A Distributed Norm Compliance Model

Ignasi Gómez-Sebastià, Sergio Alvarez-Napagao and Javier Vázquez-Salceda [a,1]
[a] *KEMLg, Universitat Politècnica de Catalunya*

Abstract. Norms can be used in the scope of distributed computational systems to provide reliable contexts of interaction between parties where acceptable behaviour is specified in terms of regulations or guidelines. These have been explored in various formalisations, and many are the theoretical frameworks that allow to implement and operationalise them. However, when applying these frameworks to complex, heterogeneous scenarios with multiple agents involved, the performance of norm compliance systems may suffer due to bottlenecks, not only in the number of events received, but also on the number and the complexity of the norms being verified. In this paper we present a formal method to distribute norms through a distributed normative monitoring system, based on production systems for maximum efficiency, and a grounding on Strongly Connected Components.

1. Introduction

With the continuous redefinition and evolution of distributed systems –e.g., Multi-Agent Systems, Grid Computing, Service-Oriented Architectures, Cloud Computing–, there is a growing need for these systems' governance. In [5] governance is defined as the distributed control of complex policies in order to ensure the coherence and the stability of these distributed systems as a whole.

There are several abstractions at different levels of expressivity that try to tackle this problem in real-world scenarios, interpreting events as symbolic facts rather than limiting themselves to pure numerical metrics. One of such abstractions is Normative Systems. Research in Normative Systems focuses on the concepts of norms and normative environments[2] in order to provide normative frameworks to restrict or guide the behaviour of (software) agents. The main idea is that the interactions among a group of such agents are ruled by a set of explicit norms expressed in a computational language representation that agents can interpret. Although some authors only see norms as inflexible restrictions to agent behaviour, others see norms not as a negative, constraining factor but as an aid that guides the agents' choices and reduces the complexity of the environment, making the behaviour of other agents more predictable.

In [2] we proposed a reduction from expressive norms to general production systems to build a norm monitoring mechanism that can be used both by agents to perceive the current normative state of their environment, and for these environments to detect norm violations and enforce sanctions. With such a reduction, an agent can configure, at a practical level, the production system at run-time by adding abstract organisational

[1]c/Jordi Girona 1-3, E08034, Barcelona, Spain e-mail: {igomez, salvarez, jvazquez}@lsi.upc.edu
[2]Usually called *institutions*.

specifications –regulative norms– and sets of counts-as rules –constitutive norms. Therefore, in our approach, the detection of normative states is a passive procedure consisting in monitoring past events and checking them against a set of active norms. This type of reasoning is already covered by the declarative aspect of production systems, so no additional implementation in an imperative language is needed. Using a forward-chaining rule engine, events will automatically trigger the normative state –based on the operational semantics– without requiring a design on how to do it.

An advantage of using general production systems is that the efficiency of the system is bound to the complexity of such systems, which is linear to the number of productions contained in the rules in the worst case and constant in the best case [8]. However, in real-world scenarios this might not even be sufficient, as there may be bottlenecks on both 1) the number of events received, and/or 2) the number of norms –and therefore, the number of rules– loaded in the production system. In this paper we focus on the latter and present a model for distributing the normative context among several monitoring systems at run-time. The objective is to effectively reduce the number of events to be taken into account by each monitor and allow each of them to process a smaller part of the whole normative context. Distributed monitors are linked in order to collectively infer the full normative state of the context, and the model ensures that these links are kept to a minimum amount in order to minimise dependencies.

From a practical perspective, we intend to follow the model of distributed interpretation introduced by Lesser et al. [10]: autonomous local nodes with a separate knowledge base, each one responsible of their own *area-of-interest*, and with a decentralised coordination. If we see normative contexts as interpretation areas of interest, our system will provide a cooperative interpretation of brute events as relevant facts from a governance point of view, and thus will allow for norm compliance with high number of norms and/or events to be handled efficiently by taking advantage of the resources of a distributed system. Coordination issues between nodes, such as conflict resolution, are out of the scope of this paper due to space constraints, but we refer to [10] for general ideas on how they can be tackled. In the current paper, we focus on how to split a normative context into separate smaller normative contexts, that is, into separate areas of interest.

