Norm Monitoring Through Observation Sharing

Bas Testerink, Mehdi Dastani, John-Jules Meyer

Utrecht University, The Netherlands, email:
{B.J.G.Testerink,M.M.Dastani,J.J.C.Meyer}@uu.nl

Abstract. The behavior of agents in multi-agent systems is hard to control without restricting the agents' autonomy. Normative organizations have been deployed successfully to deal with this. A core requirement of a normative organization is that norm violations can be observed and dealt with. Due to the distributed nature of multi-agent systems it is often fitting that this is done decentrally. In this paper we discuss a decentralized method to monitor whether norms are violated in a multi-agent system. The setting we consider is one where the organization depends on a network of monitors that share observations among each other.

1 Introduction

The behavior of multi-agent systems (MAS) is hard to predict. This makes it challenging to guarantee that agents behave according to preset guidelines. Norms are a popular candidate to deal with this [5]. The enforcement of norms requires that violations are detected, and that sanctions are applied to compensate for those violations. We will focus on the detection of norm violations. Because a MAS is highly distributed by nature, it is often fitting to also deploy a decentralized monitor to detect norm violations. Advantages of using decentralized monitoring include better scalability, graceful degradation in the face of system failures and parallel processing of sense data to increase the system's performance.

Formal methods for decentralized monitoring help to better analyze and design decentralized monitors. There are different models in related work (section 2) which are proposed under different assumptions. Our aim in this paper is to provide complementary work to analyze generic scenarios with assumptions that are not covered in other works. The basis of our method is a network of monitors of which each has its own local view on the environment. Each monitor is assigned a set of norms which concern the MAS that is being monitored. Because of the local view of monitors, it might be necessary that they communicate in order to detect violations. In our method this is done through the propagation of observations. Propagation causes delays in violation detection. The proposed method is aimed at detecting a violation in the number of computation steps that it maximally takes for an observation to be propagated across the network. We shall specify a formal framework for analyzing decentralized monitors that are based on our method.

As an example scenario we use a smarts roads use-case. We utilized the same use-case in earlier work to highlight different concepts in distributed organizations [10]. In smart roads applications highways are enriched with intelligent monitoring and control to increase safety and throughput. We use norms as a specification of the traffic laws and policies.

In our scenario, a highway segment has a special priority lane for situations where the traffic density is high (e.g. traffic jams due to accidents or rush hours). This is a lane to increase the throughput of high priority vehicles such as public transport or cars with special permits like taxis. One norm concerning the use of this priority lane is as follows: if an agent is on the priority lane, then it is obliged to get off it or have a permit, before the traffic density becomes high.

We use three different kinds of monitors in our scenario. Lane monitors observe whether a specific car is on a priority lane. Traffic density monitors observe whether the traffic density is high. And a permit monitor can observe whether a specific car has a permit to use the priority lane. The traffic density monitors can communicate with the lane monitors and the permit monitor. The norm violation in this scenario cannot be locally observed by any of the monitors. Data has to be transferred between monitors in order to detect the violation.

In section 2 we discuss related research on (decentralized) monitoring. In section 3 we explain the formal setting of our method. In particular we define the notions of norms, monitors and violation detection for (decentralized) monitors. We discuss the example scenario in section 4. In section 5 we provide the algorithms for how norm violations can be detected distributively by a set of monitors that observe the environment properties with delay.

2 Related Approaches

In earlier work [10] we discussed different aspects of distributed organizations. In this work, we focus on the monitoring aspect of distributed organizations and provide a formal framework to analyze distributed monitoring systems. Our approach is conceptually related to multi-institutions as proposed in [7]. Their proposed language InstAL can be used to specify systems where agents are subjected to the norms of multiple institutions at once. This approach mainly focuses on the analysis of multi-institute systems, not on the runtime monitoring of them.

Linear temporal logic (LTL) and computational tree logic (CTL) are commonly used to describe whether possible system traces are violating a norm (cf. [1]). The traditional approach is to define whether the norm is violated or not given an infinite trace description of a system. This works well for determining a priori whether a system behaves correctly with respect to the norms. However, many practical applications, such as monitoring at runtime, should deal with finite traces. For an overview of the usage of LTL to monitor system constraints see [4].

There are two main flavors of monitors. One option is to create a finite state machine (FSM) given a norm. As a system evolves over time, observed events are processed in the monitor. If the monitor reaches a forbidden state, then it

detects a violation. Such an approach can be found in for instance the proposed monitors for RV-LTL [4] and in [12]. In these approaches the monitors themselves are not distributed. The advantage of FSMs is that one can label states with additional information. This is done in [12], and creates a system that allows the user to determine not only whether a rule is violated, but also exactly which transition caused this. However, a disadvantage with the FSM approach is that the creation of the FSM is exponential given a set of LTL formulas and the amount of different possible events in the system. The creation of FSMs is not our the first choice as we aim for large distributed systems with sets of norms and many different events.

