Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1481–1490 DOI: 10.15439/2016F549 ACSIS, Vol. 8. ISSN 2300-5963

Run-time Injection of Norms in Simulated Smart Environments

Patrizia Ribino, Carmelo Lodato, Antonella Cavaleri, Massimo Cossentino
Istituto di Reti e Calcolo ad Alte Prestazioni
Consiglio Nazionale delle Ricerche
Palermo, Italy
Email: {ribino, c.lodato, a.cavaleri, cossentino}@pa.icar.cnr.it

Abstract—**Smart systems have to deal with environmental changes and react for adapting their behavior to changes in the operating conditions, so to always meet users' expectations. This is fundamental for those systems operating in open environments that may change frequently. Smart environments are complex systems that more than others are affected by these issues. In this paper, we propose a normative framework for regulating at run-time system behavior when some situations occur, thus providing system flexibility. The proposed approach includes also mechanisms to identify anomalous situations that can occur in the system due to the run-time injection of new norms.**

I. Introduction

A GREAT challenge in complex systems is to adequately deal with the unpredictability and the dynamics changing of the application context the systems are plugged in. Smart environments are complex systems that more than others are affected by these issues. A way to provide system flexibility is given by implementing statically normative frameworks in which norms for regulating the behavior of the system are specified at design time. This kind of solution is not effective when we face with unpredictable and unexpected situations that have not been considered at design time. Hence, solutions for modifying at run-time norms or injecting new ones into the systems are mandatory.

It is widely accepted that multi-agent systems provide relevant features for implementing smart environments [1][2][3][4]. In such field, norms have been widely employed for regulating the ideal behavior of the system. Norms are considered as a mechanism for controlling multi-agent societies and for ensuring order and predictability [5]. Such norms define standards of behaviors that have to be adopted in the society as well as undesired behaviors that have to be avoided. Hence, normative frameworks rely on a representation of norms by means of permission, obligations and prohibitions that ensure the agents behave within predefined boundaries. In particular, several works have been conducted for addressing theoretical and practical aspects about norm change [6][7][8] providing agents with mechanisms for enacting behavior modification. Typically, new plans/actions have been created for agents to be complied with new norms.

In this paper, we propose a normative framework to be coupled with goal oriented systems in order to introduce more flexibility in smart environment by means of run-time injection of norms regulating system goal fulfillment. In our approach, norms and goals allow to cope with two main aspects of a smart environment. Goals express what is the desired state of the world the system has to result in. Norms regulate the desired state of the world according to the normative context in which the system works.

In order to modify system behaviors when new norms are injected into the system, we implemented some algorithms in agents life cycle for reasoning about new goals to be pursued according to new norms. In so doing, we also manage some anomalous situations that can occur in the system due to the run-time injection of new norms. In particular, we identified three kinds of anomalies:

- the injection of an inconsistent norm, namely containing logical contradictions;
- the injection of a norm that is structurally incompatible with a goal;
- finally, the injection of a norm that creates an antinomy.

Thus, the main contributions of this work are:

- an algorithm for modifying agent goal fulfillment according to norm changing;
- an algorithm for checking the consistency of norms with the system.

In order to show in semi-natural languages some practical examples, we use GoalSpec [9] and SBVR [10] for modeling goals and norms.

Moreover, the normative framework with algorithms was implemented in MUSA [11]. It is a Middleware for User-driven Service Adaptation that provides means for supporting run-time adaptation of a process together with a multi-agent system for executing the activities of the process.

In order to validate our approach, we simulated a smart environment for improving services of a health care home. The simulation has been developed by integrating MUSA with ICasa simulator [12].

The rest of the paper is organized as follows. Section II introduces the theoretical background of the paper. In Sections III and IV, related works and motivation are respectively presented. Section V and Section VI presents the key concepts and the algorithms the proposed normative framework is based on. Section VII illustrates an application scenario of the proposed approach in a simulated smart environment. Finally, in Section VIII conclusions are drawn.

II. Background

The main purpose of this work is to introduce more flexibility in smart environments by means of run-time injection of norms for adapting the behavior of the system to new situations. This section introduces the theoretical background of the paper. In particular, it is organized in four parts: the first one presents the features making multi-agent systems more suitable for implementing smart environments; the second one introduces two modeling languages (GoalSpec [9] and SBVR [10]) we use for modeling goals and norms for specifying functionalities and setting regulations the smart environment has to own. The third one provides an overview of MUSA [11], a multi-agent system for the dynamic composition and the orchestration of services in a distributed and open environment we adopt for implementing the functionality of the smart environment. The last one introduces ICasa [12], a dynamic pervasive environment simulator, we use for simulating the injection of norms in a smart environment.

A. Multi-Agent System features for Smart Environment

A Multi-Agent System (MAS) is a distributed system composed of autonomous entities, called agents. The decentralized and loosely coupled nature of such kind of systems makes it possible to design applications that are highly flexible, scalable and adaptive. The multi-agent paradigm provides several features that make them more suitable in order to fit smart environment requirements. Among them:

- *Decentralization* - MASs provide decentralized control based on distributed autonomous entities.
- *Interactions* - MASs support complex interactions between entities, using high level semantic languages. It is essential for smart environments that commonly deal with various, heterogeneous information from physical sensors, services or users preferences.
- *Coordination* - In a MAS, individual entities with limited capabilities are able to coordinate in order to achieve complex tasks. Flexible organization patterns enable groups of agents to create and dynamically reconfigure applications depending on current conditions. In an open, and dynamic smart environment, this feature is highly suitable for dynamic composition of elementary functionality in order to accomplish more sophisticated processes.
- *Heterogeneity* - A MAS is a society of agents with different capacities and roles.

B. GoalSpec & SBVR

GoalSPEC [9] is a language designed for specifying user-goals and enabling at the same time goal injection and software agent reasoning. The concept of goal is central in GoalSpec. Goals are described as states of the world that the user desires to achieve [13]. In GoalSpec, a goal is composed of a *Trigger Condition* and a *Final State*. The trigger condition is an event that must occur in order to start acting for addressing the goal. The final state is the desired state of the world that must be addressed. Trigger conditions and final states must be expressed by using domain ontology predicates. In GoalSpec, uppercase words represent the keywords of the language, and lowercase words represent the predicates constrained by the domain ontology.

Let us suppose we want define in our smart environment system the following common user goal:

If at 07:00 the living-room temperature is below 20 C, the shutter should be closed and the air conditioning turned on at level 6.