This paper is structured as follows: first, an overview of the formalism used to define the normative monitor is provided. Secondly, this formalism is extended in order to detect information (event) dependencies among the components of the normative context, and among monitoring systems. The paper goes on by introducing an approach for dividing a normative context among several monitors. Later, related work is analysed and compared to the proposal presented in this paper. Finally authors' conclusions are provided and future work is outlined.

2. Normative Model

In this section, we introduce 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 [2].

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 will denote $\mathcal{S}$ as the set of all possible states of the world, where $\mathcal{S} = \mathcal{P}(O)$. We will 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 will 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).*

We can formalise the norms of Definition 1 as the equivalent deontic expression (using the formalism of [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 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[3]) *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. They 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).

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.

In order to track the normative state of an institution at any given point of time, we will define three sets: an instantiation set IS, a fulfillment set FS, a violation set VS, and

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

Event processed:

$$\frac{e_i = \langle \alpha, t, p \rangle}{\langle s \rangle \overset{ep}{\rhd} \langle s \cup \{p\} \rangle} \tag{1}$$

Norm instantiation:

$$\frac{activated(n, \Theta) \qquad n \in N \qquad \neg\exists n' \in N, n' \rightsquigarrow n \qquad \langle n, \Theta \rangle \notin is}{\langle is \rangle \overset{nii}{\rhd} \langle is \cup \{\langle n, \Theta \rangle\} \rangle} \tag{2}$$

Norm instance violation:

$$\frac{\neg maintained(\langle n, \Theta \rangle) \qquad NR = \bigcup_{n \rightsquigarrow n'} \langle n', \Theta \rangle \qquad n \in N \qquad \langle n, \Theta \rangle \in is \qquad \langle n, \Theta \rangle \notin vs}{\langle is, vs \rangle \overset{niv}{\rhd} \langle (is - \{\langle n, \Theta \rangle\}) \cup NR, vs \cup \{\langle n, \Theta \rangle\} \rangle} \tag{3}$$

Norm instance fulfilled:

$$\frac{deactivated(n, \Theta') \qquad n \in N \qquad \langle n, \Theta \rangle \in is \qquad \Theta' \subseteq \Theta}{\langle is, fs \rangle \overset{nif}{\rhd} \langle is - \{\langle n, \Theta \rangle\}, fs \cup \langle n, \Theta \rangle \rangle} \tag{4}$$

Norm instance violation repaired:

$$\frac{\langle n', \Theta \rangle \in fs \qquad n, n' \in N \qquad n \rightsquigarrow n' \qquad \langle n, \Theta \rangle \in vs}{\langle vs, rs \rangle \overset{nir}{\rhd} \langle vs - \{\langle n, \Theta \rangle\}, rs \cup \{\langle \langle n, \Theta \rangle, \langle n', \Theta \rangle \rangle\} \rangle} \tag{5}$$

Figure 1. Inference rules for the transition relation $\rhd$

a repairment set RS. Each of them contains norm instances $\{\langle n_i, \Theta_j \rangle, ..., \langle n_{i'}, \Theta_{j'} \rangle\}$. We adapt the semantics for normative states from [11]:

Definition 3 (Norm Lifecycle) *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 lifecycle 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 fulfillment set, VS is the violation set, and RS is the set of those norm instances ni' that have repaired a norm instance ni.

Definition 4 (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, $E = \mathcal{P}(P \times S)$.

Definition 5 (Normative Monitor) *A Normative Monitor M_N for a set of norms N is a tuple $M_N = \langle N, S, IS, VS, FS, RS, E \rangle$.*

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

Definition 6 (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* $\rhd \subseteq \Gamma_{M_N} \times L \times \Gamma_{M_N}$

The inference rules for the transition relation $\rhd$ are described in Figure 1.

This formalism, as shown in [2], has been reduced to the semantics of general production systems and an implementation in DROOLS is already available.

