S := $\{S_1$ = Stop the car on a motorway,
S_2 = Blinking lights switched on,
S_3 = A car failure happened,
S_4 = Snow chains are installed,
S_5 = Heavy snow $\}$
N := $\{N_1$, = don't stop car,
N_2 = stop for failure,
N_3 = stop for snow $\}$
Rel := $\{S_1 \xrightarrow{\text{break}} N_1$,
$(S_1, S_4 \xrightarrow{\text{satisfy}} N_3)$,
$(S_1, S_2 \xrightarrow{\text{satisfy}} N_2)$,
$S_3 \xrightarrow{\text{activate}} N_2, S_5 \xrightarrow{\text{activate}} N_3$,
$N_2 \xrightarrow{\text{derogate}} N_1, N_3 \xrightarrow{\text{derogate}} N_1\}$

Figure 3. Graphical syntax for the modeling language.

Compliance to a law. On a norm model — intended as a set of norms, situations and the relations between them — is defined Law compliance as a global property. A law as a whole is complied with if (i) each norm is complied with; or (ii) one or more norms are not complied with, but they are derogated by others, and the derogating ones are complied with; or (iii) one or more norms are not applicable; or, finally, (iv) one or more norms are rights, applicable but not exercised by their holder. A set of situations make a norm model compliant, if $\forall N \in L, tol(N) \vee com(N)$, where $tol(N)$ means that a *tolerance* condition (as of Table 1) is inferred on the norm N, $com(N)$ means that a *compliance* condition is inferred for that norm. In a model with multiple roles — in which one's duty is to ensure another's right — compliance with a right can also be interpreted in the following more constraining way: a right is complied with if and only if that right (i) is applicable, (ii) is satisfied, and (iii) there are no situations which are satisfied, and which block or break that right. We can verify if a right is satisfied in this sense if its compliance value is Compliant, and there are no block or break relations to that right, originating in a satisfied situation.

4 Reasoning with norm models

Norm models can be analyzed by means of forward or backward reasoning algorithms. **Forward reasoning** involves giving applicability/satisfiability labels to some nodes of a *Nòmos* 2 network and then propagating these labels to all other nodes of the network, as described in [2]. Specifically, to each situation (and to each norm) we associate a variable $Sat(S)$ (and $Sat(N)$) with values ST, SF, SU, representing the current evidence of satisfiability of situation S (and norm N). To each norm is associated a variable $App(N)$ with values AT, AF, AU, representing the current evidence of applicability of norm N. Default values are SU and AU. Given an initial values assignment to situations (input situations from now on) forward reasoning focuses on the forward propagation of these initial values to all other situations and to the norms of the graph according to the rules described in Table 1. So for example, if $Sat(S_1)$ = ST and $S_1 \xrightarrow{\text{activate}} N_1$, then the algorithm propagates $App(N_1)$ = AT. Initial values represent the evidence available

```
Algorithm:ComplianceAnalysis
Input: L, initialAssignment
Output: complianceValues
assignment = ForwardReason(L,initialAssignment );
foreach N_i ∈ L do
    if assignment.appValue(N_i) = AT then
        if assignment.satValue(N_i) = ST then
            solution.set( N_i, compliance );
        else
            if N_i instanceOf Duty then
                solution.set( N_i, non_compliance );
            else
                solution.set( N_i, tolerance );
    if assignment.appValue(N_i) = AU then
        solution.set(N_i, inconclusiveness);
    else
        solution.set( N_i, tolerance );
end
return solution;
```

Algorithm 1: Compliance analysis. This algorithm evaluates the compliance status of a set of norms belonging to a law (L) with respect to an initial set of situations.

```
Algorithm: ComplianceSearch
Input: L, initialAssignment
Output: selectedAssignment
assignment = ForwardReason(L,initialAssignment);
while ComplianceAnalysis(L,assignment) contains
  non_compliance or inconclusiveness do
    attempt = BackwardReason(L,assignment);
    if attempt = ∅ then
        return ∅;
    else
        assignment = attempt
    end
end
return assignment;
```

Algorithm 2: Compliance search. This algorithm evaluates the situations to satisfy in order to comply with the norms applicable from an initial set of situations.

about the satisfaction of the situations, namely evidence about the state of the situation. After the forward propagation of the initial values, the user can look the final values of the norms of interest. In other words, the user observes the effects of the initial situation over the norms.

