Verifying Norm Compliancy of Protocols*

Huib Aldewereld, Javier Vázquez-Salceda, Frank Dignum, and John-Jules Ch. Meyer

Institute of Information and Computing Sciences
Utrecht University
{huib, javier, dignum, jj}@cs.uu.nl

Abstract. There is a wide agreement on the use of norms in order to specify the expected behaviour of agents in open MAS. However, in highly regulated domains, where norms dictate what can and cannot be done, it can be hard to determine whether a desired goal can actually be achieved without violating the norms. To help the agents in this process, agents can make use of predefined (knowledge-based) protocols, which are designed to help reach a goal without violating any of the norms. But how can we guarantee that these protocols are actually norm-compliant? Can these protocols really realise results without violating the norms? In this paper we introduce a formal method, based on program verification, for checking the norm compliance of (knowledge-based) protocols.

1 Introduction

Agents in open multiagent systems are sometimes as diverse as humans, as heterogeneous agents may behave in different ways in trying to complete their specified tasks. As some of this behaviour might not be desired, one needs mechanisms to constrain the behaviour of the agents joining the system by defining what is right and wrong. By doing so one can guarantee a safe and regulated environment for the agents to work in.

An Electronic Institution (eInstitution) is such an environment, where the expected behaviour of the agents joining the institution is described by means of an explicit specification of *norms* [9] [24]. As in human institutions, norms in eInstitutions are stated in such a form that allows them to regulate a wide range of situations over time without the need for modification. To achieve this stability, the formulation of norms abstracts from a variety of concrete aspects [11] [24]; i.e., norms are expressed in terms of concepts that are kept vague and ambiguous on purpose [13].

Because of their abstract nature, norms tend to be hard to understand and, as in real life, adhering to the norms that regulate the institution of which you are a part can be, at the least, a bit challenging. It is not unlikely that in highly regulated systems agents (and humans alike) might become overly cautious,

* The research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 634.000.017

trying not to violate any of the norms and thereby seriously reducing their efficiency and even influence the outcome and success of their goals, i.e., desired results, that *are* possible to achieve, might not be achieved anymore because the agent believes that performing the actions leading to the desired result could be violating the norms. In order to help agents act in such an environment and increase their efficiency as well as their chance of success one can specify *norm-compliant protocols* for the tasks that are to be accomplished in the institution.

A norm-compliant protocol is a guideline that makes sure that, when followed, one does not violate any of the norms, and as such it provides a quick and efficient manner to do the tasks one is assigned, since one does not need to review the norms and check norm compliance whenever one is planning to perform an action. In order to guarantee this the protocol should be checked for norm compliance, which means that one should check that no norms are violated by the protocol during its execution in all situations, i.e. the norm compliance of the protocol should not depend on the state of the world. Therefore, the protocol should provide a violation-free path to achieve the agent's goals. As long as the protocol is followed *to the letter* the agent should stay out of harm's way.

In this paper we present a formal method for checking the norm compliance of protocols based on temporal logic, using an approach used in concurrent programming [14]. We have chosen this approach over traditional techniques for verifying (sequential) programs, because verification methods for concurrent programs and temporal logics allow us to see whether norms are violated in intermediate steps as well, where traditional techniques are only for checking the input and output of a program. The formalism of [14] is, however, limited to checking properties and assertions for concurrent programs, not for checking norm compliance. Therefore we enhanced the formalism with the means to express norms and violations and prove the non-violating of these norms by the protocol. Some of the additions to the formalism from [14] are mentioned in the following sections.

The outline of this paper is as follows. We start by a discussion of the work done in field norms and agents. Then, in §3, we present the formal framework and explain some of the difficulties one will encounter when formalising protocols and norms. In §4 we show how the formalism works on an example protocol taken from the medical domain. We end this paper with some conclusions and propose some future work.