The other flavor for monitoring LTL properties is the use of progression functions. The progression function proposed in [2], originally deployed in a goal planning system, has been an inspiration for related works on monitoring such as $hyMITL^{\pm}$ [8], [3] and [9]. The core idea is to produce for a given state in a trace the formula which the next future state of the trace has to satisfy. The work in [8] and [9] does not deal with decentralized monitoring. The algorithm in [3] concerns decentralized LTL monitoring, but differs on some important aspects w.r.t. our work. For example, [3] assumes that all monitors can pairwise communicate while we assume that monitors can communicate based on predefined communication channels. Another difference is that in their algorithm all monitors are monitoring one and the same global system norm. In our framework each monitor will be assigned its own set of norms to monitor. The monitors have to cooperate to detect the violations of their norms.

Our method neither builds a FSM, nor does it use a progression function, but it resembles most the progression method. The reason why progression is not needed, is that our method, in contrast to [3], monitors share observations among each other, instead of formulas representing a norm. Our method could be enhanced with a progression function as a practical improvement. Ideas on this are provided in section 6.

3 Formal Framework

We will explain decentralized monitoring in various steps. Our framework builds on LTL which we shall discuss first. Second we define norms and norm violations. Third we explain the notion of a monitor and norm violation detection. Fourth we discuss different types of monitors, that depend on their violation detection capability. Finally we discuss decentralized monitors.

3.1 Preliminaries

To determine whether a norm violation has occurred in a system we start with standard LTL with a weak until operator. We assume there is a finite set of atoms A and environment states S. An infinite trace $\sigma = s_0 s_1 \ldots$ is a trace of environment states s.t. $s_i \in S$. The valuation function $V : S \to 2^A$ returns, given a state s, the set of atoms that hold in s. We use Σ^ω to notate the set of all infinite traces, and Σ^* for the set of all finite traces. The syntax we use is as follows:

$\varphi = p|\neg\varphi|\varphi \vee \psi|\bigcirc \varphi|\varphi \mathcal{U} \psi$ where $p \in A$.

Given a trace $\sigma = s_0 s_1 \cdots \in \Sigma^\omega$, and an environment state s_i in σ, entailment $\vDash_{ltl}$ is as follows:

$$
\begin{array}{ll}
\sigma, s_i \vDash_{ltl} p & \Leftrightarrow p \in V(s_i) \\
\sigma, s_i \vDash_{ltl} \neg\varphi & \Leftrightarrow \sigma, s_i \nvDash_{ltl} \varphi \\
\sigma, s_i \vDash_{ltl} \varphi \vee \psi & \Leftrightarrow \sigma, s_i \vDash_{ltl} \varphi \text{ or } \sigma, s_i \vDash_{ltl} \psi \\
\sigma, s_i \vDash_{ltl} \bigcirc\varphi & \Leftrightarrow \sigma, s_{i+1} \vDash_{ltl} \varphi \\
\sigma, s_i \vDash_{ltl} \varphi \mathcal{U} \psi & \Leftrightarrow \forall j \in [i, \infty] : \sigma, s_j \vDash_{ltl} \varphi, \text{ or:} \\
 & \quad \exists j \in [i, \infty] : (\sigma, s_j \vDash_{ltl} \psi \text{ and } \forall k \in [i, j-1] : \sigma, s_k \vDash_{ltl} \varphi)
\end{array}
$$

Runtime monitoring concerns finite traces only. For formulas such as $\varphi \mathcal{U} \psi$ it might be the case that φ is always true in a finite trace and ψ is always false. In such cases it is not possible to evaluate the formula as it is unclear whether the formula is true in the future of the trace. Therefore we will use common finite trace semantics for LTL that evaluates formulas as being true, false or inconclusive.

We notate a finite trace of length $k+1$ as σ^k and $\sigma^k \cdot \Sigma^\omega$ is the set of infinite traces s.t. σ^k is a prefix. Given a finite trace $\sigma^k \in \Sigma^*$, and a state s_i, $i \in [0, k]$, finite trace LTL entailment $\vDash_f$ is defined as follows:

$$
\sigma^k, s_i \vDash_f \varphi = \begin{cases} \top \text{ if } \forall \sigma \in \sigma^k \cdot \Sigma^\omega : \sigma, s_i \vDash_{ltl} \varphi \\ \bot \text{ if } \forall \sigma \in \sigma^k \cdot \Sigma^\omega : \sigma, s_i \nvDash_{ltl} \varphi \\ ? \quad otherwise \end{cases}
$$

For a trace σ^k if φ is true/false in state s_i, $i \in [0, k]$, then it is true/false in s_i for any finite extension of σ^k.

Proposition 1. *Let $\sigma^k \in \Sigma^*$ be a prefix of $\sigma^j \in \Sigma^*$, φ a formula, and $i \in [0, k]$:*
If $\sigma^k, s_i \vDash_f \varphi = v$ then $\sigma^j, s_i \vDash_f \varphi = v$, for $v \in \{\bot, \top\}$.

Proof sketch: This follows from the definition of $\vDash_f$, as future states cannot change a conclusive valuation.

