Strategic Executions of Choreographed Timed Normative Multi-Agent Systems

Lăcrămioara Aştefănoaei
L.Astefanoaei@cwi.nl

Frank S. de Boer
F.S.de.Boer@cwi.nl

Mehdi Dastani
mehdi@cs.uu.nl

ABSTRACT

This paper proposes a combined mechanism for coordinating agents in timed normative multi-agent systems. Timing constraints in a multi-agent system make it possible to force action execution to happen before certain time invariants are violated. In such multi-agent systems we achieve coordination at two orthogonal levels with respect to states and actions. On the one hand, the behaviour of individual agents is regulated by means of social and organisational inspired concepts like norms and sanctions. On the other hand, the behaviour of sets of agents is restricted according to action-based coordination mechanisms called choreographies. In both cases, the resulting behaviour is constrained by time.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Agent Languages, Theory, Verification

Keywords

Formal models of agency, Normative systems, Coordination

1. INTRODUCTION

One of the challenges in the design and development of multi-agent systems is to coordinate and control the behaviour of the constituting agents. There are different approaches, from low level ones (e.g., channel-based coordination) to high level ones (e.g., normative or action-based artifacts), each with its own purpose and expressive power.

For example, the normative language proposed in [10] was designed to facilitate the implementation of norm-based organisation artifacts. Such artifacts refer to norms as a way to signal when violations take place and sanctions as a way to respond (by means of punishments) in the case of violations. Basically, a normative artifact observes the actions performed by individual agents, determines their effects in the environment (which is shared by all individual agents), determines the violations caused by performing the actions, and possibly, imposes sanctions. Thus a normative artifact can be used to *enforce* the system to be in a specific, i.e., non-violating, *state*.

Cite as: Strategic Executions of Choreographed Timed Normative Multi-Agent Systems, L. Aştefănoaei, F. S. de Boer and M. Dastani, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

On the other hand, the choreography language proposed in [5] was designed to allow the representation of global synchronisation and ordering conditions restricting the action execution of agents. Thus choreographies can be used to *enforce* specific *actions* to be executed. Introducing action-based coordination mechanisms while respecting the autonomy of the agents is however problematic. Choreographies might constrain agents' autonomy, however, this is a very common practise in multi-agent systems when specific properties need to be guaranteed. The advantage of the infrastructures proposed in [5] lies in their *exogenous* feature: the update of the agent's mental states is separated from the coordination pattern. Nobody changes the agent's beliefs but itself. Besides that choreographies are oblivious to mental aspects, they control without having to know the internal structure of the agent. More precisely, the degree of freedom of an agent can be seen, depending on the agent language, in the choice of plans or in the mechanism of handling failures (the choreography does not constrain the agent on how to select an appropriate "repair" operation). In these regards, the autonomy of agents is preserved.

Since their expressive power is not the same, in this paper, we consider a combination of the above approaches. Furthermore, we extend such approaches by explicitly modelling time. We do this by adapting the theory of timed automata [1]. There, time is modelled as clocks denoted by real-valued variables. Initially, all the clock variables are initialised with zero. They increase synchronously at the same uniform rate, counting time with respect to a fixed global time frame. Clocks should be seen as fictious, invented to express the timing properties of the system. We equip both agents and choreographies with clocks. In this way it is possible to model clock constraints which can (1) time restrict action execution, (2) enforce delays between actions and (3) enable the sanctioning of delays, for example, postponing to pay a fine. We emphasise this latter issue as being a fresh approach to introducing *timed normative rules*.

Both semantics of the normative and the choreography languages are operational, thus they have a natural encoding as rewrite theories. One of the advantages of prototyping languages as rewrite theories is that it makes it quick and easy to perform verification and to experiment with the language definitions. Prototyping the normative language as a rewrite theory has been already done using Maude [9], a rewriting logic software, as it is described in [4]. Furthermore, given instances of prototyped normative multi-agent systems have been verified with the Maude LTL model-checker [13]. The timed extension of both languages in a rewrite-based framework like Maude is practically feasible thanks to Real-Time Maude [20]. This is also the case for verifying choreographed timed normative multi-agent systems using the same technique of model-checking, however for timed systems.

We stress the importance of prototyping languages before im-

plementing them on a standard platform like Java, for example. It can be the case that during the process of prototyping new questions about design choices need to be taken, sometimes putting into light the lack of precision or weaknesses in definitions. If we take the case of normative artifacts, usually, their implementation boils down to fixing a scheduling policy for the application of normative rules. For example, one can think of an artifact which after each action execution considers *all* applicable normative rules such that *all* possible violations are signalled and resolved by means of sanctions. Or it can be that *only* the violations are recorded and at a later time corresponding sanctions are applied. In [4], the choice goes for the first option. Furthermore, the implementation of the normative artifact is hard-wired in the semantics of the normative language. There is *one* transition rule where *both* the execution of an action *and* the application of normative rules are considered. Such design decisions can give rise to further questions. There is a close dependence between the instrumentation of the normative rules and the semantics of the language. Thus, changes in the normative artifact must directly reflect in the semantics. Is there a more generic approach which would allow the implementation of different normative artifacts by using the same normative language?

In this paper we focus on such a generic approach. We propose the use of a *meta-level language* where we can define *strategies* as an alternative way to implement different normative artifacts without changing the semantics of the normative language. Thus, at the *object-level*, the normative multi-agent system is executed (its states change) with respect to the rewrite rules which give the executable semantics of the normative language. However, *how* the system changes is described at a *meta-level*, by strategically instrumenting the rules. By using strategies there is a clear separation between executions (at object-level) and control (at meta-level). This gives a great degree of flexibility which becomes important when the interest is in verification. In order to analyse or experiment with another type of normative artifact (thus a different agent society) one only needs to change the *syntax of the strategy* instead of changing *the semantics of the normative language*.

Our contribution is three fold. First, we introduce a timed agent-based framework. In this framework, we provide two distinct coordination mechanisms which, on the one hand, monitor and enforce certain normative states and, on the other hand, enforce certain actions to be executed. The advantages of our approach can be seen at both practical and theoretical level. Thanks to rewriting logic, we can prototype the timed normative and the timed choreography languages in Real-Time Maude. This makes it possible to (1) execute, by rewriting, and (2) verify, by model-checking choreographed timed normative multi-agent systems. At a theoretical level we provide the basis for a further analysis of properties of different classes of normative artifacts.