The example problem that we are going to use throughout this paper is a real-life protocol that describes which steps should be taken by a doctor to determine whether he can extract the organs of a donor or not (for the use of transplantation). A simplified version of this protocol is included in figure 1. We are using this real-life protocol because of the complexity of the norms applicable to the domain. We feel that if the formalism is able to express and handle such norms, it can be applied to all sorts of normative domains. Also, although it is not feasible to have agents performing the tasks mentioned in this example protocol, protocols that are designed for use by agents are of similar structure

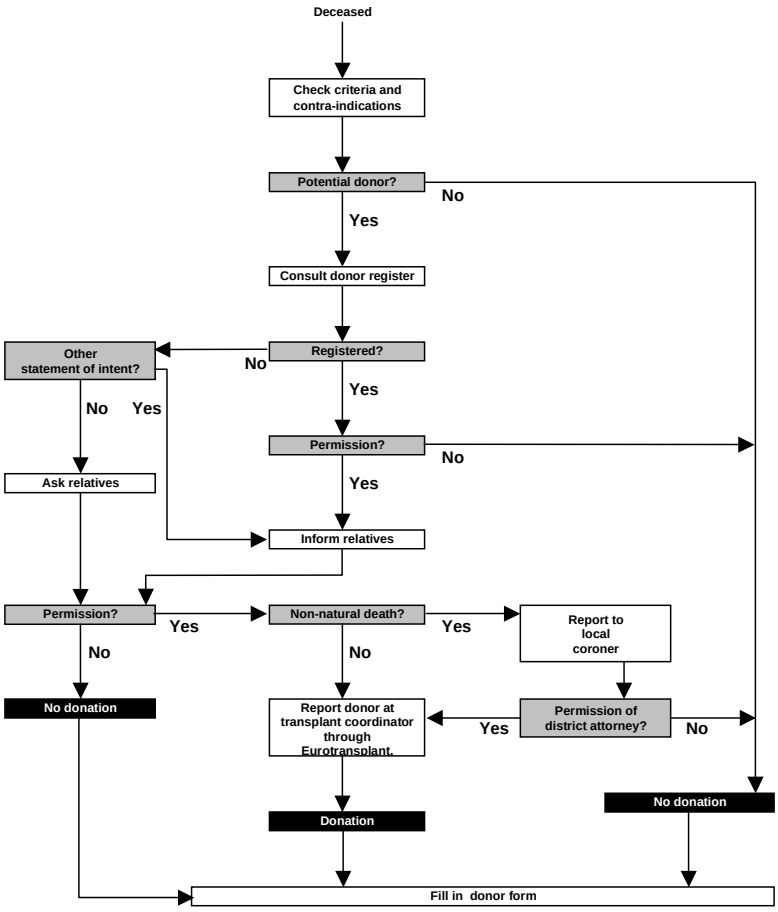

Fig. 1. Protocol for organ donation.

and complexity. And even though this protocol is knowledge-based[1], the method we present in this paper can be applied to other sorts of protocols as well.

2 Related Work

In those situations where agents might deviate from expected behaviour to fulfil their own goals, a multi-agent system needs mechanisms to defend and recommend right and wrong behaviour, along with safe environments to support those

[1] Knowledge-based protocols depend on the knowledge of the agent to decide which action is to be performed next, which results in a change of knowledge. The goal of such protocols is to determine whether something is known by the agent at the end of the protocol's execution.

mechanisms. As we mentioned in §1, in eInstitutions expected behaviour is defined by means of *norms*. But providing agents with a set of norms is not enough; an eInstitution should also ensure *norm compliance*.

In literature, there are two approaches for norm compliance from the individual agent perspective:

- agents that always obey norms [3] [20]
- agents that autonomously decide to obey norms [2] [5] [7] [16] [15] [24].

The former ensures norm compliance by default and it is used in those domains where total control of the agent behaviour is needed, but issues on the conflict between the agent goals and the norms should be solved. The latter allows the design of dynamic systems where agents are able to join a society while satisfying their own goals. The conflict between the agents' goals and the norms controlling its behaviour is handled explicitly in the reasoning process of the agent. In [15], *autonomous norm compliance* is divided in two separate processes: a) a process to deliberate about whether to comply with a norm (the *norm deliberation process*), and b) a process to update the goals and intentions of agents accordingly (the *norm compliance process*).

