Towards runtime support for norm change from a monitoring perspective*

Ignasi Gómez-Sebastià, Sergio Álvarez Napagao, Javier Vázquez Salceda and Luis Oliva Felipe

Universitat Politècnica de Catalunya
Software Department (LSI)
c/Jordi Girona 1-3, E08034, Barcelona, Spain
{igomez, salvarez, jvazquez, loliva}@lsi.upc.edu

Abstract. Nowadays electronic specifications of norms are one of the mechanisms that can be applied to define and enforce acceptable behaviour within distributed electronic systems which should comply with some (human) regulations. As in human legal systems, it is easy to foresee that some of these electronic normative environments will not be static. They should be able to evolve through time as regulations change, effectively adapting to new situations and behaviours. In this paper we present an extension of a formal normative monitoring framework capable of updating normative contexts at runtime without stopping the monitoring process.

Keywords: normative systems; normative monitoring; runtime support

1 Introduction

Electronic specifications of norms can be applied to define and enforce acceptable behaviour within distributed electronic systems, especially those that should comply with some regulations. Some of these electronic normative systems will not be static, will evolve through time as regulations change, adapting to new situations and behaviours.

One of the requirements to effectively implement Normative Systems is to be able to assess, at runtime, the state of the normative environment. Some existing lines of research (e.g., [3] and [13]) have already tried to tackle this issue on some simple scenarios. However, more complex scenarios may appear, for instance, scenarios where the normative context that defines the normative environment is not static, but it expands and contracts as new norms are added to the institution and removed from it respectively. Under these conditions, a monitoring system must be able to continue computing the state of the normative environment at runtime, as often we can not afford to perform the changes on the normative context off-line. Furthermore, it must be guaranteed the monitoring system can keep producing states of the normative environment that are consistent with the changes performed on the normative context. For instance, if a norm has been removed from the normative context, it makes no sense any more to compute normative states where the norm has been violated.

* AT2012, 15-16 October 2012, Dubrovnik, Croatia.

In this paper we present an approach for extending the normative monitoring framework presented in [3] allowing it to support normative expansion and contraction operations at runtime, without having to stop computing the normative state and, at the same time, computing states that are consistent with the expansion and contraction operations performed. The framework focuses on norm monitoring from an institutional perspective (*e.g.*, detecting violations of norms, so institutional agents can enforce sanctions and repair actions) without neglecting agent's ability to query the normative state, effectively allowing agents to make sure they comply with the norms in the institution.

We illustrate our approach via a scenario that models a simplified version of the 2005 Spanish smoking law that has been amended in 2011. Basically, the 2005 law obliges bars and restaurants with a size bigger than $100m^2$ to provide an isolated area for smoking customers. They will incur in a violation if they do not fulfil this obligation and the violation is considered as repaired once the bar habilitates an area for their smoking customers. The amended 2011 law forbids bars and restaurants to have any smoking area. They will incur in a violation if they have a smoking area and the violation will be considered as repaired once the bar removes the smoking area. We will use this scenario for providing examples along this paper.

The rest of this paper is structured as follows: in *Section* 2 we summarise the monitoring formalism presented in [3] which we extend in this work. Then in *Section* 3 we formalise the operations to be supported in order to allow for normative context expansion and contraction. We also extend the formal framework providing support for a more expressive norm life cycle. Next, in *Section* 4 we provide formal algorithms for implementing the expansion and contraction operations on the existing framework. *Section* 5 puts our proposal in contrast with existing approaches. Finally *Section* 6 presents our conclusions and outlines future lines of research.

2 Monitoring Formalism

In this section we summarise the formalism for monitoring normative systems which we will use in the rest of the paper. For more details on this formalism, please refer to [3].

We assume the use of a predicate based propositional logic language $\mathcal{L}_O$ with predicates and constants taken from an ontology O, and the logical connectives $\{\neg, \vee, \wedge\}$. The set of all possible well-formed formulas of $\mathcal{L}_O$ is denoted as $wff(\mathcal{L}_O)$ and we assume that each formula from $wff(\mathcal{L}_O)$ is normalised in Disjunctive Normal Form (*DNF*). Formulas in $wff(\mathcal{L}_O)$ can be partially grounded, if they use at least one free variable, or fully grounded if they use no free variables.

We define the *state of the world* s_t as the set of predicates holding at a specific timestamp t, where $s_t \subseteq O$, and we denote $\mathcal{S}$ as the set of all possible states of the world, where $\mathcal{S} = \mathcal{P}(O)$. We call *expansion* $F(s)$ of a state of the world s as the minimal subset of $wff(\mathcal{L}_O)$ that uses the predicates in s in combination of the logical connectives $\{\neg, \vee, \wedge\}$. We define a substitution instance $\Theta = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, ..., x_i \leftarrow t_i\}$ as the substitution of the terms $t_1, t_2, ..., t_i$ for variables $x_1, x_2, ..., x_i$ in a formula $f \in wff(\mathcal{L}_O)$. Thus, $\Theta(f(x_1, x_2, ..., x_i)) \equiv f(t_1, t_2, ..., t_i)$. We denote as $\vartheta_{(wff(\mathcal{L}_O),\mathcal{S})}$

the set of all possible substitution instances containing the variables in $wff(\mathcal{L}_O)$ and the terms in $\mathcal{S}$.