2. TIMED NORMATIVE ARTIFACTS

In this section we introduce a timed variant of the normative language presented in [4]. First, we present the standard time constructions which we further use in the paper. As we have already mentioned in the introduction, our idea of time comes from the theory of timed automata [1]. A *timed system* is a finite transition system extended with clock variables. Time advances only in states since transitions are instantaneous. Clocks can be reset at zero simultaneously with any transition. We usually denote by λ the set of clocks to be reset on transitions. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. States and transitions have *clock constraints*, defined by the following grammar:

$$\phi_c ::= x_c \leq t \mid t \leq x_c \mid x_c < t \mid t < x_c \mid \phi_c \wedge \phi_c,$$

where $t \in \mathbb{Q}$ is a constant and x_c is a clock variable. When a clock constraint is associated with a state, it is called *invariant*, and it expresses that time can elapse in the state as long as the invariant stays true. When a clock constraint is associated with a transition, it is called *guard*, and it expresses that the action may be taken only if the current values of the clocks satisfy the guard.

To record clock values one uses clock interpretations. A *clock interpretation* ν for a set of clocks λ assigns a real value to each clock. A clock interpretation ν is said to satisfy a clock constraint ϕ_c, $\nu \models \phi_c$, if and only if ϕ_c evaluates to true according to the values given by ν. For $\delta \in \mathbb{R}$, $\nu+\delta$ denotes the clock interpretation which maps every clock $x_c \in \lambda$ to the value $\nu(x_c) + \delta$. For any $\lambda_1 \subseteq \lambda$, $\nu[\lambda_1 := 0]$ denotes the clock interpretation which assigns 0 to every $x_c \in \lambda_1$ and agrees with ν over the other clocks.

2.1 Syntax

Due to space limit, we will illustrate the syntax of the timed normative language by means of an example. Before, we briefly provide an intuitive description of the key concepts. Please see [4] for a more rigorous presentation of the untimed normative language.

A timed normative multi-agent system is a collection of timed agents where the behaviour in time of the individual agents is monitored and normative rules are applied consequently. The choice of agent language is not relevant. However, for the sake of completeness, in this paper we only consider timed agent languages. In this way we can describe in a uniform manner a timed, agent-based framework. The untimed version can be obtained by simply dropping time constructions since the time extension we envisage is modular. Roughly, timed agents are agents equipped with clocks. These clocks can be seen as stop-watches which can be started and checked independently of one another, however they use the same unit to measure the passing of time. At each moment the clocks' values of any agent can be checked by an external observer. The observer cannot, however, change the agents' clocks values since it is only the agents that manipulate their own clocks. The way they can do this will be intuitively described later on in this section. The advantage of agents having their own clocks is that the normative system does not need to have a clock on its own. In order to (dis)allow the execution of actions at given instances of time or to punish delays it is sufficient[1] to consult the clocks of the agents.

We further make the remark that the agents themselves are not able to reason about the normative rules of the system since there is no assumption about the internals of individual agents. The only thing that agents can do is to perform *actions* in an *external environment* which is part of and controlled by the timed normative multi-agent system. Actions are of two types: either invisible or observable. An example of invisible ones are the actions for manipulating clocks. The ones which are of interest in a normative language are the observable actions. These are given in terms of *enabling conditions* and *effects*. The effects are recorded in the *brute state* of the environment. The enabling conditions are queries on the brute state and on the valuations of the agents' clocks.

The normative rules are either *counts-as* or *sanctions*. Syntactically they are given in the form of implications, $(\phi, \phi_c) \Rightarrow \psi$, where (ϕ, ϕ_c) generally denotes a *precondition* as a pair of a first order formula and a clock constraint and ψ, a *postcondition* as a list of literals. Informally, the meaning of counts-as (resp. sanctions) is to update the normative (resp. brute) state with the elements from the postcondition if the precondition is satisfied. Clock constraints

[1]Please note that by definition clocks cannot "break" or have "fake time units".

are present in the precondition because in a timed framework new violations and sanctions can arise due to time delays. For example not paying a fine in a given amount of time might entail the application of a new violation. Or a sanction might be cancelled when the expiration time has passed. The only difference between counts-as and sanctions is that the preconditions of counts-as query both brute and normative states while the preconditions of sanctions query only the normative state. This is because new sanctions reflected in the brute state of the system can entail the application of new counts-as rules.

We take as an illustration a timed variant of the train scenario described in [4]. Figure 1 represents a timed normative multi-agent

```
Agents:
    psg1 clock1 passenger_prog1   1
    psg2 clock2 passenger_prog2   1
Facts:
Effects:
    {not at_platform(X)}
                                  enter(X)
    {at_platform(X)}
    clock(X) < 10, {not ticket(X)}
                                  buy-ticket(X)
    {ticket(X)}
    {at_platform(X), not in_train(X)}
                                  embark(X)
    {not at_platform(X), in_train(X)}
    {fined(X, Y), not paid-fine(X)}
                                  pay-fine(X, Y)
    {paid-fine(X, Y)}
Counts-As rules:
    at_platform(X) /\ not ticket(X) =>
                              viol_ticket(X)
    ( fined(X, Y) /\ not paid-fine(X),
    clock(X) > 100 ) => viol_fine(X, Y)
Regimentation rules:
    in_train(X) /\ not ticket(X) => viol_|_
Sanction rules:
    viol_ticket(X) => fined(X, 25)
    viol_fine(X, Y) => fined(X, 2*Y)
```

Figure 1: A Timed NMAS Program

system program consisting of two agents `psg1` and `psg2` with their clocks `clock1` (resp., `clock2`) and their implementations in the files `passenger_prog1` (resp., `passenger_prog2`). The initial brute `Facts` are empty, thus by absence the fact that `psg1` is not in the train is true. The `Effects` indicate the changes in the environment, for instance, `psg1` performing `enter` when not at the platform, results in `psg1` being at the platform (with or without a ticket). The enabling conditions can include clock constraints, for example, in our scenario, buying a ticket is allowed only if this is done at most until `clock(X)`[2] shows 10 units of time. The `Counts-As rules` determine the normative effects for a given state of the multi-agent system. In our scenario, being at the platform without having a ticket counts-as a specific violation (`viol_ticket(X)`). It is also the case that fines which are not paid within 100 units of time entail new violations. These rules function as an *enforcement* mechanism [14] which is based on the idea of responding to a violation such that the system returns to an acceptable state. However, there are situations where stronger requirements need to be implemented, for example, where it is never the case that agents enter the train without having a ticket. This is what is called *regimentation* and in order to implement it we consider the literal $viol_\perp$ by means of regimentation rules. As we will see, the operational semantics of the language ensures that $viol_\perp$ can never hold during any run of the system. Intuitively, regimentation can be thought of as placing gates blocking an agent's action. In our scenario, the `Sanction rules` determines the punishment for `viol_ticket(X)`, which is the sanction `fined(X,25)`. Furthermore, not paying a fine in time results in doubling the fine.