3.2 Norms

Norms can have different forms. For instance some norms are state-based whereas others are event-based. Also, norms can have temporal aspects in the form of conditions and deadlines. In this paper, we use conditional norms with deadlines that are state-based [11]. A norm consists of a condition, obligation and deadline. Each of those norm parts is a system state for which we use propositional logic. Note that if a deadline is the passing of time, then the state of a clock can be seen as part of the system state.

Definition 1. ***Norm.*** *A norm is specified by a tuple of propositional formulas $\langle \varphi_c, \varphi_o, \varphi_d \rangle$ s.t. φ_c, φ_o and φ_d are the norm's condition, obligation and deadline, respectively.*

We use $\mathcal{N}$ for the set of possible norms and A^N for the set of all atoms that occur in norms from a set N. Conditional norms with deadlines are related to temporal logic (cf. [6]). We shall use temporal logic to define the violation of a norm. The intuitive reading of a norm is that if the condition holds, that then the obligation must be fulfilled before the deadline occurs. Therefore, a norm is violated in a finite trace if after its condition, the deadline holds earlier than the obligation.[1]

Definition 2. ***Norm violation.*** *Let σ^k be a finite trace, a norm $\langle \varphi_c, \varphi_o, \varphi_d \rangle$ is violated in σ^k iff there exists an $i \in [0, k]$ s.t.:*

$$\sigma^k, s_i \vDash_f \varphi_c \wedge \neg(\neg\varphi_d \mathcal{U} \varphi_o) = \top$$

If a norm is violated in a trace σ^k, then it is violated in any finite extension of σ^k.

Proposition 2. *If a norm $\langle \varphi_c, \varphi_o, \varphi_d \rangle$ is violated in a trace σ^k, then it is violated in all $\sigma^j \in \Sigma^*$ s.t. σ^k is a prefix of σ^j.*

Proof sketch: The violation is a conclusive formula evaluation on σ^k and a state s_i. Following proposition 1, the evaluation is the same for σ^j and s_i.

3.3 Monitors

A monitor has a set of norms that it monitors. Monitors can observe some atoms with a delay and therefore have a delay function. This function returns, given an atom, the time that it takes to observe whether that atom holds. The delay of an observation is measured in abstract time units that correspond with the time between two states in a trace. For the following definition, recall that A is the global set of atoms.

Definition 3. ***Monitor specification.*** *A monitor is specified by a tuple $\langle N, \delta \rangle$, where $N \subseteq \mathcal{N}$ is a set of norms and $\delta : A \to \mathbb{N}^\infty$ is a delay function.*

As we shall see in section 3.5 the delays in this paper are caused by communication. However delays can also originate from for instance sensors that make observations some time after they happen. Extending the framework with additional causes of delay (e.g. delayed sensors) will impact some of the results regarding the delays of when norm violations are detected. The main limitation for extending the model's delay specification is that the delay must be measured in equal time units.

We use $\mathcal{M}$ to notate the set of possible monitor specifications. A monitor is a function that given a trace returns a set of traces that it cannot distinguish according to its specification. Consider a scenario with a set of atoms $A = \{c, o, d\}$ and a monitor m that is specified by $\langle \{\langle c, o, d \rangle\}, \delta \rangle$, where $\delta(c) = \delta(d) = 0$, and $\delta(o) = 2$. Furthermore $S = \{s_a, s_b, s_c\}$, s.t. $V(s_a) = \{c, o\}, V(s_b) = \{c\}$ and $V(s_c) = \{d\}$.

[1] Note that given a finite trace σ^k and a norm $n = \langle \varphi_c, \varphi_o, \varphi_d \rangle$, σ^k is compliant with n iff for each $i \in [0, k] : \sigma^k, s_i \vDash_f \varphi_c \to (\neg\varphi_d \mathcal{U} \varphi_o) \neq \bot$

Because $\delta(o) = 2$, m's view on the environment trace at moment k does not inform m whether o is true at moment k and $k-1$. Consider the environment trace $\sigma = s_a s_c s_c$. At the initial time step m does not observe o to be true or false. Instead, m observed that c is true and that d is not true. So all possible initial states are s_a and s_b. At the second time step m determines that the second state can only be s_c, because it observed d and not c. At the third time step m observes o and d. Because of the delay, this means that o was true in the initial state. Therefore m observes that indeed in the initial time step the state was s_a. Hence the possible environment traces per time step are:

1. Possible traces given $\sigma^0 = s_a$ are s_a or s_b.
2. Possible traces given $\sigma^1 = s_a s_c$ are $s_a s_c$ or $s_b s_c$.
3. The possible trace given $\sigma^2 = s_a s_c s_c$ is $s_a s_c s_c$.

We define in definition 4 when a monitor m cannot distinguish between two traces w.r.t. the atoms that occur in norms that m monitors. These are the traces where for every state the valuation of atoms for which the delay has passed coincides. Because those are the atoms for which m has certainty.