In GoalSpec, it could be written as follow:

WHEN on(07:00 pm) AND temperature(T) AND T<20 THE system SHALL ADDRESS closed(shutter) AND turned_on(air_conditioning, level(6)).

In order to define norms in natural language, we adopted the SBVR [10] standard as a modeling language for our system.

The Semantics of Business Vocabulary and Business Rules (SBVR) is an adopted standard of the Object Management Group (OMG). It is designed for business domains for formalizing complex business rules. In business domains, a business rule describes the conditions of a business process execution. A business rule may define the semantics of business concepts, reactions to business events, constraints and preconditions on tasks and activities, as well as the prohibitions, permissions and obligations of business actors and activities. In other words, business rules guide and constrain various aspects of business, including the sequence and timing of activities [14]. SBVR uses 'semantic formulation', which is a way of describing the semantic structure of statements and definitions.

In our approach, we use GoalSPEC for defining expected results of the system in terms of functionality and we adopt SBVR for specifying the normative context the system works in. Hence, by using the same kind of abstraction of SBVR, we can see a smart environment as an organization of entities that are involved in activities for reaching goals. The entities of the smart environment, sharing the same domain ontology (the SBVR vocabulary), constitute a community's body of shared meanings that can understand SBVR rules.
An example of SBVR rules for the previously defined goal could be: *It is prohibited that the system closes shutters if John is at home.*

C. MUSA- A Middleware for User-driven Service Adaptation

The Middleware for User-driven Service Adaptation (MUSA) proposed in [11] is intended to provide a means for supporting run-time adaptation of a process based on a multi agent system for executing the activities of the process. The core element of the approach is the use of Goals for explicitly representing user-preferences into the system (what to address). The injection of goals triggers the re-organization of the agents in hierarchical groups. These self-adaptive structures allow for dealing with dynamic composition and orchestration of services. MUSA provides a platform in which *(i)* it is possible to deploy some capabilities that wrap real services, completing them with a semantic layer for their smart use; *(ii)* users can inject their goals for satisfying their specific needs. Under the hypothesis that both goals and capabilities refer to the same semantic layer (described as an ontology), then the agents of the system are able to conduct a proactive means-end reasoning for composing available capabilities into tasks for addressing the user request.

D. iCASA - a dynamic pervasive environment simulator

iCasa [12] is a set of integrated tools for the development and administration of pervasive applications. It includes a smart home simulator that allows the creation and removal of a wide range of devices that can be used by the applications. Specifically, iCasa Simulator provides:

- A graphical user interface that displays a map of the house and the localization of the different devices. It allows developers to create and configure devices, create and move physical users, and watch their actual configurations;
- Scripting facilities for controlling the environment and to test the applications under reproducible conditions.
- Notification facilities for notifying users of any modifications in the environment.

III. Related Works

Norms like obligations, permissions and prohibitions have been implemented in multi-agent systems in order to specify (un)desired behavior of agents so that the goals of the system can be reached. They also provide means for coordinating agent activities in order to reach the overall objective of the system they are part of [15]. Norm-governed systems are also known as Normative systems.

Normative systems are commonly defined as systems that specify every possible system transition, whether or not that transition is considered to be legal or not. In other words, Normative Systems specify which actions or which states should be achieved or avoided [16][17][18].

A lot of work has been done about normative frameworks in the field of Electronic Institutions or Virtual Organizations where norms have found a natural implementation. For instance, Alechina *et.al* [19] present a programming framework for developing normative organizations. Such framework is based on N-2APL, a BDI-based agent programming language for implementing norm-aware agents. N-2APL supports normative concepts such as obligations, prohibitions and sanctions. In such a work, the normative system is conceived in such away that the interaction between agents and the environment is regulated by a "normative exogenous organizations", which is defined by means of a set of conditional norms (i.e: conditional obligation and conditional prohibition). Such norms have the form *obligation(l, o, d, s)* that means "*agent l is obliged to establish an environment state satisfying o before deadline d, otherwise it will be sanctioned by updating the environment with s*"[19]. A norm-aware deliberation approach is also proposed. It allows agents to determine the set of plans (to be adopted in order to satisfy a goal) of highest priority which do not violate higher priority prohibitions.

In [20] Kollingbaum and Norman proposed the NoA Normative Agent Architecture. It supports the implementation of norm-governed practical reasoning agents. NoA agents are motivated by norms to act. In the NoA language, all the effects of a plan are declared in a plan specification. These effects are considered by agents for reasoning about plan selection and execution. Moreover, the norms governing the behavior of a NoA agent refer to either actions that are obligatory, permitted, forbidden, or states of affairs that are obligatory, permitted or forbidden. The NoA language enables an agent to be programmed in terms of plans and norms. Normative statements formulated in the NoA language express obligations, permissions and prohibitions of an agent: *Obligations* motivate the agent to achieve either a state of affairs or to perform a specific action. Prohibitions require the agent to not achieve a state of affairs or to not perform an action. The agent is forbidden to pursue a specific activity. Prohibitions represent restrictions on what capabilities the agent is allowed to reach a certain goal. Finally, Permissions allow the achievement of a state of affairs or the performance of an action.

Some works have been conducted for addressing norm change and norm consistency providing agents with mechanisms for enacting behavior modification. Typically, new plans/actions have been created to comply with new norms [8][21][22]. In [23], Jiang *et.al* propose a normative structure, named *Norm Nets* (NNs) for modeling sets of interrelated regulations. NNs aims at verifying whether executions of business process are compliance with process regulations. Authors define a norm as a tuple of elements that specify the type of deontic operator, the pair role-action (the target) to which the deontic modality is assigned, a deadline of norm validity and a precondition that determines when the target is initiated. A formal method for checking norm compliance by using Colored Petri Nets is proposed. In [24] [25], authors propose a means for automatically detecting and solving conflict and inconsistency in norm-regulated Virtual Organization and Electronic Institution.

In the next section, we provide the reasons that motivate our work with respect to the related works.

IV. Motivation

Although a considerable literature exists about norms, it is mainly directed to explore the role of norms inside Virtual Organization and Electronic Institution that are tightly coupled with agents. We take inspiration from those works and we are trying to introduce norms in smart systems as mechanisms that regulate the system at a higher level of abstraction than that where agents work.