Definition 1. (Norm) *A 'norm' n is a tuple $n = \langle f_A, f_M, f_D, f_w, w\rangle$, where*

- $f_A, f_M, f_D, f_w \in wff(\mathcal{L}_O)$, $w \in O$,
- f_A, f_M, f_D *respectively represent the activation, maintenance, and deactivation conditions of the norm.*
- f_w *is the explicit representation of the target of the norm, and* w *is the subject of the norm (role or agent).*

A norm is defined in an abstract manner, affecting all possible participants enacting a given role. Whenever a norm is active, we will say that there is a *norm instance* $ni = \langle n, \theta\rangle$ for a particular norm n and a substitution instance Θ.

We can formalise the norms of Definition 1 as the equivalent deontic expression (using the inference rules formalism in [7]):

Property 1. A norm is considered fulfilled if, and only if:

$$f_A \rightarrow [O_w(E_w f_w \leq \neg f_M)\,\mathcal{U}\, f_D]$$

where $O_w(E_a p)$ means that agent a has the obligation to *see to it that* (*stit*) p becomes true and U is the CTL* until operator.

Intuitively, Property 1 states that after the norm activation, the subject is obliged to see to it that the target becomes true before the maintenance condition is negated (either the deadline is reached or some other condition is broken) until the norm is deactivated (which is either when the norm is fulfilled or has otherwise expired).

Definition 2. (Violation handling norm[1]) *A norm $n' = \langle f'_A, f'_M, f'_D, f'_w, w'\rangle$ is a violation handling norm of $n = \langle f_A, f_M, f_D, f_w, w\rangle$, denoted as $n \rightsquigarrow n'$ iff $f_A \wedge \neg(f_M \mathcal{U} f_D) \vdash f'_A$*

Violation handling norms are special in the sense that they are only activated once another norm is violated. Please notice we consider a norm is not violated if the maintenance condition is kept until the deactivation condition holds, and is violated otherwise. Violation handling norms are used as *sanctioning norms*, if they are to be fulfilled by the norm violating actor (*e.g.*, the obligation to pay a fine if the driver broke a traffic sign), or as *reparation norms*, if they are to be fulfilled by an institutional actor (*e.g.*, the obligation of the authorities to fix the broken traffic sign).

One common problem for the monitoring of normative states is the need for an interpretation of brute events as institutional facts, also called constitution of social reality[11]. The use of *counts-as rules* helps solving this problem. Counts-as rules are multi-modal statements of the form $[c](\gamma_1 \rightarrow \gamma_2)$, read as "in context c, γ_1 *counts-as* γ_2". In our proposal, we consider a context as a set of predicates:

Definition 3. (Counts-as rule) *A counts-as rule is a tuple $c = \langle \gamma_1, \gamma_2, s\rangle$, where $\gamma_1, \gamma_2 \in wff(\mathcal{L}_O)$, and $s \subseteq O$.*

[1] Informally: the unfulfillment of the obligation of norm n entails the activation of norm n'.

A set of counts-as rules is denoted as C. Although the definition of counts-as in [11] assumes that both γ_1 and γ_2 can be any possible formula, in our work we limit γ_2 to a conjunction of predicates. This will ensure every well-formed-formula is on a standard *Disjunctive Normal Form*. Knowing before hand all well-formed-formula are on the same format, effectively simplifies the process of detecting information dependencies between well-formed formulas.

Definition 4. (Institution) *Following the definitions above, we define an* institution *as a tuple of norms, roles, participants, counts-as rules, and an ontology:*

$I = \langle N, R, P, C, O \rangle$ *where:* N, R, P, C *is a set of norms, roles, participants and counts-as rules respectively and* O *is an Ontology.*

In order to track the normative state of an institution at any given point of time, the state of each of the norms inside the context should be tracked. In order to ease this task, we define three sets: an instantiation set IS, a fulfillment set FS, a violation set VS, and a repairment set RS. Each of them contains norm instances $\{\langle n_i, \Theta_j \rangle, ..., \langle n_{i'}, \Theta_{j'} \rangle\}$. In order to define the states a norm may be in, we adapt the semantics for normative states from [13]:

Definition 5. (Norm Life-cycle) *Let* $ni = \langle n, \Theta \rangle$ *be a norm instance, such that* $n = \langle f_A, f_M, f_D, w \rangle$*, and* s *be a state of the world with an expansion* $F(s)$*. Then we define the life-cycle for a norm instance* ni *by the following normative state predicates:*

$activated(ni) \Leftrightarrow \exists f \in F(s), \Theta(f_A) \equiv f$
$maintained(ni) \Leftrightarrow \exists \Theta', \exists f \in F(s), \Theta'(f_M) \equiv f \wedge \Theta' \subseteq \Theta$
$deactivated(ni) \Leftrightarrow \exists \Theta', \exists f \in F(s), \Theta'(f_D) \equiv f \wedge \Theta' \subseteq \Theta$
$instantiated(ni) \Leftrightarrow ni \in IS$
$violated(ni) \Leftrightarrow ni \in VS$
$fulfilled(ni) \Leftrightarrow ni \in FS$
$repaired(ni, ni') \Leftrightarrow \langle ni, ni' \rangle \in RS$