In order to analyse the above scenario, we assume a specific behaviour for the agents `psg1` and `psg2`. As we have mentioned, they are timed agents. Examples of timed agent languages are described in [5]. We will not focus on their precise syntax and semantics, as this is out of the scope of our paper, but describe, as a short recipe, how to cook such languages. Roughly, the design methodology to time a language can be reduced to the following steps. First, each agent is assigned a set of clocks which can be checked by any external observer. Second, the agent language is extended with two basic constructions which can be used in the agent program for *delaying* and *resetting* clocks. For example, the delay mechanism can be provided as a default action, $\phi \rightarrow I$, where ϕ is a query on the belief base of the agent and I is an invariant like $x_c \leq 1$. This mechanism allows time to pass in a state under the condition that ϕ is satisfied and as long as I is valid. Third, action calls are surrounded by blocks, (ϕ_c, a, λ), consisting of clock constraints ϕ_c and sets λ of clock resets. A timed extension of an agent language following these steps has the important feature that "the ontology of actions is timeless". This implies that reflecting the syntactical time constructions at the semantic level can be done in a modular way. First a new transition rule for delay actions is needed for the passing of time. Second, the rule for action execution is updated such that for each action call (ϕ_c, a, λ) two additional operations are performed: (1) a check if ϕ_c is satisfied by the values of clocks and (2) a reset of the clocks in λ.

For our case of study, we assume that `psg1` can be dishonest, and its plan, `p1` in Figure 2, is to buy a ticket if its clock `xc` shows less than 9 units, otherwise it will enter without a ticket. If `psg1` manages to embark, it spends at most 200 units in the train. We note that if `psg1` does not buy a ticket and the normative system applies the sanction `fined(psg1, 25)`, then if `psg1` delays for more than 100 units of time, a new sanction consisting of the doubling of the fine is entailed. This would not be the case if `psg1` delays for less than 100 units and intends to pay the fine (`pay-fine` is in `p'`). On the other hand, we assume that `psg2` is correct and its plan, `p2`, is to always buy a ticket before entering the platform. Furthermore, it has up to 8 units of time to decide what ticket to buy and it resets the clock after the action is done. The delay `true -> yc < 10` means that `psg2` waits at most 10 units of time before embarking the train.

```
p1  =  ( ((xc < 9), buy-ticket) +
         ((xc >= 9), enter) );
         embark; (true -> (xc < 200)); p'

p2  =  ((yc < 8), buy-ticket, yc := 0);
       enter; (true -> yc < 10); embark
```

Figure 2: The plans of `psg1` and `psg2`

In Figure 2 "`;`" (resp. "`+`") denotes the usual sequencing (resp. choice) operator. Furthermore, to simplify notation, the blocks (ϕ, a, λ), are written as pairs (ϕ_c, a) (resp. a) whenever the execution of a does not reset any clock (resp. and additionally ϕ_c is considered as being true).

2.2 Operational Semantics

A timed normative multi-agent system state $\langle \mathbf{A}, \sigma_b, \sigma_n \rangle$ records the configuration of the constituting timed agents, $\mathbf{A} = \{(A_1, \nu_1), (A_2, \nu_2), \ldots, (A_n, \nu_n)\}$, together with the brute (σ_b) and normative states (σ_n). We recall from Section 2 that ν denotes clock

[2]For our scenario, `clock(X)` should be seen as a "library function" returning the valuation of the clock of agent `X`.

interpretations. Thus ν_i represents the current clock values of A_i. The states of a timed normative multi-agent system change depending on the "event" arising in the system. Roughly, updates of the brute state σ_b are triggered after the execution of actions and after the application of sanctions. Updates of the normative state σ_n are triggered after the application of counts-as rules. When a regimentation rule (a counts-as with $viol_\perp$ as postcondition) is applicable, the system enters a deadlock state, denoted by $\perp$. These changes can be modelled by means of the following transition rules which, in fact, give the semantics of the timed normative language:

$$\frac{(\phi_c, \{\phi\}\alpha\{\psi\}) \quad (A_i, \nu_i) \stackrel{\alpha}{\Rightarrow} (A_i', \nu_i') \quad \theta \in Sols(\sigma_b \models \phi) \quad \nu_i \models \phi_c}{\langle \mathbf{A}, \sigma_b, \sigma_n\rangle \rightarrow \langle \mathbf{A'}, \sigma_b \uplus \psi\theta, \sigma_n\rangle} \text{ (act)}$$

$$\frac{((\phi, \phi_c) \Rightarrow viol_\perp) \in \mathcal{R} \quad Sols(\sigma_b \cup \sigma_n \models \phi) \neq \emptyset \quad \nu^{\mathbf{A}} \models \phi_c}{\langle \mathbf{A}, \sigma_b, \sigma_n\rangle \rightarrow \perp} \text{ (reg)}$$

$$\frac{((\phi, \phi_c) \Rightarrow \psi) \in \mathcal{C} \quad \theta \in Sols(\sigma_b \cup \sigma_n \models \phi) \quad \nu^{\mathbf{A}} \models \phi_c}{\langle \mathbf{A}, \sigma_b, \sigma_n\rangle \rightarrow \langle \mathbf{A}, \sigma_b, \sigma_n \uplus \psi\theta\rangle} \text{ (counts-as)}$$

$$\frac{((\phi, \phi_c) \Rightarrow \psi) \in \mathcal{S} \quad \theta \in Sols(\sigma_b \cup \sigma_n \models \phi) \quad \nu^{\mathbf{A}} \models \phi_c}{\langle \mathbf{A}, \sigma_b, \sigma_n\rangle \rightarrow \langle \mathbf{A}, \sigma_b \uplus \psi\theta, \sigma_n\rangle} \text{ (sanction)}$$

where $\mathbf{A'} = (\mathbf{A} \setminus \{(A_i, \nu_i)\}) \cup \{(A_i', \nu_i')\}$ and $\mathcal{R}$, $\mathcal{C}$, $\mathcal{S}$ are sets of regimentation, counts-as and sanction rules. By abuse of notation we refer to $\alpha(\overline{x})$ (resp. $\phi(\overline{x})$, $\psi(\overline{x})$) as α (resp. ϕ, ψ) when $\overline{x}$, the set of variables, is not relevant. The double arrow $\stackrel{\alpha}{\Rightarrow}$ denotes the transitive closure of $\stackrel{\alpha}{\rightarrow}$ with respect to τ steps, $\rightarrow$, for internal actions, and δ steps, $\stackrel{\delta}{\rightarrow}$, for delay actions. $Sols(\sigma \models \psi)$ represents the set of all matchers of ψ against σ. The notation $\psi\theta$ denotes the usual application of the substitution θ to the set of literals ψ. The symbol $\uplus$ denotes the *update* operation. Its semantics is as follows:

$$\begin{cases} \sigma \uplus l = \sigma \cup \{l\}, & l \in \psi\theta \\ \sigma \uplus \neg l = \sigma \setminus \{l\}, & \neg l \in \psi\theta \\ \sigma \uplus \psi\theta = (\sigma \uplus a) \uplus (\psi\theta \setminus \{a\}), & a \in \psi\theta \wedge \mid \psi \mid > 1 \end{cases}$$

which means that for each atom a from $\psi\theta$, if it is a positive literal l then l is added to σ and if it is a negative literal $\neg l$ then l is removed from σ. We further denote by $\nu^{\mathbf{A}}$ the valuation of all of the clocks of all agents from $\mathbf{A}$, i.e., $\nu^{\mathbf{A}} = \{\nu_i \mid 1 \leq i \leq n\}$[3]. We say that $\nu^{\mathbf{A}} \models \phi_c$ is true whenever the valuations from $\nu^{\mathbf{A}}$ satisfy the clock constraint ϕ_c. For example, let us consider a multi-agent system with two agents, A_1 and A_2 with A_1 having two clocks x_c, y_c and A_2 having one clock z_c. Let us further assume that we "freeze" the system for an instant in a state where the clock interpretation ν_1 of the first agent is $\nu_1(x_c) = 2, \nu_1(y_c) = 4$ and the clock interpretation ν_2 of the second agent is $\nu_2(z_c) = 6$. We have that, in the clock interpretation $\nu^{\mathbf{A}}$, i.e., $\nu_1 \cup \nu_2$, the clock constraint $\phi_c = (x_c < 3) \wedge (z_c > 5)$ is satisfied however this is not the case for the clock constraint $\phi_c = (y_c < 3)$.

The meaning of the transition rules is as follows. The transition (act) takes place whenever an agent A_i can perform an action α with the additional requirements that (1) there is a substitution θ such that the precondition ϕ of α matches the current set of brute facts σ_b and (2) the clock constraint ϕ_c is satisfied by the current clock interpretation ν_i of A_i. The rule (act) then evaluates the effects ψ of α and the configuration of the timed normative multi-agent system changes such that it reflects the update of σ_b with $\psi\theta$ and the new configuration of A_i with its possibly new clock interpretation which might have changed while δ steps. We recall that time passes inside agents by means of the internal built-in delay actions. The transition (reg) can take place whenever there is a regimentation rule with a precondition matching the current set of brute and normative facts and a clock constraint satisfied by the current clock interpretation of all clocks existing in the system. The rule (reg) then blocks the execution of actions whose effects are regimented. The mechanism for the transitions (counts-as) and (sanction) is similar. They are both meant to apply correspondingly counts-as and sanction rules.

[3]In order to have a well-defined interpretation $\nu^{\mathbf{A}}$ the set of clock variables of individual agents must be disjoint

3. TIMED CHOREOGRAPHIES

The normative constructions described previously are meant to provide a monitoring mechanism and furthermore, to enforce the system to be in a certain state, non-violating or conformant for example. They cannot, however, enforce certain *actions* to take place. As we have mentioned in the introduction, it is sometimes important to require a particular order on the execution of actions. This is the case when using a scheduling policy and the result could be a better performance or an increased efficiency of the multi-agent system. This is why, based on the formalism from [5], we provide a mechanism for implementing action-based coordination artifacts by means of timed transition systems.

3.1 Syntax

We see timed choreographies as represented by a particular type of regular expressions. Thus, we define them using structural induction. The basic cases can be divided into two groups. One group is denoted by l_δ and is meant to represent the passing of time by delaying clocks. The other group is denoted by l_α and is meant to represent timed action synchronisation. Synchronisation is denoted by the parallel operator "$\|$". The parallel operator applies on pairs (A, α)[4] consisting of agents and action names. In order to have a more expressive language, we also allow *action variables* instead of action names, i.e., the pairs can have the form (A, x_a). Timing the synchronisation is modelled by surrounding the constructions l_α by clock constraints and resets. We then have that timed choreographies are any combination of l_δ and l_α obtained with the usual sequence ";", choice "+" and Kleene "*" operators. In BNF notation, timed choreographies are defined as follows:

$$\begin{array}{lll} l_\delta & ::= & x_c \leq t \\ l_\alpha & ::= & (A, \alpha) \mid (A, x_a) \mid (l_\alpha \parallel l_\alpha) \\ ch & ::= & l_\delta \mid l_\alpha \mid (\phi_c, l_\alpha, \lambda) \mid ch; ch \mid ch + ch \mid ch^* \end{array}$$

with x_a being an action variable. We use the following naming convention. Action variables are denoted by small letters with a as subscript (x_a, y_a, z_a...) in order to distinguish them from clock variables. They are meant to be placeholders for action names. Action variables are seen as global static variables, thus once bound their value cannot be changed. The binding is according to the actions that an agent is enabled to execute at a given time. Variable bindings are recorded as substitutions.

Before we present the semantics of timed choreographies, we take, as an illustration, a few examples with intuitive meaning. Let us first consider the following timed choreography ch_1:

[4]Since "$\|$" is associative and commutative, for simplicity, we use the notation $\|_{\mathcal{I}}\ (A_i, \alpha_i)$ to denote $(A_{i_1}, \alpha_{i_1}) \parallel \ldots \parallel (A_{i_j}, \alpha_{i_j})$ where $\mathcal{I} = \{i_1, \ldots, i_j\}$ and $j \geq 2$. This is also the case for x_a.

`((zc < 6), (psg2, buy-ticket))*; (psg2, xa).`

This is a choreography for the agent `psg2`. Intuitively, it specifies that `psg2` can buy tickets for at most 6 units of time and after it can execute an arbitrary action `xa`. Let us now consider the timed choreography ch_2:

`(psg1, buy-ticket);((zc≥7),(psg1, xa))`

which says that the agent `psg1` buys one ticket and as soon as `zc` shows 7 it performs an arbitrary action `xa`. If we compose ch_1 and ch_2 by a sequence operator we can add that the action `xa` performed by `psg1` is the same as the last one performed by `psg2`. If we also recall that the plan of `psg1` is to enter without tickets if more than 9 units of time have passed, we can further note that the clock constraint `(zc < 6)` ensures that such a situation never happens since `psg2` has always time to buy tickets. Thus it is impossible for `psg2` to behave dishonestly.