Backward reasoning involves giving applicability/satisfiability labels to some target nodes of a *Nòmos* 2 network and then searching back through the network for possible input values leading to some desired final value, as described [3]. We set the desired final values of a target norm, and we want to find possible initial assignments to the input situations, which would cause the desired final values of the target norms by forward propagation. For example, if $S_1 \xrightarrow{\text{activate}} N_1$, $S_2 \xrightarrow{\text{satisfy}} N_1$, $S_3 \xrightarrow{\text{satisfy}} N_1$, and in the initial assignment we specify that we want N_1 to be complied with (i.e., we want it to be $Sat(N_1)$ = ST and $App(N_1)$ = AT), then the algorithm may find the assignment $Sat(S_1)$ = ST, $Sat(S_2)$ = SF, $Sat(S_3)$ = ST. Backward and forward reasoning algorithms also take care of possible cycles in graphs.

Forward and backward reasoning algorithms can be applied to norm models to answer questions such as those raised in section 1.

Applicability analysis is intended to find the (sub)set of norms applicable to a given (sub)set of situations. The analysis consists in a straightforward application of the forward reasoning algorithm. The output of the propagation is that subset L^a of the norms L such that $\forall N \in L^a, App(N)$ = AT.

Compliance analysis aims at providing evidence of compliance (or violation) of a norm model. The algorithm for inferring compliance of a given norm model L (comprised by a set of norms, a set of situations, and a set of relations) and an assignment to their applicability and satisfiability values, is illustrated in Algorithm 1. Given an initial assignment to situations, compliance analysis propagates forward applicability and satisfiability values

(line 1). When the propagation terminates, each norm of the model is evaluated (line 2). If applicability can not be inferred (AU), inconclusiveness is concluded (line 12); if applicability is inferred to be false, tolerance is concluded (line 14); if applicability is true, then satisfiability has to be checked (line 3). If it is also true (line 4), then we are surely in a compliance case; otherwise, we have to evaluate the type of the norm: if duty, we have a violation (a duty has to be satisfied but it's not the case); if right, tolerance is concluded, as right are discretional. When the algorithm terminates $solution$ contains the compliance values of the norms.

Compliance search aims at finding a compliance solution for a given set of situations, possibly identifying other situations that complement the given set. In other words, supposing that some situation S_k is desired to be legally satisfied (i.e., satisfied and compliant), this analysis answers the question: what else must we do if we want the satisfaction S_k to be compliant?

Searching for a compliant way to satisfy a desired situation is done through an iterative algorithm, as listed in Algorithm 2. Initially, input satisfaction values of situations are propagated to norms applicability and satisfiability values (line 3). The resulting assignment is evaluated for compliance (line 2). If any non-compliance is found, an iteration is performed, until either an assignment is found, which does not include violations, or no further assignment can be made. For each iteration, the backward reasoning algorithm searches for an assignment ($attempt$, line 3) that makes the current one satisfied. If such assignment does not exist, the algorithm exits. Otherwise, the current one is checked for compliance and the algorithm exits with success or iterates again, according to whether compliance is found, or not.

5 Illustrative case study

The U.S. Health Insurance Portability and Accountability Act (HIPAA) is a well-known law to researchers and practitioners. Although years already passed since it has been laid down, the effort for engineering HIPAA-compliant requirements is still expensive because of its complexity. For example, section §164.502[3] of HIPAA lays down general rules of uses and disclosures of protected health information for covered entities. The section contains 174 paragraphs, all of them directly or indirectly concerning one single action, abstractly denoted as "use or disclosure of protected health information", which under some circumstances is forbidden, under other circumstance is permitted or even required. The paragraphs regulate such circumstances.

Figure 4 depicts a model extracted from HIPAA section §164.502. The model contains 10 situations (S_1, ..., S_{10}; situation S_{11}, S_{12}, S_{13} and S_{14} have been extracted from cross-referenced fragments and depicted for explanation purposes, but not counted here) and 12 norms, linked by 35 relations. Considering that a situation can be known to hold (ST), known to not hold (SF) or uncertain, due to insufficient knowledge, ambiguity or else (SU), we have a total of $3^{10} = 59.049$ possible assignments. On the other hand each norm can be not applicable, applicable but not complied with, or applicable and complied with. Overall, the situation generate therefore a total of $3^{12} = 531.441$ possibilities for norms. Out of these, we want to search for (i) the (sub)set of norms