Where IS *is the instantiation set,* FS *is the fulfilment set,* VS *is the violation set, and* RS *is the set of those norm instances* ni' *that have repaired a norm instance* ni*.*

Definition 6. (Event) *An event* e *is a tuple* $e = \langle \alpha, t, p \rangle$*, where*

- $\alpha \in O$*, an actor of the system,*
- t *is the timestamp of the reception of the event, and*
- *given a fully grounded subset of the set of states of the world* $p' \in S : p = p' \vee p = \neg p'$

We define E as the set of all possible events, $E = \mathcal{P}(P \times t \times S)$ where t is a timestamp. From these definition we can formalise the concept of *Normative Monitor* and the concept of *Labelled Transition System for a Normative Monitor* as follows:

Definition 7. (Normative Monitor) *A Normative Monitor* M_N *for a set of norms* N *is a tuple* $M_N = \langle N, S, IS, VS, FS, E \rangle$*.*

Where: $S = \mathcal{P}(O) \wedge IS = \mathcal{P}(N \times S \times Dom(S)) \wedge VS = \mathcal{P}(N \times S \times Dom(S)) \wedge FS = \mathcal{P}(N \times S \times Dom(S)) \wedge E$ *is the set of all possible events as defined before.*

	Ex Tunc	*Ex Nunc*
Context expansion	Retroactive Promulgation	Prospective Promulgation
Context contraction	Annulment	Abrogation

Fig. 1. Possible operations on norms

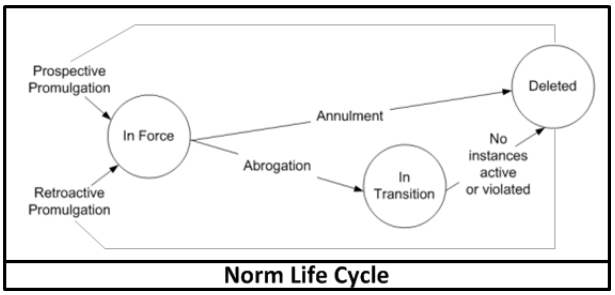

Fig. 2. Extending the norm life-cycle

Γ_{M_N} is the set of all possible configurations of a Normative Monitor M_N.

Definition 8. (Labelled Transition System) *The* Labelled Transition System LTS_{M_N} *for a Normative Monitor* M_N *is defined by* $LTS_{M_N} = \langle \Gamma_{M_N}, L, \rhd \rangle$ *where*

- $L = \{ep, nii, niv, nif, nir\}$ *is a set of labels, respectively representing* event processed*,* norm instantiation*,* norm instance violation*,* norm instance fulfilled*, and* norm instance violation repaired*, and*
- $\rhd$ *is a* transition relation *such that* $\rhd \subseteq \Gamma_{M_N} \times L \times \Gamma_{M_N}$

This formalism has been reduced to the semantics of general production systems and an implementation in DROOLS is already available.

3 A formal framework for norm change

This section defines the operations required for supporting expansion and contraction of normative contexts. It also depicts the norm life-cycle extension required for supporting these operations.

According to legal literature, one can specify two types of change operations on a normative context, context expansion (adding norms) and context contraction (removing norms; norm updates can be seen as norm removal followed by a norm addition). Each of these operations comes in two forms: *Ex Tunc*(*i.e.*, from the outset) and *Ex Nunc* (*i.e.*, from now on). Both terms are Latin legal terms which are common in law literature. An *Ex Tunc* norm is a norm that retroactively changes the normative consequences (or status) of actions committed prior to the existence of the norm, whereas an

Ex Nunc norm affects only actions committed after the existence of the norm. To summarize, two operations with two forms, means one can apply four distinct operations to normative contexts, as depicted on *Figure* 1. We can provide a more formal definition of the four operations:

- ***Prospective promulgation***: Introduces a new norm on the normative context. Events happening on the context can instantiate the norm as soon as it has been promulgated. The norm will not check for violations caused by past events, but it can be activated by past events. This is because if a given fact has been made true in the past we assume it to be true until we have proof of the contrary. For instance, if we received the event *bornAt(Spain, Manolete)* in the past we can assume the fact *Manolete* is born in *Spain* still holds in the present. Thus, if a norm applies to (*i.e.*, is activated by) individuals born in Spain, it should apply to the individual *Manolete* even if he was born before the norm promulgation.
- ***Retroactive promulgation***: Introduces a new norm on the normative context. Events happening on the context can instantiate the norm as soon as it is promulgated. The norm will check for norm instance activations or violations caused by past events. *Retroactive promulgation* can lead to a massive amount of norm instances being violated (especially if the number of past events is high). Few scenarios should require this operation for normative context modification. In fact, most real-world normative contexts forbid this operation (*e.g.*, most countries forbid retrospective law promulgation on their constitutions and bills of rights).
- ***Annulment***: Removes a norm from the normative context. All the instances of the norm are removed as well, including violated ones. This implies removing sanctions and repair actions that are yet to be enacted. Repair actions already enacted must be de-enacted, so the agents responsible of enacting repair actions (*e.g.*, institutional agents managing the institution) must be aware of the annulment. As an example, if someone is imprisoned for violating an *annulled* norm, he/she should be set free.
- ***Abrogation***: Tags a norm from the normative context as being *In transition*. Therefore, the norm can not be instantiated any more. Instances of the norm remain in the normative context as long as they are not in a terminal state (that is, either fulfilled or repaired). Once all norm's instances have reached a terminal state, the norm is removed from the system along with its instances. As an example, if someone is imprisoned for violating an *abrogated* norm, he/she should remain imprisoned until the violation has been repaired (*i.e.* the offender has spent a given amount of time imprisoned). However, once set free, he/she should be able to violate the norm again without any legal consequences.

We have introduced some extra norm states when defining the expansion and contraction operations. Therefore, in order to effectively support these operations we have to extend our norm life-cycle, as depicted in *Figure* 2, adding the following states:

- In force: Once a norm has been promulgated (either in a prospective or retroactive form) it achieves an *In force* state. From this point, the norm can be effectively activated if its activation condition is met. In some scenarios, norms introduced in the system off-line (*i.e.*, they are already there when the system starts its execution)

are by default in this state. In other scenarios they can lack this state (*i.e.*, be in *Deleted* stated instead) and are moved to it by institutional agents in case they consider they are beneficial for the overall goals of the system. This will effectively provide agents with a pool of norms that can be promulgated (either in a prospective or retroactive form) if required. A mixture of both approaches is also possible, where system designers put some important norms *In force* since the beginning of system's execution, leaving a second set of norms in the pool of norms that can be put into force by institutional agents.

- In transition: Abrogated norms go into this state. It means the norm can not be instantiated any more. However, instances of the norm that have been already instantiated (*i.e.*, they are in *active* or *violated* state) remain in the system. Once these instances change to *Fulfilled* or *Repaired* states the norm *in transition* can effectively move on to *Deleted* state.
- Deleted: As stated before, abrogated norms with no active instances are moved to this state. Annulled norms are also moved to this state, no matter if they contain active instances or not. Therefore, active instances of annulled norms are removed from the system. Mechanisms based on the already available violation handling can be defined to compensate for violated instances of annulled norms that have been repaired (*e.g.*, if an agent pays a fine for violating a norm, and then the norm is annulled, return the amount paid to the agent). Deleted norms can be promulgated again either via prospective or retroactive promulgation.

Two lines of research have already tackled this issue, Aucher et al. [5] and Governatori et al. [10]. The first one defines both expansion and contraction operations, but only supports *ex tunc* operations. The second line of research supports both *ex tunc* and *ex nunc* operations but it focuses on context expansion. Our approach can be seen as an attempt to mix the expressivity of both proposals, defining at the same a richer norm life-cycle able to cope with normative context modification operations.

Figure 3 shows examples of some operations on a normative context. Specifically it shows how the norm (*N1*) and the amendment (*N2*) are activated, fulfilled, violated or repaired depending on the events happening on the environment.

Basically *N1* is activated as soon as bar instances with more than $100m^2$ are detected on the environment. Notice how both prospective and retroactive norm promulgation check for past events. In this case, it is because if in the past a bar instance had more than $100m^2$ we assume this fact to be still true if we have not proof of the contrary. Once promulgated, the norm goes to fulfilled state if the bar has an isolated area for smoking customers, and to violated state if it does not. If the norm is violated, violation can be repaired by creating an isolated area for smokers. This action will take the norm to repaired state.

Then, norm *N1* is abrogated and *N2* promulgated. Once promulgated *N2* instantiates as soon as bar instances are detected, no size constraints are to be met. The norm is fulfilled if the bar has no smoking area, and violated if it does. If the norm is violated, violation can be repaired by removing the isolated smoking area from the bar. As norm *N1* has been abrogated, Bars that are on violated state on instances of *N1* have to perform their repair actions in order to move to repaired state even if *N1* has been already abrogated and *N2* promulgated. It would not be the case if norm *N1* was annulled in-

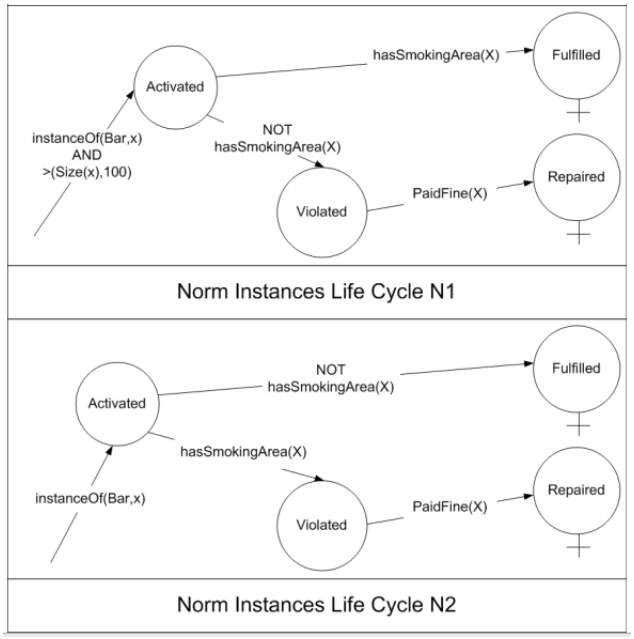

Fig. 3. Example of the execution of norms N1 and N2

stead of abrogated as annulment would force norm violations to be removed from the system, even if such violations have not been repaired.