In those systems where autonomous norm compliance is allowed from the agent perspective, there is a need to enforce to some extent the compliance of norms from the social perspective. In [15] there is no direct enforcement on norm compliance, but *influenced* norm compliance, where behaviour of other agents against the non-compliance of a norm influences the decision of each agent. In [23] a more direct approach is taken: the agent platform hosting the eInstitution provides time-efficient services to help a special type of agents (the *Police Agents*) to enforce proper behaviour. As Police Agents cannot see or control the internal mental states and the reasoning process of the other agents, norm enforcement is based on the detection of *violation situations* in terms of (public) messages and (visible) actions.

The use of protocols to ease agent interaction (as discussed in §1) adds an extra level between the norms and the agent behaviour.In this case norm compliance is divided into two different levels:

- *norm compliance of the protocol*: to ensure that a given protocol adheres to the norms defined in a context. If the protocol is norm-compliant, following the protocol ensures that the agent(s) will not violate the norms.
- *protocol compliance by the agent*: to check that the behaviour of an agent complies with the expected behaviour defined by the protocol [25].

The former is the focus of this paper (see §3.3 and §4), as it is usually overlooked in other works. The latter (protocol compliance) has been studied both in theoretical and practical approaches. In [25] a formal framework for commitment protocols is presented. Verification in this case is an external process and therefore it cannot use the internal knowledge of the agents, only the (observable) behaviour. In [8] protocol compliance is handled by means of interaction scripts that are explicitly accepted by the agents through *interaction contracts*. Each

contract includes the interacting agents, the roles they are playing, the contract clauses and the protocol. Verification of protocol compliance is an optional clause in the contract that, if included, specifies who and how will verify the interaction and the actions to take if the protocol is not followed. In [9] interaction protocols are structured in a *performative structure*. Although agents can decide not to follow the protocol (there is no direct control of the agent platform over the agents' beliefs and desires), there is an intermediate actor, the *governor*, that filters any non-allowed message that the agent tries to send to the eInstitution and is not allowed. Therefore protocol compliance is ensured by filtering those messages that, for a given state of the interaction, are not included in the protocol as possible messages to be uttered. However, in none of these works there is a method to ensure that the protocols are norm-compliant.

3 Our Approach

In this section we will set out the steps of the verification process necessary for checking the norm compliance of protocols. While discussing these steps we will also focus on some interesting aspects and problems that one can encounter.

3.1 Formalising the Protocol

First, we start with formalising the protocol that we want to check. Since protocols are very similar to programs we have based our protocol checking formalism on the formal verification methods designed for parallel programs taken from [14]. This program verification method uses first-order linear-time temporal logic (LTL) to express how programs change the world over time, and uses this logic to prove that certain specified properties of a program hold (e.g., deadlock freedom, mutual exclusion, termination, etc.). In this paper we will not go into an elaborate syntactical and semantical definition of the language used and will only give the informal interpretations of the operators. The proper definitions of the operators can be found in [1].

The protocol we want to verify is translated into a program using a syntax containing among others variable assignments, **if** $-$ **then** $-$ **else** $-$ **fi** and **while** $-$ **do** $-$ **od** statements, with the conditions of these statements being formulas of a classical first-order predicate logic $\mathscr{L}_P$. For ease of reference all statements are labelled, with the labels being unique throughout a program, i.e., no two labels occurring in a program are equal. Using this we can formalise the example-protocol from figure 1 as follows (because of space limitations we only include the part of the protocol necessary for the proof we provide in §4, the complete version can be found in [1]):

$\Pi =$ **Initial R**;

π_0: ⟨check_criteria_&_contra-indications⟩;
π_1: **if** $know\_criteria(d,y) \wedge know\_no\_contra\text{-}indication(d,y)$
then π_2: $know\_potential\_donor(d,y)$:=TRUE
else π_3: $know\_not\_potential\_donor(d,y)$:=TRUE

fi;
π_4: **if** $know\_potential\_donor(d,y)$
then π_5: ⟨consult_donor_register⟩
fi;
⋮
π_{33}: ⟨fill_donor_form⟩;
π_e: **stop**

This formal program is a representation of the top three boxes of the protocol shown in figure 1 and describes the steps taken by the doctor. These steps are; 1) the doctor first checks whether the patient satisfies the conditions and none of the contra-indications for becoming an organ donor (formalised in π_0), then 2) using the results of these tests the doctor determines whether the patient is a suitable donor (formalised in π_1 to π_3). In the case that the patient does not satisfy the criteria or shows one of the contra-indications, the doctor knows that the patient is not a potential donor, thereby terminating the protocol (shown in π_3[2]). However, if the patient does satisfy the criteria and does not show any of the contra-indications, the doctor determines that the patient is a potential donor (see π_2), and then 3) continues to check whether the patient has registered his permission (or prohibition) for extracting organs for transplantation (formalised in π_4 and π_5).

