FLAVOR: a Formal Language for A posteriori Verification of Legal Rules

Romuald Thion
Université Lyon 1, LIRIS
F-69622, France
Email: Romuald.Thion@liris.cnrs.fr

Daniel Le Métayer
INRIA Grenoble – Rhône-Alpes
F-38334, France
Email: Daniel.Le-Metayer@inrialpes.fr

***Abstract*—Organizations have to comply with a growing number of rules (legal, regulatory, contractual, etc.) and it becomes more and more challenging for them to ensure that they really meet all their obligations. IT systems, even if they cannot provide the full answer to this complex issue, can help organizations in the management and monitoring of their obligations. In this paper, we derive a set of requirements from representative examples of obligations and propose a language providing essential features such as "contrary to duty" obligations, obligations with deadlines and contextual obligations. We define its semantics, suggest its implementation as an audit mechanism, and show its application to the definition of privacy policy rules.**

***Keywords*-obligations; compliance; audit.**

I. Context and motivations

Organizations have to comply with a growing number of legal rules stemming from law, regulations, corporate policies or contractual agreements. These rules have a potential impact on all their activities and breaches may lead to different types of damages, including financial losses, lawsuits, competitive disadvantages and disrepute. Organizations also depend more and more on information technologies to carry out all their production and management tasks. It is thus of crucial importance for an organization that its IT system can act as a facilitator rather than an obstacle in its endeavour to ensure legal compliance. Indeed, manual compliance verifications are error prone and tend to exceed the capacity of most organizations. IT systems, even if they cannot provide the full answer to this complex issue, can help organizations in the management and monitoring of their obligations.

Generally speaking, the actions to be monitored can be checked either *a priori* or *a posteriori*. *A priori* checks are stronger in the sense that they make it possible to ensure that no breach will occur. However, they are too constraining, if not unenforceable, in many situations. To start with, systems may involve external agents, such as human users, whose actions cannot be entirely controlled. Also, the modification of a legacy system to introduce *a priori* checks may be difficult and these checks may result in unacceptable decrease in performance of the system. Furthermore, even when they could be implemented, *a priori* checks are not desirable in situations in which it could be legitimate to bypass the rules. For instance, it is necessary to provide emergency procedures to access personal health records when human lives are at stake, even if the medical practitioner on duty does not have sufficient permissions [10]. *A posteriori* checks are also advocated for social reasons, to avoid the potentially oppressive atmosphere generated by systematic *a priori* controls [13]. Indeed, as argued by several lawyers, it is generally preferable to consider citizens as responsible actors and to rely on *a posteriori* controls to encourage them to behave well.

Technically speaking, *a posteriori* mechanisms make it possible to deal with obligations (constraints to be satisfied in the future) which are quite common in legal rules and are more difficult to implement than access control rules. They are also less intrusive than *a priori* mechanisms because they can be added as independent components working on the logs of the system to be monitored. Most systems already record their histories of execution in logs (e.g., web servers, operating systems, databases management systems), which makes the deployment of *a posteriori* mechanisms easier and less expensive.

The first task to design a "compliance system" is to define a precise and non ambiguous language for the expression of the legal rules. In this paper, we derive a set of requirements from representative examples of obligations (Section II) and define a language called FLAVOR[1] meeting these requirements (Section III). The essential features provided by FLAVOR are the possibility to express "contrary to duty" obligations (substitute obligations to be fulfilled in case of breach of the primary obligation[2]), obligations with deadlines and contextual obligations. We define a strength ordering between obligations in Section IV and illustrate the language with an excerpt of privacy policy in Section V. In Section VI, we discuss related work and show that the combination of the aforementioned features has not been fully addressed in the literature so far. Finally, Section VII concludes the paper and suggests avenues for further research.

[1]Formal Language for A posteriori Verification Of legal Rules.

[2]Contrary-to-duty obligations are common in legal rules, for example in contracts (penalty clauses) and criminal law [17].

Table I
EXCERPT OF BUSINESS AGREEMENT

1) Within two weeks after receipt of the Software, Customer shall pay to Supplier the amount of twenty thousand Euros.
2) The payment of any additional service by Customer shall be due within four weeks after receipt of a valid invoice for the service.
3) In case of late payment, Customer shall pay, in addition to the due amount, a penalty of 5% of this amount.

Table II
EXCERPT OF PRIVACY POLICY

1) The Controller must answer any access query from the Subject within a delay specified by the Subject in the policy associated with the personal data.
2) The Controller must delete the data within a delay specified by the Subject in the policy associated with the data.
3) The Controller is allowed to transfer the personal data to a third party only in the conditions set forth in the policy associated with the data. The Subject can define one of the following transfer policies:
   - Any transfer is forbidden.
   - The Controller must inform the Subject of any transfer.
   - The Controller must receive the authorization of the Subject prior to any transfer.

II. REQUIREMENTS

In order to identify the main features required for our compliance language, we start with the analysis of some examples of legal rules expressed in natural language. These rules, which are listed in Table I and Table II, are representative of contractual as well as regulatory provisions [2], [12], [15], [16].

For the sake of generality, we consider that the system to be monitored is observable through a set of *events*. These events are characterized by properties which are used to define the rules. Traces, which are abstract versions of execution logs, are finite sequences of events on which the rules are evaluated.

In the remainder of this section, we analyze the examples of Table I (excerpt of business agreement) and Table II (excerpt of privacy policy) to identify the main requirements for a compliance language.