The normative framework we defined adopts a norm specification similar to the previous ones. But in our framework, we employ norms at a higher level of abstraction by moving them from activity's regulations to goal's regulations. This choice is motivated by the context of self-adaptive and self-organized systems (SASO) we are working on. Commonly this kind of systems are able to effectively adapt their behavior to environment changes and self-organize their internal structure for finding composed solutions in order to achieve collaborative goals. In particular, the kind of SASO systems we are considering owns the following features [11]:

- *Openness* - They are open systems that evolve at run-time because: *(i)* new services could be made available for satisfying user requirements; *(ii)* the satisfaction of new user requirements may be demanded to the system;
- *Goal-directed* - They are goal-directed systems. Goals are motivators for these systems providing them the reason for doing something. Goals express user requirements to be satisfied.

The aim of this work is to increase the openness of the system by allowing the run-time introduction of new regulations thus improving the flexibility of such systems. In so doing, we look at norms from a different perspective with respect to the classical one. Starting from the consideration that goals are key elements for the systems we focus on and they are motivators of their behavior, we look at goals as commitments the system engages with users. In other word, goals can be seen as a particular kind of obligation that has to be satisfied when some conditions occur.

Hence, in the normative framework we developed norms that are directly linked to goals where a permission norm relaxes the conditions under which the goal has to be satisfied and a prohibition norm nullifies the commitment under the circumstances expressed by the prohibition. The effect is that norms may act for increasing the opportunity for the system to pursue the goal it is committed to (Permission) or, on the contrary, norms may inhibit system intentions to pursue the goal it is committed to (Prohibition).

By adopting this perspective, the norms operate for regulating *what* the system has to satisfy and not *how* it does that.

In order to provide an example, let us to suppose there is a system that is committed for satisfying the user goal *have lunch* and at the moment of the commitment, the system owns two means to satisfy the *have lunch* goal. Fig. 1 shows a goal diagram where the goal "*have lunch*" can be satisfied by performing the tasks "*book a restaurant*" or "*take a pizza*". Let us suppose that such smart system has to provide its assistance to a diabetic patient. A norm in the system states that "*It is prohibited that a diabetic patient has lunch before taking insulin*". The norm (at the goal level) constraints the requirement that the system can provide by inhibiting its intentions rather than disabling all the possible ways the system can follow in order to satisfy that requirement.

Let us suppose now that a new task "*cook with microwave oven*" is introduced at run time for satisfying the *"have lunch"* goal (see Fig.2). Considering that the norm at the goal level spreads to the task level, we do not need to add/change any system regulations to adapt the behavior of the system to manage the change of its operative context.

Moreover let us suppose that a new norm for the goal "*have lunch*" is injected at run-time in the system. The simultaneous presence of interrelated norms may cause some system conflict or inconsistency. Indeed, when two norms are interrelated, it may happen that being compliant with a norm may cause to be uncompliant with the second one, thus generating conflicts or inconsistencies. In classical approaches, conflicts are generated when an agent wants to perform an action that is simultaneously allowed and forbidden. Inconsistencies, instead, occur when an agent may be forbidden to perform an action that may be essential for fulfilling one of its obligations [26].

In our approach, we characterize the definition of conflict and inconsistency to deal with dynamically changing environments, where the conflicting state of two norms may change according to the particular execution context. In order to address this concern, we introduce some new definitions about conflicts and inconsistencies that are based on a representation of the execution context.

Resuming the previous example, let us suppose that a new norm regulating the "*have lunch*" goal is introduced into the system, that is "*It is permitted that guests have lunch if they have performed sports activities*". This latter norm is interrelated to the previous one because they both refer to the same goal "*have lunch*". The injection of the second norm could cause a system deadlock because if the conditions of both norms are simultaneously valid an antinomy is generated and the system does not know how to behave.

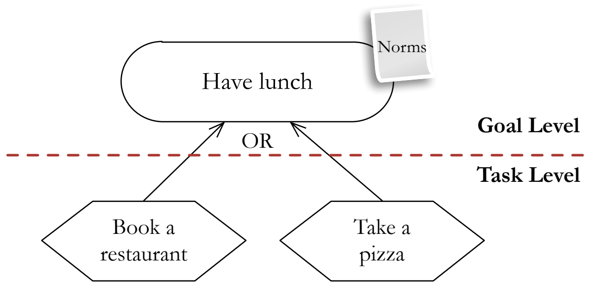


Fig. 1: An example of Means-End Analysis [27]. It introduces two tasks "*Book a restaurant*" and "*Take a pizza*" to indicate two particular ways to fulfill the goal "*Have lunch*".

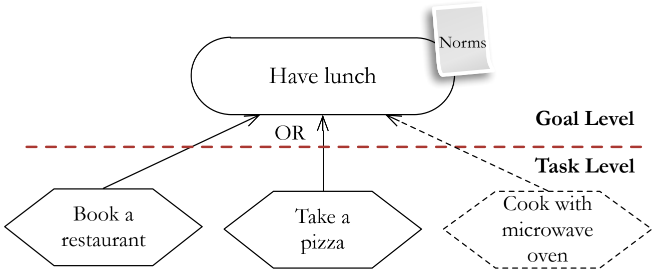


Fig. 2: The availability of the new task "*cook with microwave oven*" gives the system a new mean for satisfying the goal "*Have lunch*".

In the following section, we introduce the key concepts our approach is based on.

V. Theoretical Foundations

The normative framework for introducing more flexibility in smart environments, we propose in this work is based on three key concepts: state of the world, goal and norm. In the following, we formally introduce some definitions.

▼ Definition 1 — *State of the world*
Let $\mathcal{D}$ be the set of concepts defining a business domain. Let $\mathcal{L}$ be a first-order logic defined on $\mathcal{D}$ with $\top$ a tautology and $\bot$ a logical contradiction, where an atomic formula $p(t_1, t_2 ..., t_n) \in \mathcal{L}$ is represented by a predicate applied to a tuple of terms $(t_1, t_2 ..., t_n) \in \mathcal{D}$ and the predicate is a property of or relation between such terms that can be true or false.

A *state of the world* in a given time t ($\mathcal{W}^t$) is a subset of atomic formulae whose values are true at the time t:

$$\mathcal{W}^t = [p_1(t_1, t_2, ..., t_h), ..., p_n(t_1, t_2, ..., t_m)]$$

The *state of the world* represents a set of declarative information concerning events occurred within the environment and relations among events at a specific time. An event can be defined as the occurrence of some fact that can be perceived by or be communicated to the smart system. Events can be used to represent any information that can characterize the situation of an interacting user as well as a set of circumstances in which the smart system operates at a specific time. Definition 1 is based on the close world hypothesis[28] that assumes all facts that are not in the state of the world are considered false.