Note that the result of the actions like check_criteria_&_contra-indications is dependent on the domain, i.e. if the patient satisfies the criteria and none of the contra-indications, the result of the action in π_0 would be that the doctor knows this, thus $know\_criteria(d,y) \wedge know\_no\_contra\text{-}indication(d,y)$.

The logic used for verifying protocols consists of a classical first-order predicate logic $\mathscr{L}_P$ which is extended with $\bigcirc$ (*next-state*), $\Box$ (*always*) and atnext (*first time*) operators to obtain a first-order linear-time temporal predicate logic $\mathscr{L}_{TP}$. Using these operators we can also derive the $\Diamond$ (*sometime*) and until operators. To reason about events in the past, $\mathscr{L}_{TP}$ is extended with past-time operators, which are discussed in the next section (for formal semantics of $\mathscr{L}_{TP}$ see [14] and [1]). The logic $\mathscr{L}_{TP}$ is then expanded to $\mathscr{L}_{TP_\Pi}$ by adding the set of propositional variables at λ (which means that action labelled λ is next to be executed), to link the protocol state to a state in the temporal model of the logic. Therefore, although the protocol has actions, the logic, instead, only uses the labels of the actions.

In order to prove that a protocol is norm-compliant in all situations that might arise we need to check the protocol in various different models and see whether the norm compliance holds. For instance, a protocol for obtaining donor organs needs to be checked in models where the donor is male and in models where the donor is female to determine that the protocol does not violate a norm about not discrimination between donors based on sex, race, etc. Only if the protocol generates the desired and expected result in both situations, we can say that the protocol does not violate that norm.

[2] The fact that $know\_not\_potential\_donor(d,y)$ becomes true means in this protocol that all of the next steps are skipped, and the protocol terminates by specifying that no permission can be obtained.

Since time is defined as semi-finite (it starts at the start of the protocol), protocols cannot use information about previous runs and next runs (unless explicitly modelled). If protocols Π_1 and Π_2 are run one after another (i.e., $\Pi_1; \Pi_2$), Π_1 cannot use information from or about Π_2 and Π_2 cannot use information from or about Π_1; they are considered as separate runs. This means that the value of program variables, truth values of predicates and states and information gathered is restricted to the runtime of the protocol. Propositions and predicates *can* change their truth values during a protocol run, however, but only because of actions taken in the protocol.

3.2 Formalising the Norms

The norms that apply to the domain in which we are checking the protocol are then translated into a high-level formal language, which should provide enough room for the formal representation of the norms to keep their abstract nature. We have used a formalism similar to the one used in [23]. In order to be able to use these high-level formalised norms in the checking of the protocols we needed to extend the first-order language specified above with deontic concepts, O_x, P_x, F_x, to express x being obliged, permitted or prohibited some action, respectively. To give meaning to these deontic operators we introduced special predicates to denote when violations occur. To handle the temporal aspects of norms, such as deadlines, we used ideas from [4], [6] and [8] and adapted these to be used with the first-order temporal logic as specified above. Furthermore we have extended our language with $DO_x\ \lambda$ (x *is going to do* λ *next*) to reason about actions and $\diamondsuit\varphi$ (*past operator*) and $\odot\varphi$ (*previous-state operator*) to reason about the events that have happened (such as actions that have been done: $DONE_x\,\alpha \equiv \odot DO_x\,\alpha$).

The deontic operators discussed above are introduced as abbreviations of complex temporal formulas. Definition 1 shows the temporal translations of obligations in our formalism (based on [8]).