A. Contrary to duty obligations

A contrary to duty obligation consists of a primary obligation and an alternative obligation which becomes effective when (and if) the primary obligation is breached. Contrary to duty obligations are useful to express penalty clauses in contracts as well as compensations and sanctions for breaches of legal rules. They differ from the mere choice between two alternative obligations because the fulfillment of the alternative obligation *before* the breach of the primary obligation does not amount to a fulfillment of the whole obligation. As an illustration, Clause 1 and Clause 2 in Table I are primary obligations while Clause 3 is a contrary to duty obligation that will become effective in case of breach of Clause 1 or Clause 2.

B. Temporal and deontic modalities

One of the most pervasive characteristics of legal rules is the interaction between temporal (always, eventually) and deontic (obligatory, prohibited) modalities. This interaction clearly appears in constructions such as "shall ... within ... days after ...", "must ... within ..." or "must ... prior to ..." in Table I and Table II. Indeed, most obligations or prohibitions come with a deadline which may be defined by a fixed date, by a delay or by an event. There is a dual relationship between deontic modalities and the corresponding deadline: an obligation to perform a given action is of little practical significance without a deadline whereas a prohibition without any deadline is very strong since it applies forever.

C. Conditions and contexts

In essence, legal rules are expressed as abstract and general statements intended to be applied in a variety of circumstances. To reach this level of abstraction, the wording of a legal rule generally distinguishes the effect of the rule (action to be performed or prevented) and its context of application. The context of application typically involves conditions on the occurrence of an external event that we call the "trigger" in the sequel. It may also define the term of the obligation (termination date). Note that two kinds of triggers appear in the above tables: some of them are implicitly one-shot (single occurrence) while others are recurrent (multiple occurrences). For example, in Table I, the first trigger is the receipt of the software, which is a one-shot event. In contrast, the second trigger, the receipt of invoices for additional services, may occur several times (or never). In Table II, the first trigger is implicitly the collection of data, which is a recurrent trigger: the controller does not have any obligation as long as he has not collected any personal data; he gets new obligations each time he collects new data. Clause 1 defines another recurrent trigger corresponding to the access queries from the subject.

III. THE FLAVOR LANGUAGE

The first challenge in the design of a language meeting the requirements of the previous section is to devise an integration of the deontic and temporal modalities required to express legal rules which avoids the paradoxes and counter-intuitive meaning often arising in modal logics [18]. The second challenge is to provide a formal semantics with intuitive understanding suitable for the implementation of an auditing tool.

We assume a set K of *keys* and a set C of *values*. An *event* $e \in E$ is a function from keys into values $E = K \to C$. Keys are used to access the values of the different fields of an event.

The language is interpreted over traces. A *trace* σ is a finite sequence of events $\sigma = e_0 e_1 e_2 \dots e_{n-1} \in E^\star$. We note $\sigma(i) = e_i$, $|\sigma| = n$ the length of a trace and $\cdot : E^\star \times E^\star \to E^\star$ the concatenation operator.

A. Syntax

We assume a set V of variables and define, for any set X, $X_\perp$ as $X \cup \{\perp\}$ where $\perp$ is the undefined value. The basic building block of the language is the set of event properties. An *event property* $\rho \in P$ is a pair $\langle b, \gamma \rangle$ made of a *list of binders* b and a *condition* γ:

- The list of binders b is a list $[(k_0 \mapsto x_0), (k_1 \mapsto x_1), \dots]$ with $k_i \in K$ and $x_i \in V + C$ where $+$ is the disjoint set union. Each key and each variable can appear *at most once* in a finite list of binders. Intuitively, $(k \mapsto v)$ with $v \in C$ defines a constraint on the value of the field k of the event.
- The condition $\gamma \in \Gamma$ is a conjunction of constraints built from a set R of pre-defined predicates. We assume arithmetic predicates here, but the language can include any relevant domain specific predicate[3]. The language of constraints Γ is defined by the following grammar, where $v \in V$, $c \in C$, $t_1, t_2 \in T$ and $\gamma_1, \gamma_2 \in \Gamma$:

$$T ::= v \mid c \mid (t_1 + t_2) \mid (t_1 \times t_2)$$
$$R ::= \leq \mid \geq \mid =$$
$$\Gamma ::= \gamma_1 \wedge \gamma_2 \mid R(t_1, t_2) \mid \mathbf{tt} \mid \mathbf{ff}$$

An *environment* f is a mapping from variables to values: $f \in M = V \to C_\perp$. An *extension* f' of an environment f is such that $dom(f) \subseteq dom(f')$ and $\forall v \in dom(f)$, $f'(v) = f(v)$. If $\rho = (b, \gamma)$ is a property and f an environment, $\rho[f]$ (respectively $b[f]$ and $\gamma[f]$) is the property ρ (respectively the binder b and the condition γ) in which each variable v is replaced by $f(v)$ if $f(v) \neq \perp$.

The pattern matching function $match$ takes a property ρ, an event e and an environment f and returns an extension f' if the property and the event match and $\perp$ otherwise:

$match : P \times E \to M \to M_\perp$
$match\ (\rho, e)\ f = f'$ with
let $\rho[f] = \langle b, \gamma \rangle$ in
  if $\exists (k \mapsto c) \in b$ s.t. $c \in C$ and $c \neq e(k)$
  then $f' = \perp$
  else let $f''(x) =$ if $\exists (k \mapsto x) \in b$ s.t. $x \in V$
      then $e(k)$ else $f(x)$
    in if $\gamma[f''] \Leftrightarrow \mathbf{ff}$ then $f' = \perp$ else $f' = f''$

As an illustration, let us consider a property $\rho = ([k_1 \mapsto a, k_2 \mapsto t], t \leq a + 10)$, an event e with $e(k_1) = 3$ and $e(k_2) = 8$ and an environment f defined by $f(a) = 3$. We have $\rho[f] = ([k_1 \mapsto 3, k_2 \mapsto t], t \leq 3 + 10)$ and $match\ (\rho, e)\ f$ successfully extends f to f' with $f'(t) = 8$ and $f'(x) = f(x)$ for $x \neq t$.