▼ Definition 2 — *Goal*
Let $\mathcal{D}$, $\mathcal{L}$ and $p(t_1, t_2 ..., t_n) \in \mathcal{L}$ be as previously introduced in the definition 1. Let $t_c \in \mathcal{L}$ and $f_s \in \mathcal{L}$ be formulae that may be composed of atomic formulae by means of logic connectives AND($\wedge$), OR ($\vee$) and NOT ($\neg$).

A *Goal* is a pair $\langle t_c, f_s \rangle$ where t_c (*trigger condition*) is a condition to evaluate over a state of the world $\mathcal{W}^t$ when the goal may be actively pursued and f_s (*final state*) is a condition to evaluate over a state of the world $W^{t+\Delta t}$ when it is eventually addressed:

- *a goal is active iff* $t_c(\mathcal{W}^t) \wedge \neg f_s(\mathcal{W}^t) = true$
- *a goal is addressed iff* $f_s(\mathcal{W}^{t+\Delta t}) = true$

Goals express what is the desired state of the world the system has to result in. Conversely, *Norms* regulate the desired state of the world according to the normative context in which the system works.

▼ Definition 3 — *Norm*
Let $\mathcal{D}$, $\mathcal{L}$, $p(t_1, t_2 ..., t_n) \in \mathcal{L}$ and $\mathcal{W}^t$ be as previously introduced in the definition 1. Let $\phi \in \mathcal{L}$ and $\rho \in \mathcal{L}$ be formulae composed of atomic formula by means of logic connectives AND($\wedge$), OR ($\vee$) and NOT ($\neg$). Moreover, let $D_{op} = \{permission, obligation, prohibition\}$ be the set of deontic operators. A *Norm* is defined by the elements of the following tuple:

$$n = \langle r, g, \rho, \phi, d \rangle$$

where

- $r \in \mathcal{R}$ is the *Role* the norm refers to. The special character "_" indicates that the norm refers any role.
- $g \in \mathcal{G}$ is the *Goal* the norm refers to. The special character "_" indicates that the norm refers to any goal.
- $\rho \in \mathcal{L}$ is a formula expressing the set of actions and state of affairs that the norm disciplines.
- $\phi \in \mathcal{L}$ is a logic condition (to evaluate over a state of the world $\mathcal{W}^t$) under which the norm is applicable;
- $d \in D_{op}$ is the deontic operator applied to ρ that the norm prescribes to the couple $(r, g) \in \mathcal{R} \times \mathcal{G}$.

$$\text{In particular } d(\rho) = \begin{cases} \rho & \text{iff } d = \textit{obligation} \\ \neg\rho, & \text{iff } d = \textit{prohibition} \\ \rho \vee \neg\rho & \text{iff } d = \textit{permission} \end{cases}$$

In other words, let $\mathcal{W}^t$ be a state of the world, a norm prescribes to a couple (r, g) the deontic operator d applied to ρ if ϕ is true in $\mathcal{W}^t$ [1].

▼ Definition 4 — *State of Norm*
Let a norm $n = \langle r, g, \rho, \phi, d \rangle$ where $g = \langle t_c, f_s \rangle$ and let a state of the world in a given time t ($\mathcal{W}^t$)

A norm can assume the following states:

- n is *applicable at time t* if $\phi(\mathcal{W}^t) = true \vee \phi = \top$
- n is *active at time t* if n is applicable and $t_c(\mathcal{W}^t) = true$
- n is *logically contradictory* if ϕ *is* $\bot$
- n is *in opposition to goal* if $f_s \wedge d(\rho)$ *is* $\bot$

Moreover, let a state of the world (W^t) and let two norms $n_1 = \langle r_1, g_1, \rho_1, \phi_1, d_1 \rangle$ and $n_2 = \langle r_2, g_2, \rho_2, \phi_2, d_2 \rangle$ where $r_1 = r_2$, $g_1 = g_2$, $\rho_1 = \rho_2$

- n_1 and n_2 are *deontically contradictory* iff

$$\begin{cases} \phi_1(\mathcal{W}^t) \wedge \phi_2(\mathcal{W}^t) = true \\ d_1 \neq d_2 \end{cases}$$

It is worth noting that we talk about *logically contradictory* when the contradiction concerns the logical conditions ($\phi \in \mathcal{L}$) under which the norms are applicable. On the contrary, we talk about *deontically contradictory* when the contradiction concerns the semantic meaning of the deontic operator ($d \in D_{op}$) the norms apply to.

Moreover, a norm is in opposition to a goal when pursuing that goal violates always the prescribed norm.

In the next section, we present the algorithms for the injection of norms in a running smart environment.

VI. Algorithms for run-time injections

The aim of the normative framework is to provide some mechanisms that allow to modify the behavior of the smart environment in order to adapt it to unexpected situations that can occur by introducing new norms. The approach we propose is illustrated by the Algorithm 1.

The triple of elements the Algorithm 1 works on is composed of: a *state of the world* $\mathcal{W}^t$ that characterizes the system in a given time, a *set of goal* $\mathcal{G}$ representing the requirements the system is able to satisfy and finally a *set*

[1] It is worth noting that in order to be compliant in $\mathcal{W}^t$ with 1) an obligation ρ must be true, 2) a prohibition $\neg\rho$ must be true 3) a permission ρ or $\neg\rho$ may be true. In the context of this paper, we assume that the system does not violate norms.