Definition 1 (Obligations).

$$O_x(DO_x\,\alpha\ <\ \delta) \equiv \Diamond\delta \wedge \Big[(\neg\delta \wedge \neg viol(x,DO_x\,\alpha,\delta) \wedge \neg DONE_x\,\alpha)\ \textsf{until}\ ((DO_x\,\alpha\ \wedge \bigcirc(\Box\neg viol(x,DO_x\,\alpha,\delta))) \vee (\neg DO_x\,\alpha \wedge \bigcirc(\delta \wedge viol(x,DO_x\,\alpha,\delta))))\Big]$$

Similar temporal translations are made for permissions and prohibitions (not included here due to space constrains, see [1] for these definitions).

Norms applicable to the example mentioned in §1 are, for instance, obliging doctors to talk to relatives for obtaining permission before extracting organs from a donor, prohibiting the extraction of organs without the approval of the district attorney in case of suspicion of a non-natural death , etc[3]. In §4 we prove that the protocol abides to the norm that doctors are obliged to pronounce dead of a patient before removing an organ.

[3] A full set of norms is available in [1].

Permissions and Non-Permissions In theoretical deontic studies, such as [17], [21] permissions are normally modelled as $P_x(DO_x\,\alpha) \equiv DO_x\,\alpha \rightarrow \neg viol(x,\alpha)$, which says that being permitted to do α means that doing α leads to non-violation. Moreover, permissions are, in classic deontic studies, normally defined as being equivalent to $\neg F_x(DO_x\,\alpha)$ and $\neg O_x(\neg DO_x\,\alpha)$. The problem with this definition, which is also discussed in deontic studies (cf. [18]), is that it makes the existence of permissive norms nonessential when trying to determine whether violations occur. From observations of the legal domain, and as already proposed in [18], it follows, however, that permissions can be considered as exceptions to a general prohibition. The fact that an article in a law provides a certain set of people in a certain situation with the permission to do α means that in other situations these people, or other people at all times, are prohibited to do α. Some lawbooks even express this explicitly by means of an article that something is forbidden unless stated otherwise within that lawbook. We model this relation between permissions by a technique similar to *negation as failure*, as used in logic programming [22]; the inability to derive that you are permitted to do α means that you are forbidden to do α:

$$\sim P_x(DO_x\,\alpha) \rightarrow F_x(DO_x\,\alpha)$$

Of course, we could have opted for a relation in the other direction, i.e., $\sim F_x(DO_x\,\alpha) \rightarrow P_x(DO_x\,\alpha)$ which means that if something is not explicitly forbidden it is permitted. The choice between whether to use the first or the second relation entirely depends on the nature of the norms one is trying to formalise, i.e., the choice is dependent on the character of the legal system, thus whether it is permissive in nature or restrictive (see [19] for a discussion on the character of legal systems).

Now, since we add the $\sim P_x(DO_x\,\alpha) \rightarrow F_x(DO_x\,\alpha)$ rule to our system to model that permissions are exceptions to general prohibitions (where this general prohibition might only follow from the characteristic nature of the law), we get into trouble if we don't assume that permissions follow from obligations (i.e., $O_x(DO_x\,\alpha) \rightarrow P_x(DO_x\,\alpha)$). This assumption is an axiom in most deontic systems, but we are reluctant to insert it because we feel that in the real world this might not necessarily hold. It is, however, true that a normative system is supposed to uphold this principle, i.e., normative systems should be designed such that obligations to do α can actually be fulfilled, but this is actually the ideal situation. When designing a normative system (thus, when laws are postulated) it should be taken into account that obligations can be fulfilled. However, it is not necessarily the case that this condition is always met in normative systems (due to mistakes in designing the system). In the case presented in §4, however, we can safely assume that this assumption has not been violated by the law-maker.