We can now proceed with the definition of the FLAVOR language itself.

Definition 1. *Formal syntax*

The syntax of the FLAVOR *language is defined by the following grammar, where ψ and φ denote sentences (elements of $\mathcal{L}$), and ρ and δ denote event properties (members of P):*

$$\oplus\langle \rho, \delta \rangle \mid \ominus\langle \rho, \delta \rangle \mid \langle \rho, \delta \rangle \rightsquigarrow \varphi \mid \langle \rho, \delta \rangle \dot{\rightsquigarrow} \varphi \mid \psi \trianglerighteq \varphi \mid \psi \wedge \varphi$$

A key design choice for the language is minimality: we have chosen to introduce the minimal set of constructors necessary to meet the requirements identified in Section II.

Contrary to duty obligations (Section II-A): these obligations are captured by the constructor $\psi \trianglerighteq \varphi$, where the left-hand-side ψ is the primary obligation and the right-hand-side φ is the subsidiary obligation. For example, the obligation to pay invoices for additional services in Table I, is defined as $\oplus\langle \rho_p, \delta_p \rangle \trianglerighteq \oplus\langle \rho_s, \mathbf{ff} \rangle$. Property ρ_p is the payment of the invoice and property ρ_s is the payment of the initial amount increased by five percent. As explained below, δ_p and $\mathbf{ff}$ represent the deadlines for the corresponding events.

Temporal and deontic modalities (Section II-B): The deontic block of the language is made of pairs of events (ρ, δ) decorated with a modality $\odot \in \{\oplus, \ominus\}$. Property ρ defines the event expected ($\oplus$) or prohibited ($\ominus$) and δ defines the deadline. Continuing the above example, δ_p represents any event with a timestamp greater than "$T_i + 28$" where T_i is the timestamp of the invoice event (the unit of time is supposed to be the calendar day). The intended meaning of $\oplus\langle \rho_p, \delta_p \rangle$ is thus that an event satisfying property ρ_p should occur before any event satisfying property δ_p. In the expression $\oplus\langle \rho_s, \mathbf{ff} \rangle$ defining the contrary to duty obligation, the deadline is $\mathbf{ff}$, which means that no constraint is imposed on the occurrence of an event satisfying property ρ_s. Depending on the situation, this may a deliberate omission or an oversight that will be revealed by the expression of the obligation in FLAVOR.

Conditions and contexts (Section II-C).: Application contexts are expressed by the constructors $\langle \rho, \delta \rangle \rightsquigarrow \varphi$ and $\langle \rho, \delta \rangle \dot{\rightsquigarrow} \varphi$, where ρ is the trigerring event and δ is the event defining the termination of the rule. $\rightsquigarrow$ represents multiple occurrence triggers and $\dot{\rightsquigarrow}$ single occurrence triggers. For instance, in Table I, Clause 1 can be defined as $\langle \rho_d, \mathbf{ff} \rangle \dot{\rightsquigarrow} \oplus\langle \rho_a, \delta_a \rangle$, where ρ_d represents the receipt of the software at timestamp T_d, ρ_a the payment of the amount of twenty thousand Euros and δ_a the occurrence of an event with a timestamp greater than $T_d + 14$. Clause 2 is expressed by the obligation $\langle \rho_i, \mathbf{ff} \rangle \rightsquigarrow \oplus\langle \rho_p, \delta_p \rangle$, where ρ_i represents the receipt of a valid invoice and ρ_p the corresponding payment. Again, $\mathbf{ff}$ denotes the absence of termination date for the obligation.

[3]For instance a binary relation $Manager(x, y)$ for the hierarchy relation between employees within an organization.

Table III
DEFINITION OF $\sqcap$

$(\mathbf{ff}, i) \sqcap (\mathbf{ff}, j)$	$= (\mathbf{ff}, \min(i, j))$
$(\mathbf{ff}, i) \sqcap (\mathbf{tt}, j)$	$= (\mathbf{ff}, i)$
$(\mathbf{ff}, i) \sqcap \bot$	$= (\mathbf{ff}, i)$
$(\mathbf{tt}, i) \sqcap (\mathbf{ff}, j)$	$= (\mathbf{ff}, j)$
$\bot \sqcap (\mathbf{ff}, j)$	$= (\mathbf{ff}, j)$
$(\mathbf{tt}, i) \sqcap (\mathbf{tt}, j)$	$= (\mathbf{tt}, \max(i, j))$
otherwise	$= \bot$

B. Semantics

The formal semantics of FLAVOR is defined by the function $[\![\varphi]\!]_f$ which maps any term $\varphi \in \mathcal{L}$ and environment $f \in M$ into a function in $(E^\star \times \mathbb{N}) \to (\mathbb{B} \times \mathbb{N})_\bot$. Its definition is given by structural induction on the syntax of obligations. The complete definition is presented in Table IV. We use an auxiliary function $\sqcap$ which combines values of the semantic domain (Table III). Intuitively, this function acts as a three-valued conjunction, where breach takes precedence over satisfaction. Basically, the semantics function walks through the trace to find a *testifier* (fulfillment or breach) of the obligation. For a given trace $\sigma \in E^\star$ and index $i \in \mathbb{N}$, $[\![\varphi]\!]_f(\sigma, i)$ returns $(\mathbf{tt}, j)$ (respectively $(\mathbf{ff}, j)$) if the obligation φ is fulfilled (respectively breached) by the suffix of σ starting at position i and this fulfillment (respectively breach) is detected at position j in σ. If the obligation φ is neither fulfilled nor breached, then $[\![\varphi]\!]_f$ returns $\bot$.