In order to illustrate the use of the parallel operator let us further consider the timed choreography ch_3:

`(zc < 10);`
`((psg1, embark)‖(psg2, embark), (zc:=0))`

and analyse it in the context of ch_f = $ch_1; ch_2; ch_3$. The choreography ch_f says that after both agents bought their tickets and performed the same action `xa`, the whole system delays for at most 10 units while the agents wait for the arrival of the train and after they both embark synchronously. This synchronised action happens with a reset of `zc`. We make one last remark with respect to ch_f. The reader might have already noticed that `xa` can only be the action `enter` since it is the only one enabled after the agents buy tickets. As we will see in what follows, the substitution `[xa/enter]` is recorded in all choreography states that precede the one where the binding takes place.

3.2 Operational Semantics

We give operational semantics to timed choreographies such that it is easily integrated into the timed normative language. Before presenting the semantics we show how any timed choreography can be accepted by a timed automaton[5]. We do this following the standard approach from [17]. The timed automaton is built by induction on the structure of the choreography. Due to space limits, we will not provide a complete analysis of all cases but briefly describe the construction. The timed automaton associated with a basic choreography $x_c \leq t$ consists of only one state to which we associate the invariant $x_c \leq t$. There are no transitions in this case. The timed automaton associated with a basic choreography l_α (resp. $(\phi_c, l_\alpha, \lambda)$) has two states and one transition labelled with l_α (resp. $(\phi_c, l_\alpha, \lambda)$). Given $\mathcal{A}^{ch_1}$ and $\mathcal{A}^{ch_2}$ the timed automata associated with the choreographies ch_1 and ch_2, the timed automaton $\mathcal{A}^{ch_1;ch_2}$ is the one obtained by concatenating $\mathcal{A}^{ch_1}$ and $\mathcal{A}^{ch_2}$. As an illustration of how timed automata look like, Figure 3 describes the one associated with the timed choreography ch_f.

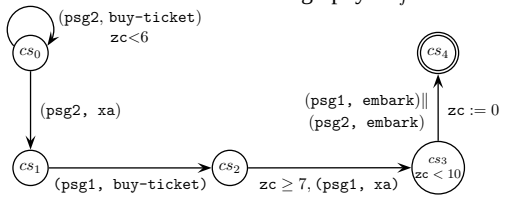

Figure 3: The Timed Automaton $\mathcal{A}^{ch_f}$

Given a choreography, a timed automaton can always be constructed, as it is stated in Proposition 3.1 and the proof is can be reduced to the basic construction steps described above.

PROPOSITION 3.1. *For any timed choreography ch there exists a timed automaton $\mathcal{A}^{ch}$ which accepts ch.*

We define the semantics of timed choreographies[6] $\mathcal{A}^{ch}$ by means of transition systems where the states are denoted as $\langle cs, \nu\rangle$ with cs being a state of $\mathcal{A}^{ch}$ and ν the current clock interpretation. The transition rules are with respect to the transition labels of $\mathcal{A}^{ch}$, that is, corresponding to delay (l_δ) and to agents' actions (l_α):

- $\langle cs, \nu\rangle \xrightarrow{\delta} \langle cs, \nu+\delta\rangle$ if $\nu+\delta \models I(cs)$ for any $\delta \in \mathbb{R}^+$
- $\langle cs, \nu\rangle \xrightarrow{\|_{\mathcal{I}}(A_i, x_{a_i})} \langle cs', \nu'\rangle$ if $cs \xrightarrow{\phi_c, \|_{\mathcal{I}}(A_i, x_{a_i}), \lambda} cs'$, $\nu \models \phi_c$, $\nu' = \nu[\lambda := 0]$ and $\nu' \models I(cs')$.

The first rule says that the choreography can pass time as long as the new valuation does not violate the invariant $I(cs)$ associated with the state cs. The second rule says that for any label $(\phi_c, \|_{\mathcal{I}}(A_i, x_{a_i}), \lambda)$ in $\mathcal{A}^{ch}$ we construct a transition labelled $\|_{\mathcal{I}}(A_i, x_{a_i})$ from $\langle cs, \nu\rangle$ only if ϕ_c is satisfied by the current clock interpretation ν and if after resetting in ν the clocks from λ the new interpretation ν' does not violate the invariant associated with cs'.

3.3 Timed Normative Systems Revisited

Adding timed choreographies to the timed normative language from Section 2 implies that we need to revise the semantics of the language. We recall that the states of the timed normative systems were defined as $\langle \mathbf{A}, \sigma_b, \sigma_n\rangle$. In the context of timed choreographies, this is no longer enough since they should reflect also the states of choreographies. We denote their new configuration by the notation $\langle \mathbf{A}, \sigma_{ch}, \sigma_b, \sigma_n\rangle$. The symbol σ_{ch} denotes triples (cs, ν, θ_{ch}). We use it to record the current clock interpretation of the choreography and the active substitution of action variables.

Taking into account the two transitions giving the semantics of timed choreographies, we need to extend the semantics of the timed normative language such that the new states $\langle \mathbf{A}, \sigma_{ch}, \sigma_b, \sigma_n\rangle$ change with respect to the "directions" given by the choreography. This means that we need to (1) add a rule for passing time when the choreography indicates a delay and (2) change the rule (act) such that only the actions specified by the choreography are executed. We first consider the rule (delay) for passing time:

$$\frac{\langle cs, \nu\rangle \xrightarrow{\delta} \langle cs, \nu+\delta\rangle \qquad \bigwedge_i ((A_i, \nu_i) \xrightarrow{\delta} (A_i, \nu_i+\delta)) \qquad (\nu^{\mathbf{A}} \cup \nu + \delta) \models I(cs)}{\langle \mathbf{A}, (cs, \nu, \theta_{ch}), \sigma_b, \sigma_n\rangle \rightarrow \langle \mathbf{A'}, (cs, \nu+\delta, \theta_{ch}), \sigma_b, \sigma_n\rangle}$$

where **A'** is $\{(A_i, \nu_i + \delta) \mid 1 \leq i \leq n\}$. This rule says that the whole system can delay δ units as long as the updated valuations do not violate the invariant of the current choreography state.