Linking Levels A problem that arises because of the high-level of abstraction for the formalisation of norms is the mapping between the concepts in the norms and the actions specified by the protocols. In order for the norms to range over

a wide variety of situations, and in order to function for a long duration without the need of modification, norms tend to abstract from a variety of concrete aspects, such as time, role, etc. Therefore, in order to check whether certain concrete actions and situations contained in the protocol violate a norm we need to map these concrete actions and situations to the abstract actions and situations described by the norms. The mappings that we can provide are generally considered to be one-way mappings, that is, a concrete action a in a protocol can be considered to be an instance of an abstract action α mentioned in the norms, but since there are many more actions conceivable that can be considered instances of α, we cannot say that a and α are equivalent; we can only say that a is an instance of α, or that doing a *counts as* doing α ($DO_x\, a \rightsquigarrow DO_x\, \alpha$). Although this mapping problem seems to follow from our high-level formal language, it is also present when using formalisms with a lower abstraction level (although implicit). The explicit mapping that we need to make between the protocols and norms now is in such a case taken care of when formalising the norms (by means of choosing the appropriate concrete concepts in the formalisation of the norms). In this paper we use a simplified version of the *counts as* as defined in [10] and [12].

3.3 Verifying Protocols

The next step of the process is the actual verification of the protocol. The formalism that we have chosen allows us to specify properties that are verified by means of automated reasoning. This means that we check the protocol in all sorts of different situations (that apply to the protocol and norms) in order to check whether all situations guarantee the norm compliance that we require.

In order to check the protocol on norm compliance we specify a *safety* property that has to be derivable from the protocol. This safety property is an invariant, a formula that should hold during the entire execution of the protocol. We define the safety property for checking protocols as follows:

Definition 2 (Safety Property of Protocols).

$$\text{start}_{\Pi} \wedge \Box Norms \rightarrow \Box \neg violation$$

Where $\text{start}_{\Pi} \equiv \text{at}\ \alpha_0$ (*the protocol is at its start label*), $Norms$ being the conjunction of all applicable norms, and $violation \equiv \bigvee_{x,\alpha,\delta} viol(x, \alpha, \delta)$ (*violation is the disjunction of all viol-formulas that occur in* $Norms$). This safety property of protocols is defined as the global invariance of $\neg violation$ for the protocol Π under the condition that $Norms$ always holds, i.e., if $\Box Norms$ holds upon the start of running Π, then $\neg violation$ will hold in all states of the run.

To prove that a protocol satisfies this property we introduce the following rule:

Theorem 1 (Invariance Rule). *The following rule is valid:*

$$\frac{\begin{array}{l}\text{start}_{\Pi} \wedge \Box Norms \rightarrow \neg violation \\ \neg violation \text{ invof } \mathscr{M}_{\Pi}\end{array}}{\text{start}_{\Pi} \wedge \Box Norms \rightarrow \Box \neg violation}$$

Where C invof $\alpha \equiv \text{at}\,\alpha \wedge C \rightarrow \bigcirc C$ (C *is an invariant of* α) and C invof $\mathscr{M} \equiv C$ invof $\alpha_1 \wedge \ldots \wedge C$ invof α_m (C *is an invariant of every* $\alpha \in \mathscr{M}$), and $\bar{\mathscr{M}}_{\Pi}$ is the set of all labels in the program except for the label of stop, the end-statement. A proof of this theorem can be found in [14]. This rule is also very close to the intuition one might have about protocols being norm-compliant, namely if there are no steps in the protocol that violate any norm, the protocol will not violate any of the norms as a whole (if no violation existed when the protocol started running).

Of course, this is not the only property that a protocol needs to satisfy. Because law is generally applicable to a single context, one who is not participating in the activities of that context is not regulated by these laws; the laws mean nothing to someone not trying to do anything regulated by that particular set of laws. For instance, traffic laws have no influence on those who do not participate in traffic situations; if someone sits at home all times, these laws will never be violated. The problem is that laws regulating a specific domain assume that you are trying to do something or otherwise participate in that domain, and only regulates these actions and participations.