The conditions in Table IV are evaluated in their order of appearance. If the end of the trace is reached ($|\sigma| < i$), the evaluation is inconclusive ($\bot$). For the atomic case $\oplus\langle\rho, \delta\rangle$ (resp. $\ominus\langle\rho, \delta\rangle$), the rule is breached (respectively satisfied) as soon as an event matches δ. The rule is satisfied (resp. breached) as soon as an event matches ρ *and* does not match δ. The rule $\langle\rho, \delta\rangle \rightsquigarrow \varphi$ is satisfied if all its instantiations are satisfied: when an event matching ρ is found, a new obligation $[\![\varphi]\!]_{f'}$ instanciated from the extended environment f' has to be satisfied. The $\langle\rho, \delta\rangle \dot{\rightsquigarrow} \varphi$ rule is defined in the same way but the associated obligation is triggered only once (the original obligation formula is discarded). A contrary to duty obligation $\psi \gg \varphi$ is satisfied if ψ is satisfied *or* if ψ is breached at some point j and φ is satisfied after this point j. This definition formally captures the fact that the penalty φ is *an alternative way* to satisfy the obligation if ψ is breached.

We conclude this section with a definition of the breach and satisfaction of a rule with respect to a trace.

Definition 2. *The rule φ is said to be:*

- satisfied *by f, σ if $\exists j$ such that* $[\![\varphi]\!]_f\ (\sigma, 0) = (\mathbf{tt}, j)$
- breached *by f, σ if $\exists j$ such that* $[\![\varphi]\!]_f\ (\sigma, 0) = (\mathbf{ff}, j)$
- pending *for f at the end of σ if* $[\![\varphi]\!]_f\ (\sigma, 0) = \bot$

IV. PROPERTIES

In this section, we show that the semantics of FLAVOR enjoys natural properties and provide a partial order on rules which captures their relative strengths.

The stability property states that when a rule is breached or satisfied at some point j of a trace, the trace can be extended from this point j without any impact on the result of the evaluation.

Property 1. *Stability*
If $[\![\varphi]\!]_f(\sigma, i) = (b, j)$ then $\forall \sigma' \in E^\star$, $[\![\varphi]\!]_f(\sigma/j \cdot \sigma', i) = (b, j)$ where (σ/j) is the prefix of σ of length j.

Proof: This property follows directly from the definition of the semantics in Table IV and Table III since the semantics function returns its result as soon as an index j where the obligation is breached or satisfied is found (irrespective of the suffix of the trace). The only interesting case is the first line of Table III which returns the minimum of the two indexes: the property follows from the examination of the four following cases which show that the result of the conjunction is actually independent of the value of the other term when one term returns $\mathbf{ff}$. ■

Definition 3. *Obliviousness*
If $[\![\varphi]\!]_f(\sigma, i) = \bot$ then j is said to be an oblivious index of (σ, i) for φ in f if $i \leq j \leq |\sigma|$ and
$\forall \sigma' \in E^\star, [\![\varphi]\!]_f(\sigma \cdot \sigma', i) = [\![\varphi]\!]_f(\sigma \backslash j \cdot \sigma', 0)$ where
$\sigma \backslash j$ is the suffix of σ of length $|\sigma| - j$ (σ without its first j elements).

Obliviousness is a significant property in the context of *a posteriori* analysis because it provides a justification for the deletion (or storage in archive files) of a trace up to a given point after an audit procedure. The definition guarantees that future audits to check pending obligations can be safely applied to the suffix of the trace up to the oblivious index. In practice, an oblivious index derived from the semantics of Table IV can be returned by the audit analyzer[4].

We proceed now with the definition of two extreme situations, respectively unbreachable and unsatisfiable obligations.

Definition 4. *Unbreachability and unsatisfiability*
An obligation φ is unbreachable (resp. unsatisfiable) in a given environment f if $\forall \sigma \in E^\star, \forall j \in \mathbb{N}, [\![\varphi]\!]_f\ (\sigma, 0) \neq (\mathbf{ff}, j)$ (resp. $\neq (\mathbf{tt}, j)$).

Property 2. *If $\forall e \in E, match\ (\delta, e)\ f = \bot$, then:*

- *$\oplus\langle\rho, \delta\rangle$ is unbreachable in f.*
- *$\ominus\langle\rho, \delta\rangle$ is unsatisfiable in f.*
- *$\langle\rho, \delta\rangle \rightsquigarrow \varphi$ is unsatisfiable in f.*

The property follows directly from the semantics by induction on the length of the trace σ. Let us note that

[4]Space considerations prevent us from describing the computation of oblivious indexes here.