Algorithm 1: RunTime injection

```
Data: $\mathcal{W}^t$, $\mathcal{G}$, $\mathcal{N}$
while $system\ is\ running$ do
    $\mathcal{N}^t_{injected} \leftarrow inject\_new\_norms$;
    ①for $j \leftarrow 1$ to $length(\mathcal{N}^t_{injected})$ do
        $\langle r, g, \rho, \phi, d\rangle \leftarrow n_j$;
        $\langle t_c, f_s\rangle \leftarrow g$;
        if $(\phi \neq \bot) \wedge (f_s \wedge d(\rho) \neq \bot)$ then
            add $\langle r, g, \rho, \phi, d\rangle$ to $\mathcal{N}$;
        else
            $revise(n_j)$
    ②foreach $g_i \in \mathcal{G}$ do
        $\langle t_{ci}, f_{si}\rangle \leftarrow g_i$;
        if $\neg f_{si}(\mathcal{W}^t)$ then
            $\mathcal{N}_i \leftarrow \{n \in \mathcal{N} : n = \langle r, g_i, \rho, \phi, d\rangle \wedge \phi(\mathcal{W}^t) = true\}$;
            Ⓐ if $card\{\mathcal{N}_i\} = 0 \wedge t_{ci}(\mathcal{W}^t) = true$;
            then
                $pursue(g_i)$;
            Ⓑ if $card\{\mathcal{N}_i\} = 1$;
            then
                $\langle r, g_i, \rho, \phi, d\rangle \leftarrow n$;
                if $(d = Permission)$ then
                    $pursue(g_i)$;
            Ⓒ if $card\{\mathcal{N}_i\} > 1$;
            then
                $\mathcal{N}_i \leftarrow Check\_Norms(\mathcal{N}_i, \mathcal{W}^t)$; (see Alg.3)
                $(\phi_{OR}, \phi_{AND}) \leftarrow Compose\_Norm\_Condition(\mathcal{N}_i)$; (see Alg.2)
                $t'_{ci} \leftarrow OR\_composition(t_{ci}, \phi_{OR})$;
                $t'_{ci} \leftarrow AND\_composition(t'_{ci}, \phi_{AND})$;
                $g'_i \leftarrow \langle t'_{ci}, f_{si}\rangle$;
                if $t'_{ci}(\mathcal{W}^t) = true$ then
                    $pursue(g_i)$;
```

of norms $\mathcal{N}$ the system has to obey in order to deal with some specific situations. Both $\mathcal{N}$ and $\mathcal{W}^t$ may change during system execution. In particular, the state of the world may change due to some events that can occur or some actions that can be performed in the environment. The set of norms may change due to norm injection.

While the system is running, new norms can be injected. Step ① makes a preliminary check on new injected norms. Such step ensures that among injecting norms neither norms are in opposition to the goal they refer nor they are logically contradictory.

Step ② is the core of the algorithm. The system is in the state of the world ($\mathcal{W}^t$), the norms have effects on the system goals only if they are not addressed yet. Thus, for each goal the system has to satisfy, the set of applicable norms ($\phi(\mathcal{W}^t) = true$) is processed.

Hence, three situations can occur. The most simple one (Ⓐ) is that there are no applicable norms for an active goal (see *Definition 2*). In such a case the system can pursue the goal without restrictions. The second situation (Ⓑ) is a basic case in which the set of applicable norms for a single goal is composed of only one norm. In such case, *(i)* if the norm is a permission it actives the related goal and the system can fulfill that goal despite $t_c(\mathcal{W}^t) = false$. This is because the permissions relax system constraints, giving alternatives; *(ii)* if the goal is regulated by a prohibition, it further constraints the goal activation. The system cannot pursue that goal until $\phi(\mathcal{W}^t) = true$. It is worth noting that generally speaking, an applicable norm influences a goal when it is active. However, a permission norm can influence a goal also when it is inactive.

The last situation (Ⓒ) is a general case in which norms are more than one and they can have different deontic operators. In this case Algorithm 1 allows to modify goals, making them norm compliant. By encapsulating the condition expressed by the norms inside the goal they refer to, it is possible to modify the activation of that goal thus making it compliant with the norms. Such composition (see Algorithm 2) takes into consideration different types of norms and it accordingly modifies the activation of a goal.

Algorithm 2: Compose_Norm_Condition

```
Data: a list of norms $NormList$
Result: a couple $(\phi_{mergedOR}, \phi_{mergedAND})$
$List\phi\_OR \leftarrow \varnothing$;
$List\phi\_AND \leftarrow \varnothing$;
// Identification of norm types
for $j \leftarrow 1$ to $size(NormList)$ do
    $\langle r, g, \rho, \phi, d\rangle \leftarrow NormList[j]$;
    switch $d$ do
        case Obligation do
            break;
        case Prohibition do
            add $\neg\phi$ to $List\phi\_AND$;
        case Permission do
            add $\phi$ to $List\phi\_OR$;
// Permissions give alternatives (OR)
if $Size(List\phi\_OR) \neq 0$ then
    $\phi_{mergedOR} \leftarrow List\phi\_OR[1]$;
    for $h \leftarrow 2$ to $Size(List\phi\_OR)$ do
        $\phi_{mergedOR} \leftarrow OR\_composition(\phi_{mergedOR}, List\phi\_OR[h])$;
// Prohibition are mandatory (AND)
if $Size(List\phi\_AND) \neq 0$ then
    $\phi_{mergedAND} \leftarrow List\phi\_AND[1]$;
    for $h \leftarrow 2$ to $Size(List\phi\_AND)$ do
        $\phi_{mergedAND} \leftarrow AND\_composition(\phi_{mergedAND}, List\phi\_AND[h])$;
```

It is worth noting that, when there are more than one applicable norm in the system (Ⓒ), it is necessary to check for deontological contradictions among norms (see Definition 4) and to remove them. This is performed by Algorithm 3. Deontological contradictions are known in legislative environments as *antinomy*. For instance, if there is an applicable norm *n1* that prohibits to pursue a goal *g1* and another applicable norm *n2* that obliges to pursue the same goal *g1*, then *n1* and *n2* generate an antinomy.

In legal theory, several criteria exist for solving such antinomy [29]: *legis posterior*, the most recent norms takes precedence; *legis superior* the norm imposed by the strongest institutional power takes precedence; and *legis specialis* the most specific norm takes precedence. In this paper, we assume to work with hierarchically equal norms (i.e: norms with the same authority) thus we adopt the *legis posterior* criterion in order to choice among conflicting norms.

In the following, we show our approach in a simulated smart environment.

Algorithm 3: Check_Norms

```
Data: a list of applicable norms $\mathcal{N}$ related to a single goal, a
      state of the world $\mathcal{W}^t$
Result: a list $\mathcal{N}_{out}$ of consistent norms
$\mathcal{N}_{out} \leftarrow chronological\_order(\mathcal{N})$;
$\mathcal{M}_{conflicts} \leftarrow \varnothing$;
for $i \leftarrow 1$ to $length(\mathcal{N}_{out})$ do
    $\langle r, g, \rho, \phi_i, d_i \rangle \leftarrow n_i$;
    for $j \leftarrow i+1$ to $length(\mathcal{N}_{out})$ do
        $\mathcal{M}_{conflicts}[i][i] \leftarrow 1$;
        if $r_i = r_j \wedge \rho_i = \rho_j \wedge d_i \neq d_j$ then
            $\mathcal{M}_{conflicts}[i][j] \leftarrow 1$;
        else
            $\mathcal{M}_{conflicts}[i][j] \leftarrow 0$;
for $i \leftarrow length(\mathcal{N}_{out})$ to $1$ do
    $n_{current} \leftarrow \mathcal{N}_{out}[i]$;
    for $j \leftarrow i-1$ to $1$ do
        if $\mathcal{M}_{conflicts}[i][j] = 1$ then
            $delete(\mathcal{N}_{out}, oldest(\mathcal{N}_{out}[j], n_{current}))$;
            $n_{current} \leftarrow \mathcal{N}_{out}[j]$;
```

VII. A Simulated Health-care Home

In order to show our approach, we simulated an health-care home provided with a smart system for supporting guest activities.