Definition 4. ***Trace indistinguishability.*** *A monitor m specified by $\langle N, \delta \rangle$ cannot distinguish between $s_0 \dots s_k$ and $s'_0 \dots s'_k$, notated as $s_0 \dots s_k \sim_m s'_0 \dots s'_k$, iff for all $i \in [0,k], p \in A^N, \delta(p) \leq k - i$: $p \in V(s_i)$ iff $p \in V(s'_i)$.*

Given a monitor m and a trace σ^k, $m(\sigma^k)$ is the set of indistinguishable/equivalent traces given m's view on the environment.

Definition 5. ***Monitor.*** *Let $m : \Sigma^* \to 2^{\Sigma^*}$ be a monitor specified by $\langle N, \delta \rangle$ and σ^k be a trace. The set of equivalent traces for m given σ^k is notated by $m(\sigma^k) = \{\sigma \in \Sigma^* | \sigma^k \sim_m \sigma\}$.*

A monitor m specified by $\langle N, \delta \rangle$ has uncertainty of atoms in A^N as far in the past as the delay of the atoms. If the delay of an atom has passed for a state s'_i in a trace $\sigma \in m(\sigma^k)$, then its valuation equals that of the state s_i in σ^k.

Proposition 3. *Let m be a monitor specified by $\langle N, \delta \rangle$ and σ^k be a trace $s_0 \dots s_k$. For all $s'_0 \dots s'_k \in m(\sigma^k), p \in A^N, i \in [0, k - \delta(p)]$:$p \in V(s_i)$ iff $p \in V(s'_i)$.*

Proof sketch: This follows from definitions 4 and 5.

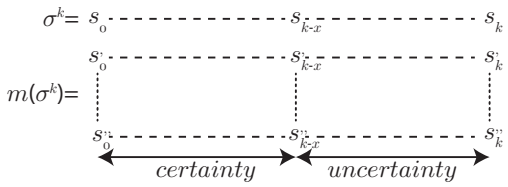

Fig. 1. A trace σ^k and a monitor's indistinguishable traces $m(\sigma^k)$, where all traces in $m(\sigma^k)$ share a prefix of length $k - x$ in which there is certainty for a set of atoms.

For a monitor m specified by $\langle N, \delta \rangle$ let x be the maximum delay for atoms in A^N. If a trace σ^k is longer than x, then all possible traces in $m(\sigma^k)$ have a prefix of length $k - x$ where formulas concerning atoms in A^N all evaluate equally in those prefixes. This is also illustrated in Figure 1.

Proposition 4. *Let m be a monitor specified by $\langle N, \delta \rangle$, σ^k be a trace, φ be a formula of which all atoms occur in A^N, $q = \arg\max_{p \in A^N} \delta(p)$ and $x = k - \delta(q)$. For all $\sigma, \sigma' \in m(\sigma^k)$, their prefixes $\sigma^x = s_0 \dots s_x$ and $\sigma'^x = s_0 \dots s'_x$, $i \in [0, x]$:*

$$\sigma^x, s_i \vDash_f \varphi \Leftrightarrow \sigma'^x, s'_i \vDash_f \varphi.$$

Proof sketch: Because all atoms evaluate equally up until $k - \delta(q)$ (prop. 3), any verdict of a formula in the prefix up until $k - \delta(q)$ with those atoms is the same.

Continuing with our earlier scenario, the monitor can reason about the violation of the norm as follows:

1. If $\sigma^0 = s_a$ then the violation is inconclusive.
   If $\sigma^0 = s_b$ then the norm is satisfied.
   So there is no violation of the norm.
2. If $\sigma^1 = s_a s_c$ then the norm is violated.
   If $\sigma^1 = s_b s_c$ then the norm is satisfied.
   So there is no certain violation of the norm.
3. $\sigma^2 = s_a s_c s_c$ so the norm is satisfied so far.

Any trace is per definition indistinguishable from itself, so $\sigma^k \in m(\sigma^k)$. A violation is detected by a monitor if a norm is violated in all possible environment traces, because then the norm is also violated in σ^k.

Definition 6. ***Violation detection.*** *Monitor m specified by $\langle N, \delta \rangle$ detects a violation of $n \in N$ in a trace σ^k at state s_k iff n is violated in all $\sigma' \in m(\sigma^k)$.*

A property of violation detection is that there is a maximum delay x for each monitor s.t. if a norm is violated, and a monitor m can detect that violation, that then this violation is always detected within x steps. This delay x depends on the delay function of m.

Proposition 5. *Let m be a monitor specified by $\langle N, \delta \rangle$, $q = \arg\max_{p \in A^N} \delta(p)$ s.t. $\delta(p) \neq \infty$, and σ^k be a trace s.t. a norm $n \in N$ is violated in σ^k. If m can detect the violation, then it does so within $\delta(q)$ steps.*

Proof sketch: Because the violation will be detected, it means that the monitor gains enough information of the environment trace up until σ^k to determine the violation. After $\delta(q)$ time steps, no more certainty can be gained for atoms (prop. 3 & 4). Hence the violation must be detected within $\delta(q)$ steps.