Table IV
DEFINITION OF THE SEMANTICS FUNCTION $[\![\varphi]\!]_f$

$$[\![\psi]\!]_f\ (\sigma,i) = \bot \text{ if } |\sigma| < i$$

$$[\![\oplus\langle\rho,\delta\rangle]\!]_f\ (\sigma,i) = \begin{cases} (\mathbf{ff},i) & \text{if } match\ (\delta,\sigma(i))\ f \neq \bot \\ (\mathbf{tt},i) & \text{if } match\ (\rho,\sigma(i))\ f \neq \bot \\ [\![\oplus\langle\rho,\delta\rangle]\!]_f\ (\sigma,i+1) & \text{otherwise} \end{cases}$$

$$[\![\ominus\langle\rho,\delta\rangle]\!]_f\ (\sigma,i) = \begin{cases} (\mathbf{tt},i) & \text{if } match\ (\delta,\sigma(i))\ f \neq \bot \\ (\mathbf{ff},i) & \text{if } match\ (\rho,\sigma(i))\ f \neq \bot \\ [\![\ominus\langle\rho,\delta\rangle]\!]_f\ (\sigma,i+1) & \text{otherwise} \end{cases}$$

$$[\![\langle\rho,\delta\rangle \rightsquigarrow \varphi]\!]_f\ (\sigma,i) = \begin{cases} (\mathbf{tt},i) & \text{if } match\ (\delta,\sigma(i))\ f \neq \bot \\ [\![\varphi]\!]_{f'}\ (\sigma,i+1) \sqcap [\![\langle\rho,\delta\rangle \rightsquigarrow \varphi]\!]_f\ (\sigma,i+1) & \text{if } match\ (\rho,\sigma(i))\ f = f' \\ [\![\langle\rho,\delta\rangle \rightsquigarrow \varphi]\!]_f\ (\sigma,i+1) & \text{otherwise} \end{cases}$$

$$[\![\langle\rho,\delta\rangle \dot{\rightsquigarrow} \varphi]\!]_f\ (\sigma,i) = \begin{cases} (\mathbf{tt},i) & \text{if } match\ (\delta,\sigma(i))\ f \neq \bot \\ [\![\varphi]\!]_{f'}\ (\sigma,i+1) & \text{if } match\ (\rho,\sigma(i))\ f = f' \\ [\![\langle\rho,\delta\rangle \dot{\rightsquigarrow} \varphi]\!]_f\ (\sigma,i+1) & \text{otherwise} \end{cases}$$

$$[\![\psi \gg \varphi]\!]_f\ (\sigma,i) = \begin{cases} (\mathbf{tt},j) & \text{if } [\![\psi]\!]_f\ (\sigma,i) = (\mathbf{tt},j) \\ [\![\varphi]\!]_f\ (\sigma,j) & \text{if } [\![\psi]\!]_f\ (\sigma,i) = (\mathbf{ff},j) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\psi \wedge \varphi]\!]_f\ (\sigma,i) = [\![\psi]\!]_f\ (\sigma,i) \sqcap [\![\varphi]\!]_f\ (\sigma,i)$$

unbreachable or unsatisfiable obligations are not necessarily pathological: in some situations, it may be meaningful to define prohibitions which must hold forever or obligations to perform certain actions within unspecified deadlines. In any case, it is interesting to be able to detect the occurrence of such obligations to check that they reflect the real intention of the actors.

In order to analyze obligations, it is also useful to be able to reason about their relative strengths. Intuitively, a rule φ_1 is stronger than a rule φ_2, written $\varphi_1 \succcurlyeq \varphi_2$, if φ_1 is breached by more traces and satisfied by less traces than φ_2. The following definition introduces an ordering between obligations based on the possible results of the semantics function.

Definition 5. *Obligation ordering*

$\varphi_1 \succcurlyeq \varphi_2$ (φ_1 is stronger than φ_2) if and only if
$\forall f \in M, \forall \sigma \in E^\star, \forall i \in \mathbb{N}$,
$[\![\varphi_1]\!]_f\ (\sigma,i) \geq [\![\varphi_2]\!]_f\ (\sigma,i)$
with $\forall i,j \in \mathbb{N}, (\mathbf{ff},i) \geq \bot \geq (\mathbf{tt},j)$

We can now establish some properties reflecting the intuition of Section II on the relative strengths of obligations:

Property 3. *Strength properties*

For all $\varphi,\psi \in \mathcal{L}$, for all $\rho,\delta \in P$:

- $\varphi \wedge \psi \succcurlyeq \varphi$
- $\varphi \wedge \psi \succcurlyeq \psi$
- $\varphi \succcurlyeq (\varphi \gg \psi)$
- $\langle\rho,\delta\rangle \rightsquigarrow \varphi \succcurlyeq \langle\rho,\delta\rangle \dot{\rightsquigarrow} \varphi$

Property 4. *Monotonicity*

For all $\psi,\varphi,\varphi_1,\varphi_2 \in \mathcal{L}$,
$\varphi_1 \succcurlyeq \varphi_2 \Rightarrow \psi[\varphi_1/\varphi] \succcurlyeq \psi[\varphi_2/\varphi]$
where $\psi[\varphi_i/\varphi]$ denotes the obligation ψ in which the occurrences of φ are replaced by φ_i.

The proofs of these properties follow directly from the definition of the semantics function.

V. EXAMPLES

In this section, we come back to the examples of Section II and show that FLAVOR, despite its minimal syntax, is expressive enough to represent typical examples of legal obligations. We also show that the formal semantics of the language makes it possible to avoid ambiguities or oversights in the original texts.