3. Formalizing Norm dependency

In this section we extend the monitoring formalism introduced in Section 2 for defining dependencies between formulas. Then, these dependencies are then extended to dependencies between norms and monitors.

3.1. Inter-formula dependency

Informally speaking, two formulas are dependent when they share common predicates and, therefore, some parts of the state of the world may affect them both at the same time. More formally, we define two formulas as mutually dependent if, and only if, there is at least one possible state of the world in $\mathcal{S}$ at which both formulas have a grounding of their predicates and both groundings share one or more predicates.

In order to formalise this, we introduce the transition function δ which given a formula and the actual state of the world will output a not partial grounding:

Definition 7 (State grounding function) *We define the function* δ *as:*

$$\delta : wff(\mathcal{L}_O) \times \mathcal{S} \rightarrow \vartheta_{(wff(\mathcal{L}_O),\mathcal{S})}$$

Given $f \in wff(\mathcal{L}_O)$ *and* $\sigma \in \mathcal{S}$*:*

$$\delta(f, \sigma) = \begin{cases} \Theta & \textit{iff } \exists \Theta : \exists g \in F(\sigma) : \Theta(f) \equiv g \\ \emptyset & \textit{otherwise} \end{cases}$$

Using the transition function δ we can state that two formulas are dependent if, given the same state of the world, it returns two sets of substitutions –groundings– with at least one common element:

Definition 8 (Formula dependency) *Given two formulas* f_x*,* f_y *such that* $f_x, f_y \in wff(\mathcal{L}_O)$ *we state* f_x *and* f_y *are dependent, and denote it by* $f_x \leftrightsquigarrow f_y$

$$f_x \leftrightsquigarrow f_y \Leftrightarrow \exists \sigma \in \mathcal{S} : \delta(f_x, \sigma) \cap \delta(f_y, \sigma) \neq \emptyset$$

The previous definitions do not take into consideration how the formulas are defined and thus are too general. In our particular case, we use formulas in propositional logic which are normalisable to DNF. The following definitions introduce how to compute $f_x \leftrightsquigarrow f_y$ with DNF formulas.

Given a formula f, we define $\mathcal{D}(f)$ as the DNF of f. The DNF is the disjunction of a set of conjunctive clauses. Given a conjunctive clause f', the function $\mathcal{C}(f')$ returns the set of predicates p on the clause. Formally: $\mathcal{C}(f') = \bigcup\limits_{p \in f'} p$. We use $\mathcal{C}(f')$ to define the function $\mathcal{F}(f)$ returning the predicates of a formula. The set of predicates of a formula in DNF form is the union of the set of predicates of every conjunctive clause f' in the formula:

Definition 9 (Predicates of a formula) *Given the DNF form of a formula $f = f'_1 \vee f'_2 \vee ... \vee f'_n$, the function $\mathcal{F}(f)$ is defined as follows:*

$$\mathcal{F}(f) = \bigcup_{f' \in \mathcal{D}(f)} \mathcal{C}(f')$$

For instance, let f_M be a formula that defines when an agent is working either on the camera ready version or on the presentation of a paper he has submitted to a conference:

$$\begin{aligned} f_M &= (camera\_ready\_of(R,P) \wedge submited\_to(P,C) \wedge working\_on(A,R)) \vee \\ &\quad presentation\_of(S,P) \wedge submited\_to(P,C) \wedge working\_on(A,S)) \\ \mathcal{F}(f_M) &= \{camera\_ready\_of(R,P), presentation\_of(S,P), submited\_to(P,C), \\ &\quad working\_on(A,R), working\_on(A,S)\} \end{aligned}$$

Intuitively, we can see that given two formulas f, f', the fact that f and f' have at least one common predicate is equivalent to the fact that they are dependent:

Proposition 1 *Given two formulas f, f':*

$$\mathcal{F}(f) \cap \mathcal{F}(f') \neq \emptyset \Leftrightarrow f \leftrightsquigarrow f'$$

In this section, the concepts of *predicate of a clause* and *predicate of a formula* have been used for introducing the concept of formula dependency. This last concept is used in the next section for introducing the concept of norm dependency.