3.4 Monitor Types

We distinguish between different types of monitors. These types indicate how well a monitor can detect norm violations. A perfect monitor detects violations whenever they occur.

Definition 7. ***Perfect monitor.*** *A monitor m specified by $\langle N, \delta \rangle$ is perfect iff for all $\sigma^k \in \Sigma^*$ if there is a violation of a norm $n \in N$ in σ^k, then m detects the violation in σ^k at s_k.*

A delayed monitor detects violations with some delay. This means that if a norm is violated, then the monitor may not immediately detect this. But it is guaranteed that the monitor will detect the violation in the future.

Definition 8. ***Delayed monitor.*** *A monitor m specified by $\langle N, \delta \rangle$ is delayed iff for all $\sigma^k \in \Sigma^*$ if there is a violation of a norm $n \in N$ in σ^k, then for all $\sigma \in \sigma^k \cdot \Sigma^\omega$, there is a prefix σ^j of σ, $j \geq k$, s.t. m detects the violation of n in σ^j at s_j.*

There is a connection between the delay function of a monitor and its type. If the delays of all atoms which are relevant for a monitor's norms are not infinite, then m is perfect and/or delayed. Also, a perfect monitor is a special case of a delayed monitor, as a perfect monitor is a delayed monitor where the delay of detecting violations is 0.

Proposition 6. *Let m be a monitor specified by $\langle N, \delta \rangle$. If $\forall p \in A^N : \delta(p) = 0$ then m is perfect, if $\forall p \in A^N : 0 \leq \delta(p) < \infty$ then m is delayed.*

Proof sketch: Following propositions 4 and 5: if a norm is violated in σ^k and $\forall p \in A^N : \delta(p) < \infty$, then the valuation of all atoms in A^N becomes equal in prefixes of indistinguishable traces for m. If $\forall p \in A^N : \delta(p) = 0$ then the maximum delay for detecting a violation is 0, thus a violation would immediately be detected, hence m would be perfect. If $\forall p \in A^N : 0 \leq \delta(p) < \infty$ then the maximum delay x is finite and m will always detect the violation after x steps , hence m would be delayed.

A flawed monitor may never detect some violations.

Definition 9. ***Flawed monitor.*** *A monitor m specified by $\langle N, \delta \rangle$ is flawed iff m is not delayed.*

For a flawed monitor there is a trace σ^k where a norm is violated and there is an infinite trace $\sigma \in \sigma^k \cdot \Sigma^\omega$ s.t. there is no prefix of σ longer than k in which the monitor detects the violation. This has as a consequence that the delay function cannot determine whether a monitor m specified by $\langle N, \delta \rangle$ is flawed. To illustrate this, consider the extreme case where for an environment there is no trace $\sigma^k \in \Sigma^*$ s.t. a norm $n \in N$ is violated in σ^k. If no norm can be violated, then there is no violation that m can miss, and hence m cannot be flawed. Even if for each atom $p \in A^N$ the delay is infinite ($\delta(p) = \infty$). This extreme case is unlikely to occur in practice, because if the norms in N cannot be violated, then there is also no use for a monitor to monitor them.

3.5 Decentralized Monitors

A decentralized monitor is a network of monitors that can propagate observations among themselves. The setting we assume is that like in [3] the network ticks at synchronous intervals that coincide with the states in the system trace. Between each tick messages can be send between monitors that are directly connected to each other. We do not assume full connectedness, i.e. the network can have any topology. Also, as seen in definition 3, each monitor has its own set of norms.

Delays arise because it takes time to propagate observations. The delay functions of monitors therefore depend on the topology of the network. We assume that if a monitor m observes an atom p to hold in the environment, that it then sends this information to each connected monitor m' in the network. If m receives an observation, then it also sends this information to each connected monitor m'. The propagation delay between two directly connected monitors is one. Each monitor has a visibility of the set of atoms A which it can observe directly in the environment, without delay. The delay of an atom p is therefore either zero if p is visible for m, or the length of the shortest path to a monitor for which p is visible, or infinite if there is no path to a monitor m' s.t. p is visible for m'.

Definition 10. ***Decentralized monitor.*** *A decentralized monitor is specified by a tuple $\langle G, \Pi \rangle$, where $G = (M, C)$ is a graph of monitors $M \subseteq \mathcal{M}$ with communication lines $C \subseteq M \times M$, and $\Pi : M \to 2^A$ is a visibility function. For all m specified by $\langle N, \delta \rangle \in M, p \in A$:*

$$\delta(p) = \begin{cases} 0 & \text{if } p \in \Pi(m) \\ x & \text{if } p \notin \Pi(m) \text{ and } x \text{ is the length of the} \\ & \text{shortest path between } m \text{ and some} \\ & m' \in M \text{ given } G \text{ s.t. } p \in \Pi(m'). \\ \infty & \text{otherwise} \end{cases}$$

A decentralized monitor detects violations of norms if any of its monitors in the network detects a violation.