To describe the content of events in the trace, we use a set of basic keys $K = \{\texttt{type},\texttt{time},\texttt{sender},\texttt{dest}\}$. For the sake of readability, an event of type E sent from S to D at time T is written $E^T_{S\to D}$ as a shorthand for $[\texttt{type} \mapsto E, \texttt{time} \mapsto T, \texttt{sender} \mapsto S, \texttt{dest} \mapsto D]$. Irrelevant values are either written with the generic placeholder "x" or omitted. To ensure a natural interpretation of traces as logs, we assume that the timestamps of the events in the traces are increasing: $i \leq j \leq |\sigma| \Rightarrow \sigma(i)(\texttt{time}) \leq \sigma(j)(\texttt{time})$

A. Business agreement

Table V presents the FLAVOR expression corresponding to the excerpt of business agreement introduced in Table I. We use the additional key $\{\texttt{amount}\}$ as a parameter A

Table V
EXCERPT OF BUSINESS AGREEMENT OF TABLE I IN FLAVOR

$$\begin{aligned}
&\langle soft^{T_d}_{S\to C}, \mathbf{ff}\rangle \dot{\rightsquigarrow} \\
&[\oplus\langle pay(20,000)_{C\to S}, x^{T_a} \wedge (T_a \geq T_d + 14)\rangle \gg \\
&\oplus\langle pay(21,000)_{C\to S}, \mathbf{ff}\rangle] \\
&\wedge \\
&\langle inv(A)^{T_i}_{S\to C}, \mathbf{ff}\rangle \rightsquigarrow \\
&[\oplus\langle pay(A)_{C\to S}, x^{T_p} \wedge (T_p \geq T_i + 28)\rangle \gg \\
&\oplus\langle pay(A')_{C\to S} \wedge A' = 1.05 \times A, \mathbf{ff}\rangle]
\end{aligned}$$

in the expression $E(A)^T_{S\to D}$ to refer to invoice and payment amounts. In this example, the set of event types is $\{soft(ware), inv(oice), pay(ment)\}$. Clause 1 of Table I is a simple one-shot obligation, which is expressed in FLAVOR using the ($\dot{\rightsquigarrow}$) constructor. The relation between the payment (type pay) and the receipt of the software (type $soft$) is established through the sharing of the variables S (the supplier), C (the customer) and T_d (the date when the software is received). The deadline is missed by *any* event with a timestamp T_a greater than or equal to $T_d + 14$ days.

The fact that FLAVOR is endowed with a formal semantics enables a formal analysis of the definition of Table V which reveals that the agreement contains at least an implicit reference and an ambiguity. In Table V, it is assumed that Clause 2 is independent from Clause 1. This may be discussed, as Clause 2 probably refers to additional services related to the software of Clause 1. It is possible to express this alternative interpretation by using the conjunction in the right-hand side of $\dot{\rightsquigarrow}$ rather than at the highest level. Property 3 shows that an expression with a contrary to duty obligation is more permissive than without. In the present case, the lack of contrary to duty obligation is clearly too permissive: since there is no deadline associated with the penalty, from Property 2, we know that it cannot be breached. A solution is to add an explicit deadline for the penalty, for instance, half the delay of the original payment.

This example illustrates the benefits of including contexts in a specification language for legal rules. For each invoice, whatever the associated price is, a new obligation to pay is triggered for this specific price, which cannot be captured by a propositional language (or only at the price of a heavy encoding). Note that contexts allow us to express more complex penalty systems in FLAVOR. For example, the wording "interest shall be charged at an annual rate of 10% applicable beyond the due date" can be expressed as:

$$pay(A')^{T_p}_{C\to S} \wedge A' = A \times (1 + \frac{T_p - (T_i + 14)}{365} \times 10\%)$$

B. Privacy policy

Table VI presents the definition of the privacy policy of Table II in FLAVOR. Basic set of keys $K = \{\texttt{type}, \texttt{time}, \texttt{sender}, \texttt{dest}\}$ is extended with keys for collected data (`about`), deadline (`delay`), retention period (`expiration`), transfer policy (`policy`) and third party requesting for data transfer (`party`). With the convention defined at the beginning of section V, pattern $pii(D, \delta_d, \delta_e, inform)^{T_i}_{S\to C}$ is read "*any pii event from S to C at time T_i, about data D, delay δ_d, retention period δ_e and inform transfer policy*". Parameter X in ask events stands for the `party` involved in transfer. Clause 1 ("answer queries within δ_d") of Table II is formalized into FLAVOR by rule φ_1. Clause 2 ("delete data within δ_e") is formalized by rule φ_2. Third clause is divided into three cases: "transfer to third party is forbidden" (value $none$), "controller must inform" (value $inform$) and "controller cannot transfer without prior authorization" (value $auth$). Each case is formalized by the corresponding rules φ_n, φ_i and φ_a.

The need to record the preferences expressed by the subject is a distinguishing feature of privacy policies: in the *sticky policy* paradigm, personal data (variable D in the example) and privacy preferences (variables δ_d and δ_e, and `policy`'s value) are glued together (by pii events). The system must record and update its set of pending obligations whenever a piece of personal information is collected. These obligations are used when new events related to the subject have to be dealt with (e.g., request for transfer to third party). This kind of example is another illustration of the benefit of including contexts within the language.

The excerpt of privacy policy of Table II does not involve contrary to duty obligations. Actually, this feature can be used to overcome the limitations of existing privacy policy languages which cannot express the fact that certain actions may need to be taken in case of breach. As an illustration, the new European ePrivacy Directive includes a data breach notification obligation for telecom providers. Another example is the definition of privacy policies for electronic health record systems, where contrary to duty obligations can be used to express the fact that any unauthorized access (e.g. in case of emergency when the subject's live is at risk) must be followed by the registration of a record with a justification for the breach.