3.2. Inter-norm dependency

This subsection extends the definitions provided in *Section* 3.1 to provide a formal definition of dependencies between norms.

First, we use $\mathcal{F}(f)$ to define the function $\mathcal{Q}(n)$ that returns the predicates of a norm:

Definition 10 (Predicates of a Norm) *Given a norm $n = \langle f_A, f_M, f_D, f_w, w \rangle$, the function $\mathcal{Q}(n)$ is defined as follows:*

$$\mathcal{Q}(n) = \bigcup_{f \in \{f_A, f_M, f_D\}} \mathcal{F}(f)$$

The concept of *predicates of a norm* allows us to introduce the concept of norm dependency. We state two norms are dependent if their sets of predicates are not disjoint:

Definition 11 (Norm Dependency) *Given two norms n and n' we state they are dependent, and denote it by $n \leftrightarrows n'$ when:*

$$n \leftrightarrows n' \iff \mathcal{Q}(n) \cap \mathcal{Q}(n') \neq \emptyset$$

The concept of norm-dependency is used to define the concept of normative monitor dependency. Two normative monitors are dependent if and only if for at least one norm in one of the monitors, there is a norm on the other monitor dependent on it:

Definition 12 (Monitor Dependency) *Given two monitors M_N and $M_{N'}$ we state they are dependent, and denote it by $M_N \leftrightarrows M_{N'}$ when:*

$$M_N \leftrightarrows M_{N'} \iff \exists n \in N, \exists n' \in N' : \mathcal{Q}(n) \cap \mathcal{Q}(n') \neq \emptyset$$

4. Architecture for distributed monitoring

This section introduces the architecture for splitting a normative context –bound to a single monitor– to a set of normative contexts bound to a set of interconnected and distributed monitors. The idea is to perform the context splitting in such a way that the connections between distributed monitors are reduced to the minimum. This will allow to reduce communication overhead between monitors and to reach higher efficiency.

4.1. From norms to graphs

This subsection introduces the idea of modelling a set of norms as a graph. First of all, we define the graph resulting from a set of norms. Then we introduce some basic concepts on *Strongly Connected Component* (*SCC* from now on) applied to graphs. Finally, we glue all these concepts together by applying SCCs [4] to sets of norms.

Definition 13 (Graph) *We define a (directed) graph as a pair $G = \langle V, E \rangle$. V is a finite set of nodes and $E \subseteq V \times V$ is a set of edges. We denote the existence of a path from v to w of length k as $path(v, w, k)$.*

From the definition of a graph we can model the normative framework as a graph:

Definition 14 *Given a monitor for a normative framework M_N we define a graph $G_N = \langle G_V, G_E \rangle$ such that:*

1. *The set of nodes in the graph is the set of norms: $G_V = N$*
2. $\forall_{n,n'} \in N \; : \; (n, n') \in G_E \Leftrightarrow n \leftrightarrows n'$

G_N denotes the graph representation of the norms in the normative monitor M_N.

Now we are ready to apply our normative model to *Strongly Connected Components*. A *SCC* of $G = \langle V, E \rangle$ is a maximal set $C \subseteq V$ such that: $\forall v, w \in C,\; path(v, w, i)$. A *SCC* is not trivial if: $\forall v, w \in C,\; path(v, w, i) \wedge i > 0$. Given a node $v \in V$, $SCC(v)$ denotes the SCC that contains v.