Definition 11. ***Decentralized violation detection.*** *Let $\langle (M, C), \Pi \rangle$ be the specification of a decentralized monitor. The violation of a norm $n \in \mathcal{N}$ in a trace σ^k is decentrally detected in σ^k at s_k iff there is a monitor $m \in M$ specified by $\langle N, \delta \rangle$ s.t. $n \in N$ and m detects a violation of n given σ^k at s_k.*

For a decentralized monitor D specified by $\langle (M, C), \Pi \rangle$, C and Π determine whether a monitor $m \in M$ is perfect, delayed or possibly flawed. For instance if for a monitor m specified by $\langle N, \delta \rangle \in M$ it holds that $A^N \subseteq \Pi(m)$, then $\delta(p) = 0$ for any $p \in A^N$, which by proposition 6 makes the monitor perfect. Also if $C = \emptyset$, i.e. if monitors cannot communicate, then there are no shortest paths among monitors. Therefore $\delta(p)$ is either 0 or ∞ for any $p \in A^N$. Thus all monitors would be either perfect or flawed.

If all monitors in the network are perfect, then any decentrally detected norm violation will occur instantaneously as the violation occurs. But the detection can

also be instantaneous if for instance all monitors are flawed. To illustrate this, consider two monitors m and m' that are specified by $\langle\{n\}, \delta\rangle$ and $\langle\{n\}, \delta'\rangle$, where $n = \langle a \vee b, o, d\rangle$. Let the environment states S be the set $\{s_a, s_b\}$ s.t. $V(s_a) = \{a, d\}$ and $V(s_b) = \{b, d\}$. In both s_a or s_b the norm will be violated, because in those states the condition and the deadline occur at the same time, but not the obligation. If $\delta(a) = \delta(o) = \delta(d) = 0$ and $\delta(b) = \infty$, then m is flawed, because in the event that the trace always is a repetition of s_b, then m will never detect the violation. Equally for m' if $\delta'(b) = \delta'(o) = \delta'(d) = 0$ and $\delta'(a) = \infty$, then m' is flawed because of the possible repetition of s_a. However, in whatever trace occurs, one of the flawed monitors detects the violation of the norm instantaneously.

If a violation of a norm n is decentrally detected, then the maximum time that it took to detect the violation depends on the maximum time of a local monitor m specified by $\langle N, \delta\rangle$ with $n \in N$ to detect the violation. Following proposition 5 this maximum is the maximum delay of atoms. In a decentralized monitor, the delay of an atom is the time it takes to propagate an observation, which corresponds to the shortest distance of one monitor to another monitor. Given the set of monitor specifications M, this maximum delay is thus $|M|$.

Proposition 7. *Let $\langle(M, C), \Pi\rangle$ be a decentralized monitor specification, $n \in \mathcal{N}$, and σ^k be a trace. If n is violated in σ^k and it is decentrally detected, then the detection was within $|M|$ steps.*

Proof sketch: For all monitors the maximum delay for an atom aside from ∞ is $|M|$, which is the case if the network has the form of a linear list. Therefore any local monitor that detects the violation does this within $|M|$ steps (proposition 5).

4 Example Formalized

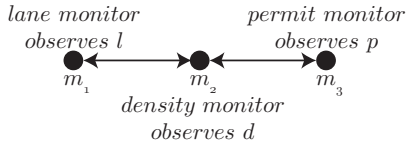

Fig. 2. Example decentralized monitor.

We continue with the earlier mentioned smart roads scenario. To simplify the following we only consider one lane, density and permit monitor as depicted in Figure 2. We also assume the system monitors a single vehicle. The atoms with which we model our example are as follows:

- l stands for *the vehicle is on a priority lane.*
- d stands for *traffic density is high.*
- p stands for *the vehicle has a permit.*

For each possible combination of l, d and p there is a state in the set of possible environment states S s.t. that combination holds. The complete setup is as follows:

- $A = \{l, d, p\}$, and $\forall A' \subseteq A, \exists s \in S : V(s) = A'$.
- $D = \langle (M, C), \Pi \} \rangle$, where:
  $M = \{m_1, m_2, m_3\}$,
  $m_2 = \langle \{n\}, \delta \rangle$,
  $n = \langle l, p \vee \neg l, d \rangle$,
  $C = \{(m_1, m_2), (m_2, m_3), (m_2, m_1), (m_3, m_2)\}$,
  $\Pi(m_1) = \{l\}$, $\Pi(m_2) = \{d\}$, $\Pi(m_3) = \{p\}$.
- $\delta_{m_1}(l) = 0, \delta_{m_1}(p) = 2, \delta_{m_1}(d) = 1$.
- $\delta_{m_2}(l) = 1, \delta_{m_2}(p) = 1, \delta_{m_2}(d) = 0$.
- $\delta_{m_3}(l) = 2, \delta_{m_3}(p) = 0, \delta_{m_3}(d) = 1$.
- $\sigma^2 = s_0 s_1 s_2$, where:
  - $V(s_0) = \{l\}$
  - $V(s_1) = \{d, l\}$
  - $V(s_2) = \{p, d, l\}$

Monitor m_2 is a delayed monitor. The environment trace indicates that the car drove on the priority lane and obtained a permit too late when the traffic density became high. The norm is violated at the state s_1, because the car is on the priority lane during high traffic density, without having a permit. We first determine m_2's observations about the environment trace per time step as depicted below.