VI. RELATED WORK

Several extensions of Standard Deontic Logic (SDL) have been proposed to capture conditional activation, termination and temporal features [6]. Contrary to duty obligations have received considerable interest via defeasible and non-monotonic reasoning [12] or enriched interpretation structures [11], [17]. A major limitation of these approaches is that they inherently suffer from the paradoxes of the standard deontic framework. Moreover, propositional modal logics are not flexible enough to accommodate contexts and

Table VI
PRIVACY POLICY OF TABLE II IN FLAVOR

$$\varphi \stackrel{\text{def}}{=} \varphi_0 \wedge \varphi_n \wedge \varphi_i \wedge \varphi_a$$
$$\varphi_0 \stackrel{\text{def}}{=} \langle pii(D,\delta_d,\delta_e)^{T_i}_{S\to C}, \mathbf{ff}\rangle \rightsquigarrow (\varphi_1 \wedge \varphi_2)$$
$$\varphi_1 \stackrel{\text{def}}{=} \langle access(D)^{T_q}_{S\to C}, delete(D)_{C\to C}\rangle \rightsquigarrow \oplus\langle ans(D)_{C\to S}, x^{T_d} \wedge (T_d \geq T_q + \delta_d)\rangle$$
$$\varphi_2 \stackrel{\text{def}}{=} \oplus\langle delete(D)_{C\to C}, x^{T_r} \wedge (T_r \geq T_i + \delta_e)\rangle$$
$$\varphi_n \stackrel{\text{def}}{=} \langle pii(D,\delta_d,\delta_e,none)^{T_i}_{S\to C}, \mathbf{ff}\rangle \rightsquigarrow \ominus\langle transfer(D)^{T_t}_{C\to X}, delete(D)_{C\to C}\rangle$$
$$\varphi_i \stackrel{\text{def}}{=} \langle pii(D,\delta_d,\delta_e,inform)^{T_i}_{S\to C}, \mathbf{ff}\rangle \rightsquigarrow \langle transfer(D)^{T_t}_{C\to X}, delete(D)_{C\to C}\rangle \rightsquigarrow \oplus\langle notice(D)_{C\to S}, x^{T} \wedge (T \geq T_t + \delta_d)\rangle$$
$$\varphi_a \stackrel{\text{def}}{=} \langle pii(D,\delta_d,\delta_e,auth)^{T_i}_{S\to C}, \mathbf{ff}\rangle \rightsquigarrow \langle collect(D)_{X\to C}, delete(D)_{C\to C}\rangle \rightsquigarrow$$
$$\oplus\langle ask(D,X)_{C\to S}, transfer(D)^{T_t}_{C\to X}\rangle \wedge \ominus\langle transfer(D)^{T_t}_{C\to X}, auth(D,X)_{S\to C}\rangle\rangle$$

instantiations of rules. This problem has received attention with *real-time* temporal logics [1]. The use of variables in FLAVOR is close to freeze-quantification, which extends temporal logic with explicit references to timestamps of events. However temporal logics do not cope with the deontic modalities which are required to express legal rules.

A lot of work has been made by the business process community on regulatory compliance in organizations. The objective is to improve reliability and to minimize the risks of process failures. Although the focus is generally put on *a priori* verification techniques [3], [15], techniques for *a posteriori* regulatory compliance on traces have been proposed. For example, [9] defines a logic with deontic modalities involving conditional obligations and permissions. Emphasis is put on the ability to represent exceptions via references among norms.

A very expressive language called $\mathcal{CL}$ is proposed in [18]. $\mathcal{CL}$, which is based on the Propositional Dynamic Logic (PDL) and deontic logic, makes it possible to define rules on concurrent actions. Our approach clearly shares some of the motivations and technical choices of this work such as the restricted use of modalities. However, FLAVOR involves a very different treatment of conditions and provides facilities to define deadlines systematically. Technically, $\mathcal{CL}$ is a propositional language and it is not clear how to extend it to context dependent rules. Moreover, FLAVOR provides a more flexible use of contrary to duty obligations: it is not limited to atomic deontic formulae and can be applied to non atomic rules.

The need for parametric obligations in privacy policies and "on violation" statements has been identified in [7] which proposes a scalable obligation management framework able to deal with potentially large sets of personal data. This project is complementary to the work presented here which focuses on the formal definition of a general purpose obligation language, as opposed to a dedicated privacy policy language, and does not address scalability issues.

Strong arguments are put forward in [5] in favor of abstraction in the definition of privacy languages. FLAVOR shares this philosophy to some extent and does not define a specific set of actions since the semantics of events remains unspecified. In contrast with [5] however, FLAVOR involves two concepts that were felt as essential in a general obligation language : trigger and contrary to duty obligations. Actually, the focus in [5] is put on the verification that the privacy preferences defined by a subject are met by the potential data collector whereas our objective is rather to provide a formal framework for *a posteriori* verification of obligations, for example in the context of an audit procedure.

The interest of *a posteriori* compliance control has been strongly advocated in [8] and [10]. The audit logic proposed in [8] forms the basis for a formal audit procedure for *a posteriori* access control. In this framework, users may be asked to prove that their actions have been executed in compliance with discretionary access control policies. In contrast with FLAVOR (and similarly to [5]), the policy language makes it possible to express delegations through a "says" modality. The notion of obligation in [8] is not as general as obligations in FLAVOR however (action to be done in the future, with potential deadlines) and the language, which is dedicated to the management of data usage policies, does not involve contrary to duty obligations. The APPLE (A Posteriori PoLicy Enforcement) framework [10] is based on the notion of sticky policies associated with electronic documents and combines a logic for accountability with a trust management system. Another model for the formal definition of accountability and audit is proposed in [14] based on communicating process calculus and discrete timed process algebra. The main contribution of this work is the formalization of the audit process itself, with its potential limitations, and the study of its properties.