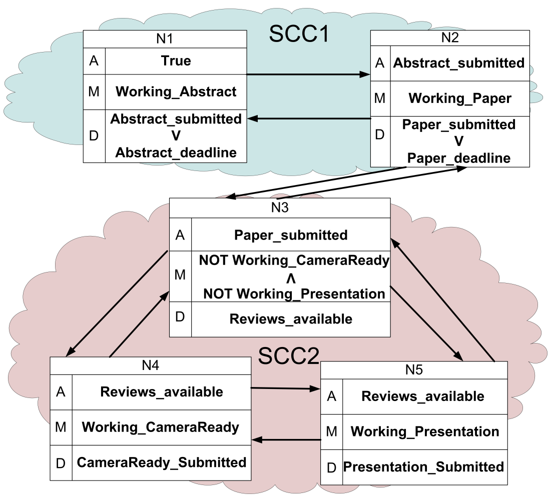

Figure 2. Graph with norm dependencies and resulting strongly connected components(SCC)

Definition 15 (Component graph of a normative context) *The component graph $\mathcal{G}_N$ of the normative context G_N is $\mathcal{G}_{\mathcal{N}}(G_N) = \langle V', E' \rangle$, where*

- $V' = \{SCC(v) \mid v \in V\}$
- $E' = \{(C = SCC(v), C' = SCC(v')) \mid C \neq C' \wedge (v, v') \in E\}$

Figure 2 shows a set of norms and dependency relationships among them. *SCC* are identified and depicted as clouds grouping the norms.

4.2. Distributing the graph

In the previous section we have introduced means for representing the normative context associated to a monitor as a Graph, and computing the *SCC* of the graph. The idea is replacing every SCC by a single vertex, in order to obtain a smaller graph, known as *component graph*. We can reduce the original problem to sub-problems on each particular SCC, plus one more sub-problem on the *component graph*. If such sub-problems are distributed among different computational nodes we have effectively distributed the computational process associated to the original problem.

When applying the notion of graph and *SCC* to a Normative Monitor for a set of norms, we can effectively reduce the problem of monitoring the original set of norms to several sub-problems: monitoring the sub-set of norms in each *SCC* of the graph. Each of these sub-problems can be computed by a different normative monitor. The process of splitting a Normative Monitor consists of three steps: 1) Computing the graph associated to the set of norms of the monitor. Then, computing the *component graph* of the graph. 2) For every *SCC* (that is, every node in the *component graph*), create an empty monitor that will compute these sub-set of norms from now on. 3) For every monitor created, subscribe it to another monitor[4] when the corresponding *SCC* are connected by a vertex on the *component graph* 14 vertices are created based on norm dependency. Therefore

[4]This subscription allows a monitor to notify events to other monitors dependent on it

we can state that two *SCC* are connected if they contain dependent norms. As seen in *Definition* 12, norm dependency can be extended to Normative Monitor dependency. Therefore, we can state that two monitors are connected when at least a pair of norms, one of each monitor, are dependent between themselves.

Algorithm 1 Normative Monitor Splitting

```
**Require:** $M_N = \langle N, S, IS, VS, FS, RS, E\rangle$
  $MonitorSet = \emptyset : graph = G_N : componentGraph = \mathcal{G}(graph)$
  **for all** $v \in componentGraph$ **do**
    $M_v = \langle v, S, \emptyset, \emptyset, \emptyset, \emptyset, E\rangle : MonitorSet = MonitorSet \cup M_v$
  **for all** $M_N, M_{N'} \in MonitorSet$ **do**
    **if** $M_N \leftrightarrows M_{N'}$ **then**
      $Subscribe(M_N, M_{N'})$
```

We will use the concepts and the algorithm introduced during this section to formalise monitor splitting. If a monitor becomes overflowed –e.g., it is receiving more events than it can process–, it can be split by *Algorithm* 1.

5. Related work

Artikis [3] proposed to use a knowledge representation framework for distributing part of the monitoring infrastructure in order to provide runtime support to large-scale multi-agent systems regulated by norms. Artikis defines a scenario based on the Open Packet World, where agents move on a grid-like world and score points when they deliver packets. Artikis runs a set of experiments on a 40×40 grid area and 10, 30 and 50 agents respectively, finding out the time needed to compute the social and physical state (that is, normative state) of a container is proportional to the number of events taking place in the container. By distributing the grid that represents the world in two (20×40) and four (20×20) containers, a gain in performance is achieved. Our proposal is similar to Artikis' in the sense that we aim to split the events into distributed and interconnected monitors. However, there is a fundamental difference. Our approach does not rely on a physical division of the normative context, and this provides an important benefit: our approach can be applied to scenarios where the representation of the world is either unclear, hard or even impossible to define.