[3] http://www.hipaasurvivalguide.com/hipaa-regulations/164-502.php

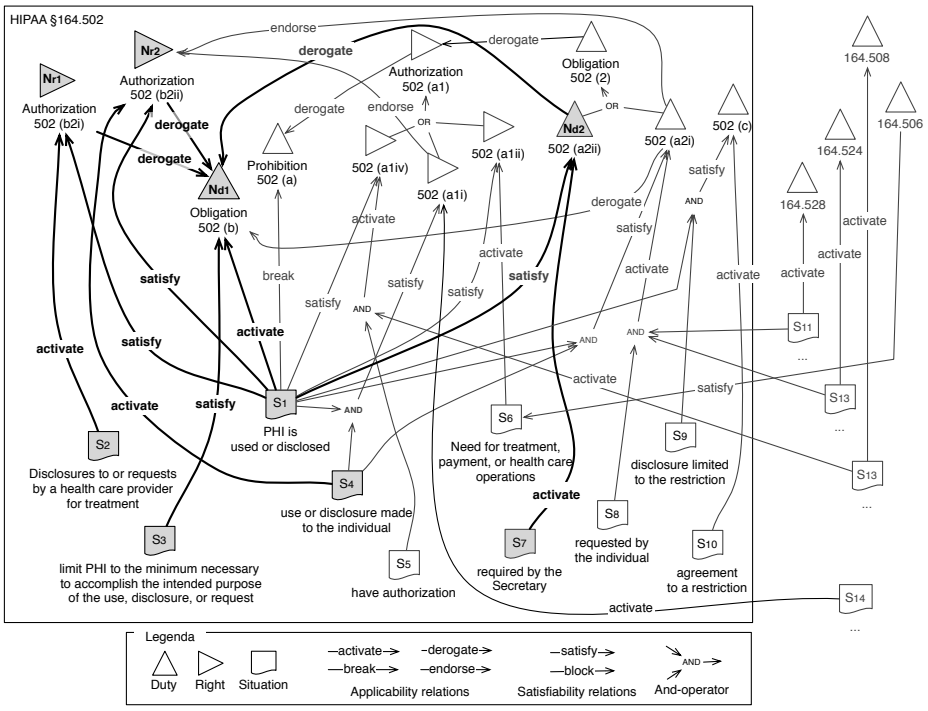


Figure 4. HIPAA paragraphs 502 (1) to (2vi) law model. The situations, relations, and norms highlighted in the model are the one considered in the illustrative case study.

applicable to a given (sub)set of situations, and (ii) the set of situations that need to be satisfied in order to comply with the norms activated by an initial set of situation. To the first question we answer through *Applicability analysis*; to the latter question we answer through *Compliance analysis* and *Compliance search*.

We now show how these three analyses work in an example taken from the model depicted in Figure 4. In the following example we consider the sub-model composed of five situations S_1, S_2, S_3, S_4, S_7 (highlighted in grey in Figure 4) and the effects of these five situations on four norms N_{r1}, N_{r2}, N_{d1}, N_{d2}.[4] The relations considered in the example are highlighted in bold.

Applicability analysis. We are interested in *finding the applicability values for the norms*. We start the analysis with the initial assignment of satisfaction values for the situations, $Sat(S_1, S_2, S_3, S_4, S_7)$ = (SF, SF, ST, ST, ST). The four relations in S_1 propagate values to all four norms (AU or SU respectively), the relation in S_2 assigns AU to N_{r1}, S_3 assigns ST to N_{d1}, S_4 propagates AT to N_{r2}, while S_7 sets N_{d2} to AT. The result of this forward analysis is summarized in Table 4 (second column). Analyzing the

[4] The subscript r indicates a norm of type right (N_r), while d is for duties (N_d).

Table 4. Values returned by the three analyses with the given input assignment (left part of the table) and after one iteration of the compliance search (right part of the table).

InitialAssignment: $Sat(S_1,S_2,S_3,S_4,S_7)$=(SF, SF, ST, ST, ST)				$Sat(S_1,S_2,S_3,S_4,S_7)$=(ST, SF, ST, ST, ST)		
Norm	Forward reasoning	Compliance analysis	Compliance search	Forward reasoning	Compliance analysis	Compliance search
N_{r1}	AU, SU	inconclusiv.		AU,ST	inconclusiv.	
N_{r2}	AT, SU	tolerance	$Sat(S_1, S_2, S_3, S_4, S_7)$	AT,ST	compliance	$Sat(S_1, S_2, S_3, S_4, S_7)$
N_{d1}	AU, ST	inconclusiv.	= (ST, SF, ST, ST, ST)	AF,ST*	tolerance	= (ST, ST, ST, ST, ST)
N_{d2}	AT, SU	non compliance		AT,ST	compliance	