This simple scenario allows us to reinforce the idea system designers need to be very careful when using norm abrogation. Bars that have spent money in creating areas for smokers due to the promulgation of *N1*, have to spend money again for removing these areas once norm *N2* is promulgated. Otherwise they would be violating *N2*. Therefore, in order to achieve a fair norm promulgation, system designers should include a reward for bars that fulfilled norm *N1* by creating areas for smokers when promulgating norm *N2*. The situation is even worse for bars with more than $100m^2$ that do not have an isolated area for smokers. Once norm *N1* is abrogated and *N2* promulgated, these bars have to create an isolated area for smokers, effectively repairing the violation of *N1*. But once this area is created, they are violating *N2*, and have to disable the area in order to repair the violation of *N2*. This would have been avoided if norm *N1* was annulled instead of abrogated. This fact reinforces the idea norm design has to be carefully analysed when expanding and contracting the normative context, in order to bring the system closer to its overall (multiple and sometimes even conflicting) goals. In particular, it depicts how norm designers have to carefully balance the benefits of norm abrogation and annulment, choosing the correct operation for contracting the normative context on every situation.

4 Implementing Norm change support

In this section we show how the monitoring formalism presented in [3] can be extended to support normative context modifications at run-time. We also depict the different normative context operations that our monitoring system supports and the pseudo-code algorithms required for fulfilling each normative context operation form a monitor component perspective.

If norms are represented as rules, then rule change can be represented as non-monotonic inference. According to [9], changing a normative system would amount to adding new rules or removing the existing ones. The formalism presented in [3], based on rule creation from normative specifications, allows us to define the operationalisation of norm change, in its four forms, as extensions of the main monitoring process. By using all or some of the labels described in *Definition* 8, we can constrain the exact normative-related actions that the monitor will be able to carry out, and thus we can define and create, at runtime, diverse monitoring *contexts*.

The ability to create a different monitoring context is due to the fact that some norm changes can be retroactive. Thus the only way to generate a normative state compliant with the one we had before the norm change is to analyse the full stream of events generated since the beginning of the monitoring process. Such a monitoring context can be created at runtime by using two constructs from the monitoring formalism: the *normative monitor* and the labels of its *labelled transition system*. With the first one we can create a new monitor, specific to a set of norms (the ones to be added), and by constraining the labels we can control the level of retro-activity of the norm change type.

How these constructs have to be used depends on each type of norm change, as defined in *Section* 3. As seen on *Figure* 1 there are a total of four possible operations on a normative context, formalised by the following *Algorithms*:

- **Prospective Promulgation *Algorithm* 1 *Figure* 4**: The norm is inserted into the normative monitor. A secondary monitor computes the instantiations of the norm caused by past events. Once computed, the instantiations are inserted on the normative monitor.
- **Retrospective Promulgation *Algorithm* 2 *Figure* 5**: The norm is inserted into the normative monitor. A secondary monitor computes the following norm states caused by past events: norm instantiation, norm fulfilment, norm violation and norm repair. Once computed, the states are inserted on the normative monitor.
- **Annulment *Algorithm* 3 *Figure* 6**: Instances of the norm in instantiated, repaired, violated or fulfilled state are removed from the normative monitor. In the case of instances in repaired state, the institutional agents are notified in case an action to undo the reparation is required. Then, the norm is removed from the normative monitor.
- **Abrogation *Algorithm* 4 *Figure* 7**: Instances of the norm in fulfilled or repair state are removed. Then, the norm goes to *transition* state. Therefore, the norm can not be instantiated any more. It is checked periodically if the norm contains instances in instantiated or violated state. If it does not, instances of the norm on fulfilled or repair state are removed. Then, the norm is removed from the system. If the norm

contains instances in instantiated or violated state, the process waits a given amount of time before checking again for instances in instantiated or violated state.