Cheng [6] and Gupta [9] explore parallel execution as an approach to achieve higher execution speed on rule-based expert systems. Cheng creates a vertex for each rule in the production system and adds a direct edge between every pair of nodes if the rules represented by the nodes are dependent. After that, it computes the *SCC* on the graph, concluding every *SCC* is a *forward independent rule-set*. Gupta [9] explores high-speed implementations of a particular kind of rule-based production system (OPS5) via a parallel implementation of the RETE algorithm. When compared to Cheng's and Gupta's work, our approach is more generic, as we are building a dependency graph of a set of norms, no matter how these norms are computed internally by each monitor (either production rules, Java programs or alternative approaches). Our approach has been adapted to new techniques, mainly, substituting parallelism for distributed computing.

6. Conclusions and further work

In this paper we have presented a model to reduce a normative context to a SCC component graph, and thus facilitating the distribution in real-time of the interpretation of normative states and addressing the problem of efficiency of norm-compliance monitoring for the governance of distributed architectures. The use of SCC's allows us to apply state-of-the-art mechanisms, such as component detection algorithms [4]. Although our focus in this paper has been on regulative norms, we intend to extend the support to constitutional norms [1]. An important aspect of our approach is that, instead of focusing on the events, we intend to tackle the interpretation part, which is often rather computational consuming when working at a symbolic level. Thus, we have an architecture for distributed interpretation [10]. While in this paper we have focused on the structural part of this problem –i.e., how to split the context–, there is also the topic of how to coordinate the local nodes for handling their dependencies, and the topic of improving the context merging procedure to better balance load in the remaining monitors. We will present our approach to these issues in immediate future work.

Finally, we still have no extensive experimental results, but we will provide them in further work, along with an analysis of the algorithmic complexity of the distributed system and a procedure for merging divided contexts.

References

[1] H. Aldewereld, S. Alvarez-Napagao, F. Dignum, and J. Vázquez-Salceda. Making Norms Concrete. In *Proc. of 9th Int. Conf. on AAMAS 2010*, pages 807–814, May 2010.

[2] S. Alvarez-Napagao, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Normative Monitoring: Semantics and Implementation. In *COIN 2010*, pages 321–336. Springer, Berlin Heidelberg, May 2011.

[3] A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1):1–42, Jan. 2009.

[4] R. Bloem, H. N. Gabow, and F. Somenzi. An Algorithm for Strongly Connected Component Analysis in n log n Symbolic Steps. In *Formal Methods in System Design*, pages 37–56. Springer Science + Business Media, Inc., The Netherlands, 2006.

[5] F. Brazier, F. Dignum, V. Dignum, M. H. Huhns, T. Lessner, J. Padget, T. Quillinan, and M. P. Singh. Governance of Services: A Natural Function for Agents. In *Proceedings of the 8th International Workshop on Service-Oriented Computing: Agents, Semantics and Engineering*, Toronto, Canada, May 2010.

[6] A. M. K. Cheng. Parallel execution of real-time rule-based systems. In *Proceedings of Seventh International Parallel Processing Symposium*, pages 779–786, Newport, CA , USA, Apr. 1993. IEEE.

[7] F. Dignum, J. Broersen, V. Dignum, and J.-J. Meyer. Meeting the Deadline: Why, When and How. In *Formal Approaches to Agent-Based Systems*, pages 30–40. Springer, Berlin Heidelberg, Apr. 2004.

[8] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[9] A. Gupta, C. Forgy, and A. Newell. High-speed implementations of rule-based systems. *ACM Transactions on Computer Systems (TOCS)*, 7(2):119–146, May 1989.

[10] V. R. Lesser and L. D. Erman. Distributed interpretation: A model and experiment. *IEEE Transactions on Computers*, 29(12):1144–1163, Dec. 1980.

[11] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. In *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 156–171. Springer, Berlin / Heidelberg, May 2009.