* The relation $N_{r1} \xrightarrow{\text{derogate}} N_{d1}$ now holds as both S_1 and S_4 are now ST.

shortcut-relation $N_{r1} \xrightarrow{\text{derogate}} N_{d1}$, we have that it was as if in the model there was a relation $S_1 \wedge S_2 \xrightarrow{\text{block}} N_{d1}$. Since the two situations are not satisfied, the block relation does not hold, and therefore the norm is not derogated. Similarly in the other derogation relation, the applicability of N_{d1} is not altered. The applicability analysis therefore returns that N_{r2} and N_{d2} are applicable, while the other two norms have undefined applicability.

Compliance analysis. Given the labeling propagation identified in the previous step (line 1 of Algorithm 2), this analysis *evaluates all norms and returns their compliance status* (see Table 4, third column). Norm N_{r1} is evaluated as inconclusive because — as we can see also from Table 3 — a right with undefined applicability and satisfiability has status of inconclusiveness (line 12 of Algorithm 1). Norm N_{r2} is tolerated because an applicable right with undefined satisfiability (evaluated by line 10 of the algorithm), is assigned value "tolerance". Norm N_{d1} is evaluated as inconclusive because when a duty has undefined applicability but is satisfied, then a precise compliance value cannot be drawn. Finally, norm N_{d2} is identified in a status of non-compliance because it is an instance of a duty that is applicable and is not satisfied (line 8).

Compliance search. We are now interested in *finding the situations* (and their satisfiability value) *needed to comply with the applicable norms* of our initial situation. Following Algorithm 2 (line 1) we perform forward reasoning analysis and Compliance Analysis, obtaining the results described in the left side of Table 4. As the Compliance Analysis of our example has both non-compliance and inconclusiveness instances (line 2), the algorithm performs backward reasoning (line 3). Tracing backward the relation $S_1 \xrightarrow{\text{satisfy}} N_{d2}$, the algorithm identifies that both issues can be solved by satisfying S_1, which is now set to ST. A new iteration is started (with $Sat(S_1)$=ST) and Compliance Analysis is performed again, revealing that now N_{r1} is evaluated as inconclusive (its label are now AU,ST) (see right side of Table 4). A new iteration of the algorithm finds that the final assignment $Sat(S_1, S_2, S_3, S_4, S_7)$ = (ST, ST, ST, ST, ST) leads to a compliant status where no case of inconclusiveness or non-compliance are present.

6 Related Work

In Tropos [4] requirements of the system-to-be are analyzed in terms of goal models, represented by means of goal graphs composed of goal nodes and goal relations. It is

possible to analyze goal graphs with both forward reasoning and backward reasoning. In Tropos approach however, the notion of applicability is absent and so the inference rules to reason about compliance. In KAOS [5] requirements are modeled and analyzed through goals, requirements, scenarios, and responsibility assignments. Goal models are formalized using temporal logic to achieve correctness. Such formalism has several advantages, but they lack the concepts of norm and situation. Goal oriented techniques have been used to represent legal prescriptions. For example, Darimont and Lemoine have used KAOS to represent objectives extracted from regulation texts [6]. Ghanavati et al. [7] use URN (User Requirements Notation) to model goals and actions prescribed by laws. Likewise, Rifaut and Dubois use *i** to produce a goal model of the Basel II regulation [8]. Goal-oriented approaches to norm modeling are useful when the complexity of norms is little enough to be reduced to goal relations. When the notion of applicability is needed, goal-oriented techniques fail in capturing its effects on reasoning about alternatives.

AI techniques have been used to build formal models of regulations. Among recent works, [9] presents a formal framework for reasoning about derogation in law. In [10] conditional elements of norms are formalized into an operational framework. [11] provides a strong characterization for a formal elaboration of the notion of applicability. We depart from these approaches, in that our approach is less expressive but better suited for the purpose of supporting requirements engineering. Legal reasoning approaches are hardly usable by non-experts, so that they are rendered useless during the requirements engineering phase, which generally involves a continuous interaction with stakeholders to check that the work of the analysts has captured their needs and preferences.