The simulation framework is responsible for time advancing. In particular, we adopt a discrete time-stepped simulation. The virtual time advances with a fixed interval (i.e.: each time step is 30 minutes).

In the simulated health-care home, each guest is provided with some devices for individual recognition and with wearable sensors for monitoring physiological parameters (i.e: body temperature, heart rate, blood pressure etc...). The smart environment owns a knowledge base containing information about health-care home guests (i.e: diseases, pharmacological therapy, clinical investigations etc...). The health-care home is endowed with a plurality of sensors and devices. The smart system provides each guest with a personal virtual tutor that supports daily activities such as taking medicines, doing medical examinations and so on.

Fig. 3 shows a screen-shot of health-care house simulated with ICasa[12]. It is composed of several bedrooms for guests,

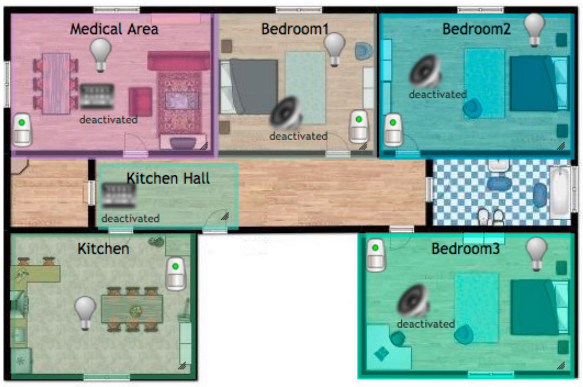


Fig. 3: The simulated health-care house

a medical room and a restaurant. Each room is endowed with a presence sensor that detects the occupancy of a space by people. Some monitors are located in common areas in order to show personalized advertisements to guests.

An excerpt of user goals the system can satisfy is described in the following by adopting the GoalSpec specification.

goal_1: *WHEN is_Time_to_WakeUp(guest) THE system SHALL ADDRESS light(guest_room, on) AND alarm(guest_room, on) AND guest(awake).*

goal_2: *WHEN is_Time_to_Have_Lunch(guest) THE system SHALL ADDRESS restaurant_service(guest, available).*

goal 3: *WHEN is_Time_to_Take_Medicine(guest) THE system SHALL ADDRESS at(guest, medical_room) AND done(took_medicine).*

...

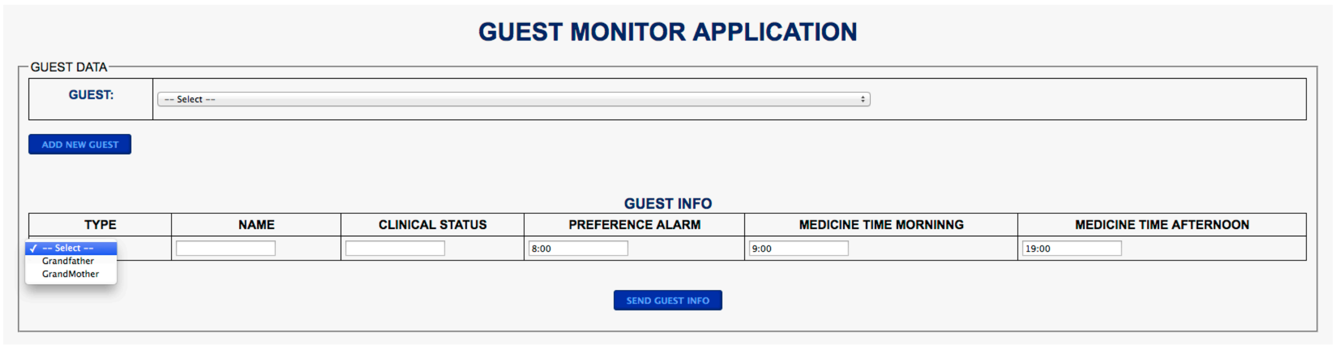
goal n: *WHEN is_Time_to_Meet_Doctor THE system SHALL ADDRESS done(checked, guest)*

The first goal means that the Smart Environment has to reach the desired state of the world in which the light and the alarm of guest's room are turned on and guest is awake. The second one makes the restaurant available to guests. The third one indicated that the smart environment has to fulfill the state of the world in which the guest is at medical room and he takes his medicine. Finally, the system allows to periodically make medical checks for monitoring the wellness of guests.

For the sake of clarity, in the following scenarios we exemplify norms in SBVR language. It is worth noting that our application translates SBVR rule according to Definition 3. In the following we provide an example:

SBVR norm: *It is permitted that guests have lunch if they have performed sports activities*

System norm: *norm(type(permission), role(guest), goal (have_launch), condition(is(guest,diabetic))).*



GUEST MONITOR APPLICATION

GUEST DATA

GUEST: -- Select --

ADD NEW GUEST

GUEST INFO

TYPE	NAME	CLINICAL STATUS	PREFERENCE ALARM	MEDICINE TIME MORNINNG	MEDICINE TIME AFTERNOON
✓ -- Select -- / Grandfather / GrandMother			8:00	9:00	19:00

SEND GUEST INFO

Fig. 4: Guest Monitor Application

We initially assume that the guests of the health-care home are individuals without particular diseases. The system behaves in the same way for each guest thus satisfying the previous goals. Over time, other people with particular illnesses are received and their preferences and clinical status are registered in the system. The manager of the health-care home inserts new norms in the system in order to change system behavior accordingly to the new situations. In particular, we simulated the following scenarios.

a) Permission Norm Injection: During the normal execution of the system, the manager of the health-care home gives the permission to guests that get already had some sports activities to have lunch before the established time. Thus, he introduces into the system the following norm:

norm_1: *It is permitted that guests have lunch if they have performed sports activities.*

In such case, the system differently behaves according to the particular guest. It allows only sportive guests to go to the restaurant before the regular time for lunch (i.e.: 13 o'clock). In Fig.5 two guests (Paul and Maria) want to go to the restaurant at midday. In such scenario, Paul went to jogging in the morning. Thus, the system permits Paul to go to the restaurant before the regular time for lunch. Conversely, it is prohibited for Maria that has not performed any sport activities.