VII. CONCLUSION

The main challenge of the work described in this paper was to propose a language which is (i) expressive enough

to represent typical policies that organizations have to abide with, (ii) well-defined to avoid ambiguities and (iii) easy to implement, in particular through *a posteriori* checks.

The formal semantics introduced in Section III has been implemented in Haskell, a functional language based on lazy evaluation which allows us to translate the definition of Table IV in a rather direct and concise way (approximately 400 lines of code), thus minimizing the risks of discrepancy between the formal semantics and the implementation of the auditing tool. FLAVOR is embedded into Haskell as a Domain Specific Language (DSL): the set $\mathcal{L}$ is represented as a datatype and sentences are inhabitants of that type defined as Haskell expressions. The translation of the semantics definitions leads to an executable auditing tool for the language: given a trace and a FLAVOR sentence, the auditing tool returns the verdict "satisfied", "breached" or "pending". In the last case, it can also return an oblivious index as defined in Section IV to allow for incremental auditing. This implementation strategy greatly enhances the usability of the auditing tool as it can be easily instrumented and integrated with other software components.

For the sake of conciseness, we have introduced the key features of FLAVOR through simple examples in this paper. An extended version of the language presented here makes it possible to refer to previous violations in a chain of contrary to duty obligations. This need may arise for example when a more restrictive sanction is defined in case of a breach of a first sanction. Another extension concerns the introduction of a sequence operator $\lessdot$ between obligations which allows us to express the fact that an obligation ψ has to be triggered after a first obligation φ is satisfied (rather than breached in the case $\gtrdot$).

It is well known that the definition of temporal logic properties over finite traces involves some subtleties. The challenge stems from the inconclusive case that does not exist with infinite traces. Domains richer than $\{\mathbf{tt}, \mathbf{ff}\}$ have been proposed to deal with this issue in LTL [4]. These extensions would make it possible to refine the inconclusive case $\perp$ of our semantics into two cases:

- *not breached*, when nothing bad has happened yet,
- *not satisfied*, when nothing good has happened yet.

ACKNOWLEDGMENT

This work has been funded by ANR (Agence Nationale de la Recherche) under the grant ANR-07-SESU-005 (project FLUOR).

REFERENCES

[1] Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) REX Workshop. LNCS, vol. 600, pp. 74–106. Springer (1991)

[2] Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and contextual integrity: Framework and applications. In: Proceedings of the 2006 IEEE S&P. pp. 184–198 (2006)

[3] Barth, A., Mitchell, J., Datta, A., Sundaram, S.: Privacy and utility in business processes. In: Computer Security Foundations Symposium 07. pp. 279–294. IEEE Computer Society, Washington, DC, USA (2007)

[4] Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: Runtime Verification 07. pp. 126–138. Springer-Verlag, Berlin, Heidelberg (2007)

[5] Becker, M., Malkis, A., Bussard, L.: A practical generic privacy language. In: Jha, S., Mathuria, A. (eds.) Information Systems Security. LNCS, vol. 6503, pp. 125–139. Springer Berlin / Heidelberg (2011)

[6] Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In: Lomuscio, A., Nute, D. (eds.) DEON. Lecture Notes in Computer Science, vol. 3065, pp. 43–56. Springer (2004)

[7] Casassa Mont, M., Beato, F.: On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. In: Proceedings of the 8th IEEE POLICY. pp. 51–55 (2007)

[8] Cederquist, J.G., Corin, R., Dekker, M.A.C., Etalle, S., den Hartog, J.I., Lenzini, G.: Audit-based compliance control. Int. J. Inf. Secur. 6(2), 133–151 (2007)

[9] Dinesh, N., Joshi, A.K., Lee, I., Sokolsky, O.: Checking traces for regulatory conformance. In: Leucker, M. (ed.) Runtime Verification 08. LNCS, vol. 5289, pp. 86–103. Springer (2008)

[10] Etalle, S., Winsborough, W.: A posteriori compliance control. In: SACMAT'07 (2007)

[11] Gabbay, D.M.: Reactive kripke models and contrary to duty obligations. In: van der Meyden, R., van der Torre, L. (eds.) DEON. LNCS, vol. 5076, pp. 155–173. Springer (2008)

[12] Governatori, G., Milosevic, Z.: Dealing with contract violations: formalism and domain specific language. In: EDOC. pp. 46–57. IEEE Computer Society (2005)

[13] Jacobs, B.: Keeping our surveillance society non-totalitarian. Amsterdam Law Forum 1(4) (2009), http://ojs.ubvu.vu.nl/alf/article/view/91/156

[14] Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a theory of accountability and audit. In: Backes, M., Ning, P. (eds.) ESORICS. LNCS, vol. 5789, pp. 152–167. Springer (2009)

[15] Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal 46(2), 335–362 (2007)

[16] Pace, G.J., Schneider, G.: Challenges in the specification of full contracts. In: Leuschel, M., Wehrheim, H. (eds.) IFM. LNCS, vol. 5423, pp. 292–306. Springer (2009)

[17] Prakken, H., Sergot, M.J.: Contrary-to-duty obligations. Studia Logica 57(1), 91–115 (1996)

[18] Prisacariu, C., Schneider, G.: CL: An action-based logic for reasoning about contracts. In: Ono, H., Kanazawa, M., de Queiroz, R.J.G.B. (eds.) WoLLIC. Lecture Notes in Computer Science, vol. 5514, pp. 335–349. Springer (2009)