The rule (sync-act) for timed synchronised action execution replaces the rule (act). The changes are as follows:

$$\frac{\begin{array}{c}\langle cs, \nu\rangle \xrightarrow{\|_{\mathcal{I}}(A_i, x_{a_i})} \langle cs', \nu'\rangle \\ \bigwedge_i ((A_i, \nu_i) \overset{\alpha_i}{\Rightarrow} (A'_i, \nu'_i)) \qquad \bigwedge_i (\phi_{c_i}, \{\phi_i\}\alpha_i\{\psi_i\}) \\ \theta \in Sols(\sigma_b \models \bigwedge_i \phi_i) \qquad \theta'_{ch} \in Sols(\bigwedge_i (\alpha_i \models x_{a_i}\theta_{ch})) \\ \nu^{\mathbf{A}} \cup \nu \models \bigwedge_i \phi_{c_i} \qquad \nu^{\mathbf{A'}} \cup \nu' \models I(cs')\end{array}}{\langle \mathbf{A}, (cs, \nu, \theta_{ch}), \sigma_b, \sigma_n\rangle \rightarrow \langle \mathbf{A'}, (cs', \nu', \theta_{ch}\theta'_{ch}), \sigma_b \uplus \psi', \sigma_n\rangle}$$

[5]For the present context, it is sufficient to see timed automata as timed transition systems with accepting states

[6]For convenience, we use $\mathcal{A}^{ch}$ instead of ch.

where **A** is $\{(A_i, \nu_i) \mid i \in \mathcal{I}\} \cup$ **A"** (resp. **A'** is $\{(A'_i, \nu'_i) \mid i \in \mathcal{I}\} \cup$ **A"**), with **A"** = $\{(A_j, \nu_j) \mid j \in \{1, \ldots, n\} \setminus \mathcal{I}\}$, ψ' is $\{\psi_i\theta \mid i \in \mathcal{I}\}$ and $I(cs)$ is the invariant associated with the choreography state cs. In the construction $Sols(\bigwedge_i(\alpha_i \models x_{a_i}\theta_{ch}))$, by abuse of notation we use action names α_i as atoms. A solution in this case is a substitution, either the identity or one which binds variables x_{a_j} not bound already by θ_{ch}. With respect to the previous rule (act), this rule says, in addition, that only the agents from the subset $\{A_i \mid i \in \mathcal{I}\}$ are allowed to execute actions while the ones from **A"** remain unchanged. Furthermore, if for an agent A_i the choreography only specifies that it can do an arbitrary action x_a and if the set $E(A_i)$ of enabled actions of A_i is not empty, then x_a will be bound to an action name from $E(A_i)$. This binding is recorded in the new state σ_{ch} such that whenever x_a appears again in the choreography it will be substituted by the binding.

4. EXECUTING NORMATIVE SYSTEMS

In [4] it is described how the untimed version of the normative language presented in Section 2 can be prototyped as a *rewrite theory*. The process of prototyping the timed language as a *real-time rewrite theory* ([20]) is similar, nevertheless longer and slightly more difficult to follow due to a more complex notation. This and also the space limit are the reasons why we decide to take the untimed normative language as reference from now on until the end of the paper. Besides, we appraise as valuable the conciseness and the elegance of rewriting logic which ease the reading.

A rewrite theory consists of a signature (types and function symbols), equations and rewrite rules. In our case, the signature describes the states of the normative multi-agent system. The rewrite rules describe how the states change. There is a natural encoding of transition rules as *conditional rewrite rules*. The general mathematical format of a conditional rewrite rule is:

$$l : t \rightarrow t' \; if \; (\bigwedge_i u_i = v_i) \wedge (\bigwedge_j w_j : s_j) \wedge (\bigwedge_k p_k \rightarrow q_k)$$

which says that l is the label of the rewrite rule $t \rightarrow t'$ which is used to "rewrite" the term t to t' when the conditions on the left side are satisfied. Such conditions can be either equations like $u_i = v_i$, memberships like $w_j : s_j$ (that is, w_j is of type s_j) or other rewrites like $p_k \rightarrow q_k$. For example, the corresponding rewrite rule for the untimed transition (act) from Section 2 is:

$$act : \langle \{A_i, \mathbf{A}\}, \sigma_b, \sigma_n \rangle \rightarrow \langle \{A'_i, \mathbf{A}\}, update(\sigma_b, \psi\theta), \sigma_n \rangle \; if \\ A_i \xrightarrow{\alpha} A'_i \wedge \alpha = (\phi, \psi) \wedge \theta = match(\sigma_b, \phi)$$

where $update$ and $match$ are functions defined by equations. Due to space limit, we do not further explain the encoding of the language as a rewrite theory. From the same reason neither do we explain the encoding of the choreography language. We only note that prototyping the untimed $\mathcal{A}^{ch}$ as a rewrite theory is a straightforward process. However, to make the connection between the prototypes of choreographies and normative systems, that is, to encode the transition (sync-act) from Section 3 is slightly more complicated because the set $\mathcal{I}$ has an arbitrary size[7]. To understand the next section, we only need to remember that each transition has a corresponding rewrite rule labelled with the same name.

4.1 Promoting Strategies

Having the normative language encoded as a rewrite theory we can execute normative multi-agent systems by rewriting. Since there might be more rewrite rules applicable at the same time, the execution process is highly nondeterministic. Depending on the application order of the rewrite rules *act*, *reg*, *counts-as* and *sanction* we execute, in fact, different types of "agent societies" corresponding to specific normative artifacts. For example, in a "totalitarian" agent society, the normative artifact monitors each action execution and applies all active normative rules. Such an artifact corresponds to a scheduling policy which specifies that first a check if no *reg* rule is enable should occur, then all *counts-as* rules are applied until no longer possible, and finally all *sanction* rules. The process is reiterated until it is no longer the case.

In the next section we show how we can use rewriting strategies to specify different scheduling policies of the normative rules. Thus we reduce the inherent nondeterminism in the semantics of the normative language by means of strategies. A strategy language S can be viewed as a transformation of a rewrite theory R into $S(R)$ such that the latter represents the execution of R in a controlled way, i.e., the application of rewrite rules is controlled by the strategy.

We make a short remark that to implement normative artifacts at the object-level means to encode directly into the semantics of the normative language specific procedures based on closure set computations. This creates a dependency between the normative artifact and the normative language. Such a dependency has the implication that changing to a different normative artifact must be reflected in the semantics of the normative language.