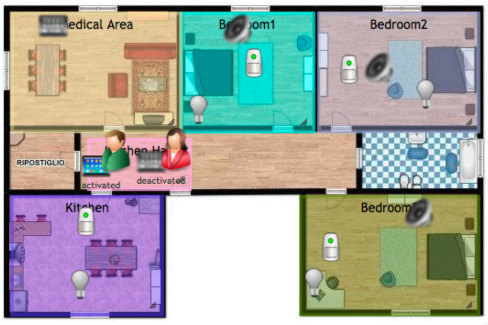


Fig. 5: The system behaves differently for different users.

b) Diabetic Guest: A new guest (Mark) is received in the health-care home. It is the first diabetic patient of the house. Fig.4 shows a screen-shot of the simulation framework that allows to introduce new guest into the simulated environment along with some guest information and preferences. The following norm is injected at run-time into the system:

norm_2: *It is prohibited that a guest has lunch before taking insulin if the guest is diabetic.*

In such scenario, the diabetic guest goes to the restaurant at 13 o'clock but before taking insulin. In such case, the system does not allow Mark to eat at the restaurant and its virtual tutor advises him that has to take the medicine before lunch (see Fig.6 (a)). Then, Mark goes to the medical area and he takes insulin (see Fig.6 (b)). Thus, the system updating the state of the world will permit Mark to have lunch.

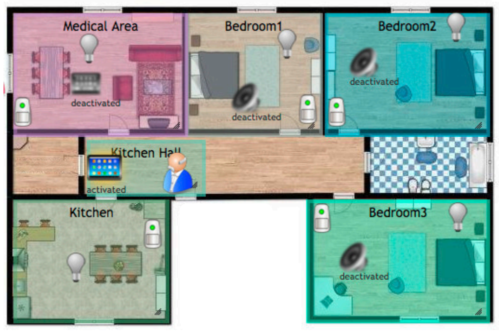
c) Heart-Rate Control: The manager of the health-care home wants to provide better services to his guests. Thus, he introduces a new norm for allowing the system to suggest a medical check when some physiological parameters are out of their normal range.

norm_3: *It is permitted to do a medical check if a guest has an irregular heart rate.*

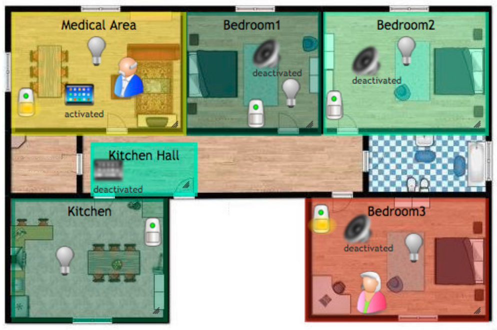
The introduction of the previous norm modifies the normal system behavior, thus allowing not only periodic medical check but also appropriate ones. In such case, the system that is able to monitor the health of guests, when it perceives a heart rate out of normal range (see Fig.7 (a)), it suggests the guest to go to see a doctor and it alerts the nurses (Fig.7 (b)).

d) Conflict Scenario: In such scenario, the diabetic guest goes to the restaurant after having had some gym and before taking insulin. This particular state of the world cause a conflicting situation among two norms. In such case, the system notifies the manager that there is a conflicting situation due to the simultaneous application of the norm_1 and norm_2. If the manager does not modify anything, the system applies the *legis posterior* criterion thus prohibiting the user to go to the restaurant before taking insulin (see Fig.8).

In the following section, we draw some conclusions.

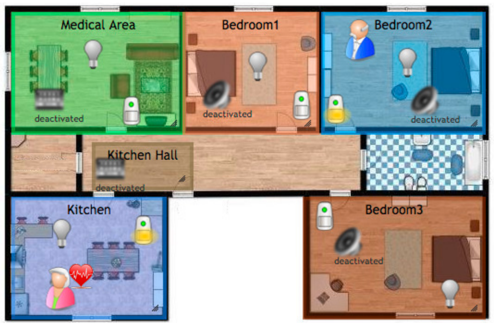


(a) The guest does not use restaurant services. The virtual tutor advises the guest to take insulin.

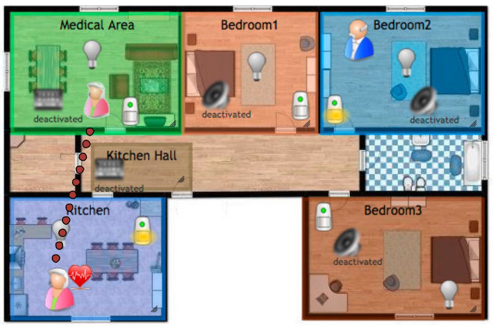


(b) The guest goes to medical area and he takes insulin. The system updates the new state of the world.

Fig. 6: Diabetic Guest scenario



(a) The system perceives an irregular heart rate and it alerts the nurses. The virtual tutor advises the guest to go to make a check.



(b) The guest goes to medical area.

Fig. 7: Heart-Rate Control scenario

VIII. Conclusions

Norms are well know means for regulating the behavior of multi-agent systems, thus ensuring the fulfillment of the overall objective of the society the agents live in. This work takes place in the context of self-adaptive and self-organized systems that consider goals as key elements. They are systems conceived for satisfying user requirements that can be also established at run-time. These systems may evolve over time by increasing the objectives they are able to satisfy. In such context, we defined a normative framework that owns a direct link with the goals the system is able to pursue thus hiding the agent level. In our approach we consider the goals as a particular kind of obligation that have to be satisfied under certain conditions. Besides, we see permission and prohibition norms as promoters or inhibitors of the system in pursuing its goals. By introducing norms at run-time we also make the system more flexible to environment changes and able to self-adapt to new normative contexts in which it could be employed.

Moreover, the proposed algorithm takes into consideration the simultaneous presence of multiple norms related to the same goal, thus determining their joint effect. The approach for run-time injection also provides a means for checking norm conflicts and inconsistencies. Such an approach also implements a recovery mechanism based on the *legis posterior* criterion for solving inconsistencies among norms.