1. $d \notin V(s_0)$
2. $d \notin V(s_0)$, $l \notin V(s_0)$, $p \notin V(s_0)$,$d \in V(s_1)$.
3. $d \notin V(s_0)$, $l \notin V(s_0)$, $p \notin V(s_0)$,$d \in V(s_1)$, $l \in V(s_1)$, $p \notin V(s_1)$,$d \in V(s_2)$.

For any of the traces $\sigma = s'_0 s'_1 s'_2 \in m_2(\sigma^2)$ the observations of time step 3 must hold. This means that for all those traces it holds that:
$\sigma, s'_1 \vDash_f l = \top$, because $l \in V(s_1)$
$\sigma, s'_1 \vDash_f d = \top$, because $d \in V(s_1)$
$\sigma, s'_1 \vDash_f \neg(p \vee \neg l) = \top$, because $l \in V(s_1)$ and $p \notin V(s_1)$.
Therefore: for all $\sigma = s'_0 s'_1 s'_2 \in m_2(\sigma^2)$ there is a i (i.e. $i = 1$) s.t.
$\sigma, s'_i \vDash_f l \wedge \neg(\neg d \mathcal{U}(p \vee \neg l)) = \top$
So m_2 detects the violation given σ^2 at s_2. Therefore the violation is also decentrally detected. The norm was already violated when the trace was $\sigma^1 = s_0 s_1$, but at that moment m_2 did not have certainty about whether l or p were in s_1 or not. Had the norm been assigned to m_1, then it would have taken another time step for the decentralized monitor to detect the violation.

5 Translation to Local Monitors

A monitor that is part of a decentralized monitor may have no perfect view on the environment. We shall show it can construct a set of possible environment traces, with which it can check whether a norm is violated. For the shown algorithms we assume that monitors execute in a synchronous manner and execute the algorithms instantaneously. For space reasons, we do not provide details on communication protocols. A monitor's input is a set of observations $O \subseteq A$,

which includes communicated observations. Observations for an atom p from monitors which are not the closest connected monitor that can observe p are not part of O.

Recall that the environment is always in a state from S. A monitor m maintains a set of possible environment states per time step which is updated given the observations of the next time step. In particular m maintains a vector of sets of environment states $\boldsymbol{S} = S_0, \ldots, S_k$, s.t. all states $s_i \in S_i$ are the possible environment states at time i. Given a set of observations $O \subseteq A$, this vector is expanded with S and then all impossible states given the observations are filtered out. For every atom $p \in A$ if p is or is not in O then the monitor knows whether the atom was or was not true in the environment state at moment $k - \delta(p)$. If for instance $p \in O$ then all $s_i \in S_i$, $k - \delta(p) = i$, where $p \notin s_i$ have to be removed from S_i. This algorithm is given below.

Algorithm 5.1: ProcessObservations(O)

global S
local $\delta, \boldsymbol{S} = S_0, S_1, \ldots, S_{k-1}$
$S_k \leftarrow S$
$\boldsymbol{S} \leftarrow S_0, S_1, \ldots S_{k-1}, S_k$
for each $S_i, 0 \leq i \leq k$
do
- **for each** $s_i \in S_i$
  - **do**
    - **if** $\exists p \in O \wedge p \notin V(s_i) \wedge k - \delta(p) = i$ **then** $S_i \leftarrow S_i \setminus \{s_i\}$
    - **if** $\exists p \notin O \wedge p \in V(s_i) \wedge k - \delta(p) = i$ **then** $S_i \leftarrow S_i \setminus \{s_i\}$

The following algorithm is for detecting norm violations. For each possible environment trace up until now a norm is checked based on $\models_f$ entailment. If there is a time step where in all possible traces the norm is violated, then it must be true that in the actual environment trace the norm is violated. If a violation is detected then ViolationProcedure() could be used to for instance execute a sanction.

Algorithm 5.2: CheckViolation($\langle \varphi_c, \varphi_o, \varphi_d \rangle$)

local $\boldsymbol{S} = S_0, \ldots, S_k$
$violAll \leftarrow \top$
for each $\sigma^k = s_0 \ldots s_k, s.t. s_i \in S_i$
do
- $violatedInTrace \leftarrow \bot$
- **for each** $j \in [0, k]$
  - **do**
    - **if** $(\sigma^k, s_j \models_f \varphi_c \wedge \neg(\neg\varphi_d \mathcal{U} \varphi_o) = \top)$ **then** $violatedInTrace \leftarrow \top$
- $violAll \leftarrow violAll \wedge violInTrace$

if $violAll$
then ViolationProcedure()

Algorithm 5.1 mirrors for the monitor the construction of the set $m(\sigma^k)$ at moment k. Algorithm 5.2 relates to the definition of violation detection, because the monitor checks whether in all indistinguishable traces up until now the norm is violated at some time point given finite LTL semantics. Note that in theory following proposition 7 a monitor needs to store a maximum the last $|M|$ states of the trace, where $|M|$ is the amount of monitors in the network.