Algorithm 1 Prospective Promulgation of *PPNorm*

Require: $PPNorm = \langle f_A, f_M, f_D, f_w, w \rangle$
Require: $M_N = \langle N, S, IS, VS, FS, RS, E \rangle$
Require: $PPNorm \notin N$
$M_{N'} = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, N \cup \{PPNorm\}, E \rangle$
$LTS_{M_{N'}} = \{\Gamma, \{nii\}, \triangleright\}$
$engine.create(LTS_{M_{n'}})$
$it = E.iterator$
while $it.hasNext$
$engine.insert(it.next)$
$engine.infer$
$M_N.IS = M_N.IS \cup M_{N'}.IS$

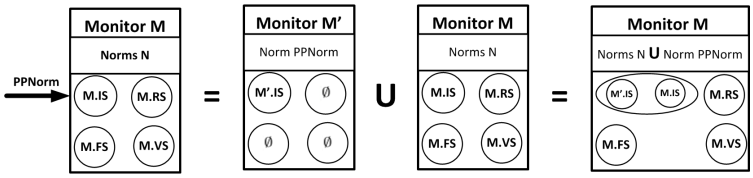

Fig. 4. Diagram of the prospective promulgation of PPNorm

5 Related work

There is currently an important amount of work being done in context change management, we have identified four main trends.

Governatori [10] proposes an extension of his logics for normative monitoring that enables capturing the different temporal aspects of *abrogation* and *annulment*. The extension increases the expressive power of his logics allowing it to represent meta-norms describing norm modifications. Meta-norms refer to a variety of possible time-lines through which conclusions, rules and derivations can persist over time. In particular, the extension defines temporal constraints that permit either allowing for or blocking persistency with respect to specific time lines. The idea behind Governatori's approach is blocking of derivations across repositories (*i.e.*, time-lines). When a modification is applied to the normative context, it is split into two repositories: where the modification occurs and where it does not. For instance, if a norm is *abrogated*, norm's conclusions

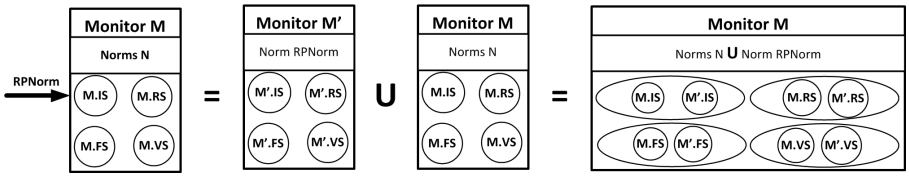

Fig. 5. Diagram of the retrospective promulgation of RPNorm

Algorithm 2 Retrospective Promulgation of *RPNorm*

Require: $RPNorm = \langle f_A, f_M, f_D, f_w, w \rangle$
Require: $M_N = \langle N, S, IS, VS, FS, RS, E \rangle$
Require: $RPNorm \notin N$
  $M_{N'} = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, N \cup \{RPNorm\}, E \rangle$
  $LTS_{M_{N'}} = \{\Gamma, \{nii, niv, nif, nir\}, \triangleright\}$
  $engine.create(LTS_{M_{n'}})$
  $it = E.iterator$
  while $it.hasNext$
    $engine.insert(it.next)$
    $engine.infer$
  $M_N.IS = M_N.IS \cup M_{N'}.IS$
  $M_N.VS = M_N.VS \cup M_{N'}.VS$
  $M_N.FS = M_N.FS \cup M_{N'}.FS$
  $M_N.RS = M_N.RS \cup M_{N'}.RS$

Algorithm 3 Annulment of *AnNorm*

Require: $AnNorm = \langle f_A, f_M, f_D, f_w, w \rangle$
Require: $M_N = \langle N, S, IS, VS, FS, RS, E \rangle$
Require: $AnNorm \in N$
  for all $ni \in IS$
    if $ni.norm == AnNorm$ **then** $IS = IS - ni$
  for all $ni \in VS$
    if $ni.norm == AnNorm$ **then** $VS = VS - ni$
  for all $ni \in FS$
    if $ni.norm == AnNorm$ **then** $FS = FS - ni$
  for all $\langle ni, ni' \rangle \in RS$
    if $ni.norm == AnNorm$ **then**
      $RS = RS - \langle ni, ni' \rangle$
      $notifyManager(reparationToRollback, ni')$
  $N = N - AnNorm$