Following [12], we promote the design principle that automated deduction methods (e.g., closure sets) should be specified *declaratively* and not *procedurally*. Depending on the normative multi-agent system application, specific algorithms for implementing the normative artifact should be specified as *strategies* to apply the rewrite rules. This implies that the control of the rewriting is at the meta-level. The separation between *execution* (by rewriting) at the object-level and *control* (of rewriting) at the meta-level makes it simpler to reason in a modular way about normative multi-agent systems. When something goes wrong in the system one can first verify the normative artifact for an error. Only if this process is unsuccessful one needs to focus on debugging one by one the constituting agent programs.

4.2 Normative Artifacts as Strategies

In this section we briefly describe a strategy language S for implementing normative artifacts. The language has been introduced in [12]. Given the untimed normative language from Section 2, we denote the corresponding rewrite theory by R. Given a normative multi-agent system written as a program in the normative language, its initial state has a corresponding term which we denote by t. This term can be rewritten by the rewrite rules from R. Given a strategy expression s in the strategy language S, the application of s to t is denoted by $s@t$. The semantics of $s@t$ is the set of successors which result by rewriting t in $S(R)$.

The simplest strategies we can define in the strategy language S are the constants *idle* and *fail*: *idle* @ $t = \{t\}$, *fail* @ $t = \emptyset$. Another basic strategy consists of applying to a normative multi-agent system state t a rule identified by one of the labels: *act*, *reg*, *counts-as* or *sanction*, possibly with instantiating some variables appearing in the rule. The semantics of $l@t$, where l is one of the above rule labels, is the set of all terms to which t rewrites in one step using the rule labelled l. For example, applying the strategy *act* to the untimed initial[8] normative multi-agent system from Figure 1 has 2 solutions corresponding to each agent executing the action from the head of their initial plans, i.e., one solution reflects that

[7]Technically, we address this issue by consuming one by one the indexes from $\mathcal{I}$ and maintaining a history of remaining indexes

[8]To simplify, we take untimed `p1` as `enter; embark` (untimed `p2` is left unchanged, `buy-ticket; enter; embark`).

psg1 entered the platform and the other, that psg2 bought a ticket.

The language S allows further strategy definitions by combining them under the usual regular expression constructions: concatenation (";"), union ("|"), iteration ("*", "+"). Thus, given the strategies E, E', the strategy $(E; E')@t$ is defined as $E'@(E@t)$, that is, E' is applied to the result of applying E to t. The strategy $(E \mid E')@t$ defined as $(E@t) \cup (E'@t)$ means that both E and E' are applied to t. The strategy $E^+@t$ is defined as $\bigcup_{i \geq 1}(E^i@t)$ with $E^1 = E$ and $E^n = E^{n-1}; E$, $E^* = idle \mid E^+$, thus it recursively re-applies itself.

The *if-then-else* combinators are denoted by E ? E' : E'' and their definition is (if $(E@t) \neq \emptyset$ then $E'@(E@t)$ else $E''@t$ fi) with the meaning that if, when evaluated in a given state term, the strategy E is successful then the strategy E' is evaluated in the resulting states, otherwise E'' is evaluated in the *initial* state. This strategy is further used to define:

not(E) = E ? *fail* : *idle* *try*(E) = E ? *idle* : *idle*
test(E) = *not*(E) ? *fail* : *idle* $E!$ = E^* ; *not*(E)

which have the following meaning. The strategy *not* reverses the result of applying E. The strategy *try* changes the state term if the evaluation of E is successful, and if not, returns the initial state. The strategy *test* checks the success/failure result of E but it does not change the initial state. The strategy $E!$ "repeats until the end".

We now describe how to implement different normative multi-agent systems using strategies. We start with:

vigilant = *test*(*act* ; *reg*) ? *fails* :
(*act* ; *counts-as* ! ; *sanction* !)!

saying that actions are executed only if they do not enable the application of regimentation rules (in which case the strategy fails). After executing an action, counts-as rules are applied until no longer possible. Finally, all corresponding sanctions are applied. This process is iterated until no action can be executed. In our train scenario, if we assume that a fine equals having a ticket, then the result of applying this strategy reflects that both agents are in the train with tickets and that previously, psg1 has been sanctioned. If this were not the case, then the system is in a deadlock state because psg1 embarks without a ticket, thus enabling the application of the regimentation rule. We note that a simple change like substituting ";" by "|" in *counts-as* ! ; *sanction* ! leads to a less restrictive normative system. An illustrative scenario is that of a video camera monitoring in a supermarket, or of a radar measuring the velocity of the passing vehicles. In such cases, sanctions do not necessarily follow immediately after recording an infraction.

A more restricted society is implemented by means of:

totalitarian = (*test*(*act* ; *reg*) ? *fails* :
(*act* ; (*counts-as* ! ; *sanction* !)!))!

saying that the process of applying counts-as rules followed by sanctions is iterated until it is no longer possible. This characterises scenarios where the application of a sanction enables the application of a new counts-as. We take, for instance, a traffic scenario where an actor drives through the red light, thus violating the traffic law. Consequently, a fine is applied. We assume that this is done automatically by withdrawing a certain amount of money from the actor's account. It is then the case that not enough money in the account results in a new violation. This is under the supposition that the bank has a regulation specifying that the client must not go below a certain debt level, otherwise the client is added to the bank's black list and has to pay an additional fee. We note that this latter sanction rule can never be applied when the system runs with respect to the strategy *vigilant*. However, in the case of the train scenario the result of applying either one of the strategies *vigilant* or *totalitarian* is the same.

A liberal society is implemented using the strategy:

liberal = (*test*(*act* ; *reg*) ? *fails* : *act*)* ;
(*try*(*counts-as*) ? *try*(*sanction*) : *idle*)*

This strategy imposes no restrictions on *when* normative rules are applied. One possible result could be that the agent psg1 was sanctioned because of being at the platform without a ticket. Another solution in a "liberal" agent society could be also the case that psg1 is at the platform without a ticket and without being fined. Such a scenario would never be possible when using either one of the strategies *vigilant* or *totalitarian*.

We conclude this section with a short discussion of how we can further use strategies at both theoretical and practical level. So far, we have shown how, thanks to strategies, we can explicitly implement normative artifacts. Having a classification of types of normative systems we can systematically study the expressive power for each class separately. By expressive power of a normative artifact we mean the domain of possible resulting behaviours when the multi-agent system runs under the coordination of the artifact. For example, in totalitarian societies certain correctness (in terms of safety) properties are modelled by definition. It is not difficult to see that this is no longer the case in liberal societies. Yet another point of interest is to find not only the differences but also the connections between classes. For example, we can state the following proposition:

PROPOSITION 4.1. *For any* $\langle \mathbf{A}, \sigma_b, \sigma_n \rangle$ *there exists* $\mathcal{A}^{ch}$ *such that* $totalitarian@\langle \mathbf{A}, \sigma_b, \sigma_n \rangle = liberal@\langle \mathbf{A}, \sigma_{ch}, \sigma_b, \sigma_n \rangle$ *where* σ_{ch} *is* $(cs_0, \emptyset, \nu^{\mathbf{A}} := 0)$ *and* cs_0 *is the initial state of* $\mathcal{A}^{ch}$.

which says that there exists a choreography such that totalitarian societies and liberal ones running under the directions of the choreography have the same power. This result suggests that one could see choreographies as a way to implement regimentation.