6 Discussion and Future Research

We considered norms with obligations. The counterpart of obligations, prohibitions, can straightforwardly be used as well. Instead of checking for $\varphi_c \wedge \neg(\neg\varphi_d \mathcal{U} \varphi_o)$, one would check for $\varphi_c \wedge \neg(\neg\varphi_f \mathcal{U} \varphi_d)$, where φ_f is a propositional formula denoting a forbidden state, and φ_d is the deadline.

The presented algorithms for local monitors are prototypical and are designed to correspond with the definition of delayed monitoring. Working with the set of possible states is often exponential in the number of atoms. For a better performance, the monitors could use a progression function as it is used in related work. Also the algorithms are made under the assumption that monitors are fully synchronized. In a MAS there is a high level of distribution so this assumption may not always be realizable in practice. However, many critical real life monitoring systems work with synchronization in order to be more predictable, and hence safer. It remains an important topic to see what happens if various assumptions are changed or dropped.

We will further develop our monitoring method by optimizing the algorithms for local monitors. This includes an analysis of which communication needs to take place given a specification of a decentralized monitor. Aside from runtime concerns, we also want to investigate how we can design efficient decentralized monitors. We saw in our example scenario that the delay with which violations are detected depends on which monitors monitor what norms. But a faster detection may require more communication.

The presented decentralized monitoring method is a step in a longer research endeavor where we aim to investigate different aspects of distributed organizations. For the near future we aim to connect this work to decentralized control of a multi-agent system, so that we have a complete picture of how norms can be enforced in distributed organizations.

With this we can continue to develop a practical framework for the design and development of distributed organizations. This in turn can be used to construct complex norm enforcement systems. Example target domains are service oriented architectures and large scale simulations based on autonomic computing and/or multi-agent systems.

7 Conclusion

We have presented a decentralized norm monitoring method. The proposal is to deploy a decentralized monitor that consists of a network of local monitors. Each of the local monitors has its own view on the environment state. Observations

can be propagated among monitors in the network. We provided the formal tools to analyze a decentralized monitor and a prototype algorithm for how a local monitor can detect a norm violation.

Local monitors in our method maintain a set of possible environment traces based on their (possibly delayed) observations. If in all those traces a norm is violated, then it is guaranteed that the norm was also violated in the environment trace.The proposed method guarantees that the violation of a norm is detected within the time that is equal to the maximum propagation delay of observations between two monitors. With this decentralized monitoring method we continue our investigations in distributed organizations.

References

1. T. Agotnes, W. Van Der Hoek, J. Rodriguez-Aguilar, C. Sierra, and M. Wooldridge. On the logic of normative systems. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1181–1186. 2007.
2. F. Bacchus and F. Kabanza. Planning for temporally extended goals. volume 22, pages 5–27. Kluwer Academic Publishers, 1998.
3. A. Bauer and Y. Falcone. Decentralised LTL monitoring. In *FM 2012: Formal Methods*, pages 85–100. Springer Berlin Heidelberg, 2012.
4. A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *J. Log. and Comput.*, 20(3):651–674, June 2010.
5. Guido Boella and Leendert Van Der Torre. Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, 12:71–79, 2006.
6. J. Broersen, F. Dignum, V. Dignum, and J.-J.Ch. Meyer. Designing a deontic logic of deadlines. In A. Lomuscio and D. Nute, editors, *Deontic Logic in Computer Science*, volume 3065 of *LNCS*, pages 43–56. Springer Berlin Heidelberg, 2004.
7. O. Cliffe, M. De Vos, and J. Padget. Specifying and reasoning about multiple institutions. In *AAMAS'06: Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN-2006)*, 2006.
8. Stephen Cranefield. A rule language for modelling and monitoring social expectations in multi-agent systems. In O. Boissier et al., editor, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 246–258. Springer Berlin Heidelberg, 2006.
9. Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *Proceedings of the 16th Annual International Conference on Automated Software Engineering, 2001.(ASE 2001).*, pages 135–143. IEEE, 2001.
10. B. Testerink, M. Dastani, and J-.J.Ch. Meyer. Norms in distributed organizations. In *AAMAS'13: Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN-2013)*, 2013.
11. N. A M Tinnemeier, M.M. Dastani, J-J Ch Meyer, and L. van der Torre. Programming normative artifacts with declarative obligations and prohibitions. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09.*, volume 2, pages 145–152, Sept 2009.
12. M. Westergaard. Better algorithms for analyzing and enacting declarative workflow languages using ltl. In S. Rinderle-Ma et al., editor, *Business Process Management*, volume 6896 of *LNCS*, pages 83–98. Springer Berlin Heidelberg, 2011.