Finally, the simulated environment provides us a means for testing new hypotheses about a real system. In particular, in our future works, it will used for studying the consequences of norms injection in order to discover undesired behaviors of the system or new kinds of conflicting situations.

References

[1] Diane J Cook. Multi-agent smart environments. *Journal of Ambient Intelligence and Smart Environments*, 1(1):51–55, 2009.

[2] Diane J Cook, Michael Youngblood, and Sajal K Das. A multi-agent approach to controlling a smart environment. *Designing smart homes*, 4008:165–182, 2006.

[3] Mathieu Vallée, Fano Ramparany, and Laurent Vercouter. *A multi-agent system for dynamic service composition in ambient intelligence environments*. Citeseer, 2005.

[4] Laura Klein, Jun-young Kwak, Geoffrey Kavulya, Farrokh Jazizadeh, Burcin Becerik-Gerber, Pradeep Varakantham, and Milind Tambe. Coordinating occupant behavior for building energy and comfort management using multi-agent systems. *Automation in Construction*, 22:525–536, 2012.

[5] Huib Aldewereld, Frank Dignum, Andrés García-Camino, Pablo Noriega, Juan Antonio Rodríguez-Aguilar, and Carles Sierra. Operationalisation of norms for electronic institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, pages 163–176. Springer, 2007.

[6] Guido Boella and Leendert WN van der Torre. Regulative and constitutive norms in normative multiagent systems. *KR*, 4:255–265, 2004.

[7] Mehdi Dastani, John-Jules Meyer, and Nick Tinnemeier. Programming norm change. *Journal of Applied Non-Classical Logics*, 22(1-2):151–180, 2012.

[8] Felipe Meneguzzi and Michael Luck. Norm-based behaviour modification in bdi agents. In *Proceedings of The 8th International Conference*

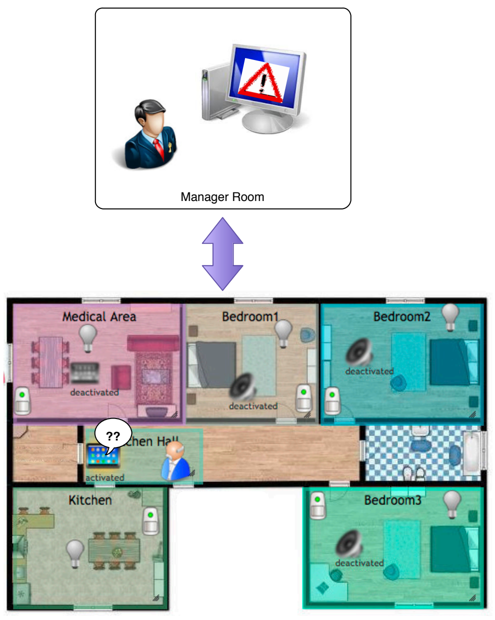


Fig. 8: A conflicting situation is detected by the system. It is notified to the manager. The *legis posterior* criterion is applied.

on Autonomous Agents and Multiagent Systems-Volume 1, pages 177–184. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[9] Luca Sabatucci, Patrizia Ribino, Carmelo Lodato, Salvatore Lopes, and Massimo Cossentino. Goalspec: A goal specification language supporting adaptivity and evolution. In *Engineering Multi-Agent Systems*, pages 235–254. Springer, 2013.

[10] Object Management Group. Semantics of business vocabulary and business rules (sbvr). version 1.3. may 2015.

[11] Massimo Cossentino, Carmelo Lodato, Salvatore Lopes, and Luca Sabatucci. Musa: a middleware for user-driven service adaptation. *in proc. of XVI Workshop "Dagli Ogetti agli Agenti", Napoli, June, 17-19, 2015*, 1382, 2015.

[12] Icasa : a dynamic pervasive environment simulator http://adele.imag.fr/icasa-a-dynamic-pervasive-environment-simulator/.

[13] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[14] Graham Witt. *Writing Effective Business Rules: A Practical Method*. Elsevier, 2012.

[15] Frank Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[16] Thomas Agotnes, Wiebe Van Der Hoek, JA Rodriguez-Aguilar, Carles Sierra, and Michael Wooldridge. On the logic of normative systems. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1181–1186, 2007.

[17] Marek Sergot. Action and agency in norm-governed multi-agent systems. In *Engineering Societies in the Agents World VIII*, pages 1–54. Springer, 2007.

[18] Mehdi Dastani, Nick AM Tinnemeier, and John-Jules Ch Meyer. A programming language for normative multi-agent systems. *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 397–417, 2009.

[19] Natasha Alechina, Mehdi Dastani, and Brian Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1057–1064. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[20] Martin J Kollingbaum and Timothy J Norman. A contract management framework for supervised interaction. In *Working Notes of the 5th UK Workshop on Multi-Agent Systems UKMAS 2002*, 2002.

[21] Nick Tinnemeier, Mehdi Dastani, and John-Jules Meyer. Programming norm change. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 957–964. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[22] Max Knobbout, Mehdi Dastani, and John-Jules Ch Meyer. Reasoning about dynamic normative systems. In *Logics in Artificial Intelligence*, pages 628–636. Springer, 2014.

[23] Jie Jiang, Huib Aldewereld, Virginia Dignum, and Yao-Hua Tan. Compliance checking of organizational interactions. *ACM Transactions on Management Information Systems (TMIS)*, 5(4):23, 2015.

[24] Wamberto Vasconcelos, Martin J Kollingbaum, and Timothy J Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 91. ACM, 2007.

[25] Marc Esteva, Wamberto Vasconcelos, Carles Sierra, and Juan A Rodriguez-Aguilar. Norm consistency in electronic institutions. In *Advances in Artificial Intelligence–SBIA 2004*, pages 494–505. Springer, 2004.

[26] Martin J Kollingbaum, Timothy J Norman, Alun Preece, and Derek Sleeman. Norm conflicts and inconsistencies in virtual organisations. In *Coordination, organizations, institutions, and norms in agent systems II*, pages 245–258. Springer, 2007.

[27] Eric Yu. Modelling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering*, 11:2011, 2011.

[28] Raymond Reiter. *On closed world data bases*. Springer, 1978.

[29] Andrés García-Camino, Pablo Noriega, and Juan-Antonio Rodríguez-Aguilar. An algorithm for conflict resolution in regulated compound activities. In *Engineering Societies in the Agents World VII*, pages 193–208. Springer, 2006.