At a more practical level, we mention that the strategy language S is, in fact, implemented in the Maude system. This made it simple to experiment the use of strategies on the Maude prototype of the untimed normative multi-agent system from Figure 1. Thanks to Real-Time Maude, prototyping the timed extension of the normative (resp. choreography) language as real-time rewrite theories should be an easy exercise. However, since the strategy language S is implemented on top of Full-Maude, further effort is needed in order to use S on Real-Time Maude modules.

5. CONCLUSIONS

In this paper, we have focused on coordination in timed normative multi-agent systems. We have first described how timed agent systems can be implemented. We then presented how coordination in timed agent systems can be achieved at two orthogonal levels. On the one hand, we have shown how the *states* of a system can be enforced by means of *norms and sanctions*. The use of social and organisational concepts (e.g., norms, roles, groups, responsibility) and mechanisms for monitoring agents' actions and sanctioning has already been advocated in [11, 10, 7]. Temporal aspects of normative structures have been addressed in [2, 22]. The main difference in our approach is the "separation of concerns": actions are *untimed*, time constraints are *application-specific*. Actions have a natural definition as belief base transformers, thus having an untimed ontology of actions allows reusability. Furthermore, having time constraints *on top of actions* allows expressing synchronisations and multiple independent delays between actions.

On the other hand, we have shown how *action execution* can be enforced by means of *timed choreographies*. Related work with respect to action-based coordination artifacts appears in [21] in terms of resource access relation. The concepts of choreography and orchestration have already been introduced to web services in [18, 19]. With respect to [6], though we use the same terminology, our framework is in essence different since we deliberately ignore communication issues. Our choreography model is explicit whereas in [6] is implicit in the communication protocol. Being external, the choreography represents, in fact, contexts while in the other approaches there is a distinction between the modularity and the contextuality of the communication operator. Timed choreographies are inspired from the theory of timed automata. Timed automata has been applied to testing real-time systems specifications [15], to scheduling problems [8], and to web-services [16]. The use of timed automata in a normative multi-agent setting is new.

Besides timed coordination, we have also approached the issue of implicit nondeterminism in the operational semantics of the timed normative language. We have presented strategies as a way to handle nondeterminism at a meta-level. Such strategies we use to implement different normative artifacts for normative multi-agent systems. We did not discuss termination issues with respect to strategies. Because of "malformed" counts-as rules, e.g., recursive, the application of *vigilant* may not always terminate. It is also the case that "circularities" can lead to non-terminating *totalitarian* strategies. These aspects are subject to future work, however, more details and some examples can be found in [3].

We recall that currently, when regimentation rules are applicable, the system reaches a deadlock state. This is not an optimal exception handling mechanism. If an agent tries to do an action which leads to the application of *reg* but it can also do a permitted action a, it should not be the case that the system enters a deadlock state but it constrains the agent to execute a. We view such mechanisms as self repairing and we will formalise them in future work.

6. REFERENCES

[1] R. Alur. Timed automata. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*, LNCS, pages 8–22. Springer, 1999.

[2] A. Artikis, M. J. Sergot, and J. V. Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Log.*, 10(1), 2009.

[3] L. Astefanoaei, M. Dastani, J.-J. Meyer, and F. de Boer. On the semantics and verification of normative multi-agent systems. *Journal of Universal Computer Science (J.UCS)*, 15(13):2629–2652, 2009. `http://www.jucs.org/jucs_15_13/on_the_semantics_and`.

[4] L. Astefanoaei, M. Dastani, J.-J. C. Meyer, and F. S. de Boer. A verification framework for normative multi-agent systems. In *Proceedings of the 11th Pacific Rim International Conference on Multi-Agents (PRIMA)*, LNCS, pages 54–65. Springer, 2008.

[5] L. Astefanoaei, F. S. de Boer, and M. Dastani. The refinement of choreographed multi-agent systems. In *Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies (DALT)*. LNAI, 2009.

[6] M. Baldoni, C. Baroglio, A. K. Chopra, N. Desai, V. Patti, and M. P. Singh. Choice, interoperability, and conformance in interaction protocols and service choreographies. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 843–850. IFAAMAS, 2009.

[7] G. Boella and L. van der Torre. Substantive and procedural norms in normative multiagent systems. *J. Applied Logic*, 6(2):152–171, 2008.

[8] P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

[9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.

[10] M. Dastani, D. Grossi, J.-J. C. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *Proceedings of the Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS)*, 2008.

[11] V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Utrecht University, 2003.

[12] S. Eker, N. Martí-Oliet, J. Meseguer, and A. Verdejo. Deduction, strategies, and rewriting. *ENTCS*, 174(11):3–25, 2007.

[13] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker and its implementation. In *Model Checking Software: Proc. 10 th Intl. SPIN Workshop*, volume 2648 of *LNCS*, pages 230–234. Springer, 2003.

[14] D. Grossi, F. Dignum, and J.-J. C. Meyer. A formal road from institutional norms to organizational structures. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, page 89. IFAAMAS, 2007.

[15] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. Testing real-time systems using UPPAAL. In *Formal Methods and Testing*, LNCS, pages 77–117. Springer, 2008.

[16] A. Lomuscio, W. Penczek, M. Solanki, and M. Szreter. Runtime monitoring of contract regulated web services. In *Proceedings of the 12th International Workshop on Concurrency, Specification and Programming (CS&P09)*, 2009. to appear.

[17] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE*, 9:39–47, 1960.

[18] S. Meng and F. Arbab. Web services choreography and orchestration in Reo and constraint automata. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 346–353. ACM, 2007.

[19] J. Misra. A programming model for the orchestration of web services. In *SEFM*, pages 2–11. IEEE, 2004.

[20] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.

[21] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: the A&A approach for engineering working environments in mas. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, page 150. IFAAMAS, 2007.

[22] F. Viganò, N. Fornara, and M. Colombetti. An event driven approach to norms in artificial institutions. In *Proceedings of the International Workshop on Agents, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM)*, LNCS, pages 142–154. Springer, 2005.