Breaux & Antón focus on information extraction from natural language texts of privacy policies [12]. A set of heuristics is created to systematically convert unstructured legal texts into structured artifacts. Artifacts are then combined into a frame-based method for manually acquiring legal requirements from regulations. Such approach has been used as a basis for tool-supporting the identification of requirements in legal documents [13]. The Privacy APIs framework converts regulatory privacy rules in natural language into formal expressions, which are automatically analyzed to identify problematic statements [14]. The SALEM system adapts a linguistic approach for the extraction of semantic concepts from Italian legal documents, such as actors, actions and properties. It also classifies different types of provisions using a text categorization algorithm [15]. We depart from such works in that our analysis starts after the information extraction from legal text has been concluded.

7 Conclusions

In this paper we propose *Nòmos* 2, a framework for modeling preconditions and post-conditions of legal norms and their relationships into early representations of the requirements problem and solution space. Moreover, we describe how to traverse norm models to evaluate the applicability and satisfiability of norms for given sets of requirements, and thereby the compliance of these requirements. Automatic reasoning is accomplished by using extensions of existing forward and backward reasoning algorithms. The scope of the present work is limited to analyze situations that concern a single role. Further work is needed to reason on multiple roles and delegation of responsibilities.

Acknowledgements This research has been partially funded by the European Research Council (ERC) through advanced grant 267856, titled "Lucretius: Foundations for Software Evolution" (04/2011-03/2016). http://www.lucretius.eu.

References

1. A. Siena, J. Mylopoulos, A. Perini, and A. Susi, "Designing law-compliant software requirements," in *Conceptual Modeling - ER 2009*, pp. 472–486, 2009.
2. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal reasoning techniques for goal models," *J. Data Semantics*, vol. 1, pp. 1–20, 2003.
3. R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models," in *CAiSE*, pp. 20–35, 2004.
4. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.
5. A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
6. R. Darimont and M. Lemoine, "Goal-oriented analysis of regulations," in *ReMo2V, held at CAiSE'06* (R. Laleau and M. Lemoine, eds.), vol. 241 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2006.
7. S. Ghanavati, D. Amyot, and L. Peyton, "Towards a framework for tracking legal compliance in healthcare.," in *CAiSE* (J. Krogstie, A. L. Opdahl, and G. Sindre, eds.), vol. 4495 of *Lecture Notes in Computer Science*, pp. 218–232, Springer, 2007.
8. A. Rifaut and E. Dubois, "Using goal-oriented requirements engineering for improving the quality of iso/iec 15504 based compliance assessment frameworks," in *Proceedings of RE 2008*, (Washington, DC, USA), pp. 33–42, IEEE Computer Society, 2008.
9. N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky, "Reasoning about conditions and exceptions to laws in regulatory conformance checking," in *Proc. of the 9th Int. Conf. on Deontic Logic in Computer Science*, DEON '08, (Berlin, Heidelberg), pp. 110–124, Springer-Verlag, 2008.
10. G. Sartor, "The structure of norm conditions and nonmonotonic reasoning in law," in *Proceedings of the 3rd international conference on Artificial intelligence and law*, ICAIL '91, (New York, NY, USA), pp. 155–164, ACM, 1991.
11. G. Boella, G. Governatori, A. Rotolo, and L. Van Der Torre, "Lex minus dixit quam voluit, lex magis dixit quam voluit: a formal study on legal compliance and interpretation," in *AICOL-I/IVR-XXIV'09*, (Berlin, Heidelberg), pp. 162–183, Springer-Verlag, 2010.
12. N. Kiyavitskaya, N. Zeni, T. D. Breaux, A. I. Antón, J. R. Cordy, L. Mich, and J. Mylopoulos, "Automating the extraction of rights and obligations for regulatory compliance.," in *ER2008* (Q. Li, S. Spaccapietra, E. Yu, and A. Olivé, eds.), vol. 5231 of *Lecture Notes in Computer Science*, pp. 154–168, Springer, 2008.
13. T. Breaux and A. Antón, "Analyzing regulatory rules for privacy and security requirements," *IEEE Trans. Softw. Eng.*, vol. 34, pp. 5–20, January 2008.
14. M. J. May, C. A. Gunter, and I. Lee, "Privacy apis: Access control techniques to analyze and verify legal privacy policies," in *Proceedings of the 19th IEEE workshop on Computer Security Foundations*, (Washington, DC, USA), pp. 85–97, IEEE Computer Society, 2006.
15. C. Biagioli, E. Francesconi, A. Passerini, S. Montemagni, and C. Soria, "Automatic semantics extraction in law documents," in *Proceedings of the 10th international conference on Artificial intelligence and law*, ICAIL '05, (New York, NY, USA), pp. 133–140, ACM, 2005.