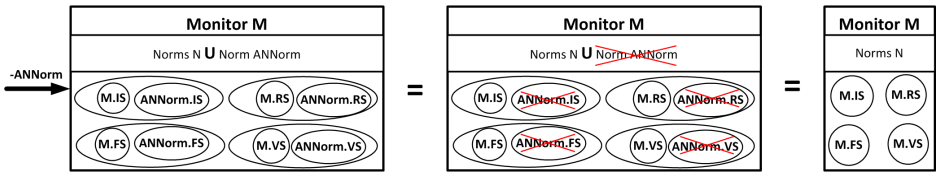

Fig. 6. Diagram of the annulment of ANNorm

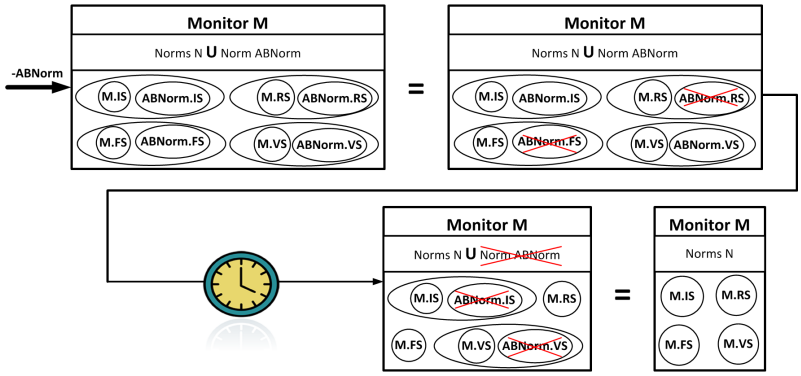

Fig. 7. Diagram of the abrogation of ABNorm

Algorithm 4 Abrogation of *AbNorm*

Require: $AbNorm = \langle f_A, f_M, f_D, f_w, w \rangle$
Require: $M_N = \langle N, S, IS, VS, FS, RS, E \rangle$
Require: $AbNorm \in N$
  for all $ni \in FS$
    if $ni.norm == AbNorm$ **then** $FS = FS - ni$
  for all $\langle ni, ni' \rangle \in RS$
    if $ni.norm == AbNorm$ **then** $RS = RS - \langle ni, ni' \rangle$
  {At this point, $AbNorm$ becomes $InTransition$}
  while $AbNorm \in N$
    $deleteNorm = true$
    for all $ni \in IS$
      if $ni.norm == AbNorm$ **then** $deleteNorm = false$
    for all $ni \in VS$
      if $ni.norm == AbNorm$ **then** $deleteNorm = false$
    if $deleteNorm$ **then**
      for all $ni \in FS$
        if $ni.norm == AbNorm$ **then** $FS = FS - ni$
      for all $\langle ni, ni' \rangle \in RS$
        if $ni.norm == AbNorm$ **then** $RS = RS - \langle ni, ni' \rangle$
      $N = N - AbNorm$
    $sleep(sometime)$

are derived only in the repositories where the rule has not been *abrogated*. When compared to our approach, Governatori's has the following drawbacks: 1) Norm *annulment* presents a problem under this approach, conclusions of *annulled* norms might remain on the repository after the norm has been *annulled*, and the solution proposed to remove the conclusion seems quite ad-hoc. 2) Governatori's solution provides no explicit support for *retroactive promulgation*. 3) Governatori's approach is not able to update the deontic part of the context (*i.e.*, obligations and permissions), in fact Governatori states that an explicit differentiation between norms, obligations and permissions has to be made. 4) Governatori's approach does not provide support for *constitutive rules* (*i.e.*, counts-as).

Aucher's proposal [5] is similar to ours, in the sense both are event-based, it does not make distinctions between deontic statements and norms and has full support for *constitutive rules*. However, neither context expansion or contraction provides support for *ex tunc* operations. In fact, only one expansion and one contraction operations are introduced on his approach; both operations seem to (implicitly) be of *ex nunc* type. Means for ensuring that the normative context is consistent (after the expansion/contraction operation) are included on Aucher's approach, whereas we have not taken care of such issues.

Campos' approach [6] raises from the need of turning *Electronic Institutions* (EI from now on) into *Situated Electronic Institutions* (SEI from now on). *EI* are static and self-contained, agent actions are filtered so norm violations will never occur. *SEI* control over external agents is not tight, therefore violations can occur, and *SERI* can adapt themselves to changes in the dynamic existing social systems. Campos' approach uses a *Bridge* for communicating the *SEI* with the environment. The *Bridge* is similar to the *event-bus* we use on our approach, but contains an API tightly coupled with the domain the environment refers to, while our *event-bus* is domain independent. Besides this, Campos' approach does not support *constitutive rules*. However, Campos' approach allows for *SEI* to automatically adapt the normative context in order to perform better in new environments. Re-configuration is achieved via *transition functions* (TF) that define some basic updates on the normative context of the *SEI*. For instance, if violating the norm *N* implies the payment of a fine, and the system detects the number of violations of *N* is higher than the expected value, a *TF* can increase the value of the fine. Thus, *TF* define very simple modifications to the normative context, without support for norm promulgation, abrogation and annulment.

Tinnemeier et al. [14] propose a framework based on the syntax and the operational semantics of generic programming constructs for norm modification. Using their framework, programmers can modify norms and norm instances via rule-based constructs. On the framework proposed by Tinnemeier et al. norms take the form of conditional obligations and prohibitions. The rule-based constructs for changing norms come in two flavours, norm instance change rules and norm scheme change rules (ic-rules and sc-rules respectively). Ic-rules and sc-rules are of the form $\beta \Rightarrow [ni_0, ..., ni_n][ni'_0, ..., ni'_n]$ with the intuitive reading that under circumstances β the set of norm instances or norm schemes $[ni_0, ..., ni_n]$ are to be removed and the set of norm instances or norm schemes $[ni'_0, ..., ni'_n]$ are to be added. Regarding sc-rules, in some cases it is desirable that the instantiated norm instances remain unaffected, whereas in other cases the associated

norm instances should be changed accordingly. For covering both cases Tinnemeier et al. defines *instance-preserving* sc-rules that leave the instantiated norm-instances unaltered and *instance-revising* sc-rules that revise the associated norm-instances accordingly. Tinnemeier et al.'s approach is similar to ours in the sense it provides the syntax and operational semantics for performing norm-change at run-time. However, Tinnemeier et al.'s approach subjects norm change to a pre-existing condition (*i.e.* β) and does not support retroactive norm promulgation. What's more, in Tinnemeier et al.'s approach norm-change is restricted by a set of norm change rules specified by the normative framework, whereas in our approach it is open.

6 Conclusions and future work

In this paper we have introduced a formal method for monitoring electronic normative environments able to evolve through time as regulations change to adapt to new situations and behaviours. We have started by introducing the four operations supported for updating normative contexts. The operations mix the expressivity of two previous approaches [5] [10] effectively supporting both prospective and retroactive context expansion and contraction. Then we have proposed a formal extension of the base normative monitoring framework that will allow it to support normative context modifications at run-time. That is, the normative environment can be expanded and contracted without having to stop the monitoring process. Furthermore, all expansions and contractions performed leave the normative context in a consistent state (*e.g.*, if a norm was violated, it will remain violated, unless an *Ex Tunc* contraction of the norm has been performed). The extension is outlined by providing algorithms for supporting the four operations for updating normative contexts.

The proposed framework has room for improvement. The interaction between the support for runtime change of constitutive rules that is already available [1] and the support we present in this paper has to be analysed. We also have to analyse the interaction between the extension proposed in this paper and another extension for scaling the framework via distributed monitors [8] that has been developed previously. We also plan using the formalisation presented in this paper to extend the available prototype implementing support for runtime norm change. The implementation will effectively allow for testing the framework on use-case scenarios, making an assessment on the framework's efficiency. We have already explored scenarios based on adaptable AIs for video-games [4]. Finally, we plan to include measures [15] to ensure normative-context modifications result in a consistent and non-redundant model.

Having support for dynamic normative contexts opens one main line of research, adaptive normative contexts, able to insert or remove norms into the system depending on how the system evolves with respect to its overall objectives. In order to go ahead with this line of research we have to extend our framework with methods for detecting when norm change is required from an institutional point of view [2]. Then, we should provide means to institutional agents to perform this norm change [12].

References

1. H. Aldewereld, S. Álvarez-Napagao, F. Dignum, and J. Vázquez-Salceda. Making norms concrete. *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (Toronto, Canada), Nov 2010.
2. H. Aldewereld, F. Dignum, V. Dignum, and L. Penserini. A formal specification for organizational adaptation. In M. P. Gleizes and J. J. Gómez-Sanz, editors, *AOSE*, volume 6038 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 2009.
3. S. Álvarez-Napagao, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Normative monitoring: semantics and implementation. In *Proceedings of the 6th international conference on Coordination, organizations, institutions, and norms in agent systems*, COIN@AAMAS'10, pages 321–336, Berlin, Heidelberg, 2011. Springer-Verlag.
4. S. Álvarez-Napagao, I. Gómez-Sebastià, J. Vázquez-Salceda, and F. Koch. conciens: Organizational awareness in real-time strategy games. In R. Alquézar, A. Moreno, and J. Aguilar-Martin, editors, *CCIA*, volume 210 of *Frontiers in Artificial Intelligence and Applications*, pages 69–78. IOS Press, 2010.
5. G. Aucher, D. Grossi, A. Herzig, and E. Lorini. Dynamic context logic and its application to norm change. 2010.
6. J. Campos, M. López-Sánchez, J. A. Rodríguez-Aguilar, and M. Esteva. Formalising situatedness and adaptation in electronic institutions. *Coordination, Organizations, Institutions and Norms in Agent Systems IV, Lecture Notes in Computer Science, Springer Berlin / Heidelberg*, 5428:126–139, Jan 2009.
7. F. Dignum, J. Broersen, V. Dignum, and J.-J. C. Meyer. Meeting the deadline: Why, when and how. In *FAABS*, pages 30–40, 2004.
8. I. Gómez-Sebastià, S. Álvarez-Napagao, and J. Vázquez-Salceda. A distributed norm compliance model. In C. Fernández, H. Geffner, and F. Manyà, editors, *CCIA*, volume 232 of *Frontiers in Artificial Intelligence and Applications*, pages 110–119. IOS Press, 2011.
9. G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment part i: Revision of defeasible theories. In *DEON*, pages 3–18, 2008.
10. G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment. part ii: Temporalised defeasible logic. In *NORMAS*, pages 112–127, 2008.
11. D. Grossi. Designing invisible handcuffs : Formal investigations in institutions and organizations for multi-agent systems. 2007.
12. N. Oren, M. Luck, and S. Miles. A model of normative power. In *AAMAS*, pages 815–822, 2010.
13. N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. In J. Hbner, E. Matson, O. Boissier, and V. Dignum, editors, *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, Lecture Notes in Computer Science, pages 156–171. Springer Berlin / Heidelberg, 2009.
14. N. A. M. Tinnemeier, M. Dastani, and J.-J. C. Meyer. Programming norm change. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 957–964. IFAAMAS, 2010.
15. W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, 2009.