While all protocols that satisfy the aforementioned safety property are compliant to the norms, we would actually like to be able to say a bit more about the protocols we are trying to verify. Since protocols that do satisfy the safety protocol, and thereby the norm compliance, that merely consist of actions that are not regulated by the applicable norms, are not that interesting to the agents interacting in the eInstitutions (e.g., although "**while** `True` **do** skip **od**" does satisfy almost all violation invariances, it is not very interesting from an interaction or institution's point of view). Therefore, we need to define another property that allows us to determine whether a protocol is, next to being norm-compliant, also trying to achieve something interesting. Norm-compliant protocols that are actually relevant to the domain not only satisfy the violation invariance property, but also a liveness property. This sort of properties specify that a protocol/program will, at sometime, reach a certain (interesting) state. We can use this to check whether the protocol achieves a specified goal at the end of its run:

Definition 3 (Liveness Property of Protocols).

$$\text{start}_{\Pi} \wedge \Box Norms \rightarrow \Diamond(\text{at}\ \alpha_e \wedge goal)$$

Where at α_e is the stop-statement of Π and *goal* is the goal that the protocol should reach. In our example this is a complex declarative statement specifying that when the conditions hold (i.e., the donation should ideally take place), the agent/doctor running the protocol will know that the donation can take place, and when one of the conditions for the donation fails, the agent/doctor knows that the donation cannot take place.

4 Practice

Now that we have seen a description of the approach we are using to verify the norm compliance, we show in this section how this approach is to be used. We show this by using the example protocol mentioned above in figure 1. To ensure the norm compliance of this protocol we need to check whether the safety and liveness properties, as specified before, are satisfied. Although it is possible to give a fully formal proof we will only show the first steps due to space limitations. In this proof we assume that y denotes the patient with respect to whom the protocol is run, d denotes the doctor running the protocol and d' is a doctor-variable (denoting a unspecified doctor).

For the invariance proof, i.e. proving that $\neg violation$ is an invariant of the protocol, we make use of the invariance rule as mentioned in theorem 1. We assume that $\text{start}_\Pi \wedge \Box Norms \rightarrow \neg violation$ holds (1) and will try to prove that $\neg violation$ is an invariance of every following step of the protocol, thereby deriving that $\neg violation$ is an invariant of the protocol. We can make this assumption because we are not interested in the situations where this assumption does not hold, such as the situation in which the protocol is started when a violation has already occurred, since starting the protocol in such a situation would say nothing about the norm compliance of the protocol, only that it cannot "repair" the situation it started in.

Note that we only need to check the actions taken by the protocol, since the "control points" used in the protocol (i.e. protocol labels referring to conditions of **if**-clauses) are trivially norm-compliant since they do not change the value of any *viol*-predicate (actually, the action that is thereafter chosen shows whether the decision made at the control point was correct). This is expressed in step (3).

(1) $\text{start}_\Pi \wedge \Box Norms \rightarrow \neg violation$	assumption
(2) $\text{start}_\Pi \rightarrow (\text{at } \pi_0 \wedge intented(organ\_removal))$	definition of start_Π
(3) $\neg violation \text{ invof } \mathscr{M}_\Pi \setminus \{\pi_0, \pi_5, \pi_7, \pi_9, \pi_{14}, \pi_{16}, \pi_{21}, \pi_{23}, \pi_{24}, \pi_{26}, \pi_{33}\}$	Trivial

Next we prove that step π_0 of the protocol (checking whether the patient satisfies the criteria and none of the contra-indications for being a donor) is norm-compliant. The only norm in the law concerning this actions is the fact that doctors are supposed to check whether a patient is brain death before removing any organs, of which the translation is seen in step (5). In order to use this deontic expression for determining whether violations occur, we need to "expand" the norm in (5) to its temporal counterpart by using the definition 1 seen earlier, as seen in (6). Now, since we can derive from the structure of the protocol that $DO_{d'}\, remove\_organ(d', y)$ has not yet occurred, or is occurring now (7), we can derive that the value of V_1 will not be changed by $DO_d\, certify\_dead(d, y)$, shown in (8) and (10). Finally, after connecting the abstract norm level to the protocol level using (4) to derive (11), remembering the fact that obligations imply permissions (12) (and therefore do not lead to violations by acting upon

the obligation)[4], and adding the fact that no other norms were applicable and thereby cannot be violated (13), we can conclude that $\neg violation$ is an invariant of π_0, see (15).

(4)	at $\pi_0 \rightsquigarrow DO_d\ certify\_dead(d,y)$	
(5)	$O_d(DO_d\ certify\_dead(d,y) < DO_{d'}\ remove\_organ(d',y))$	Art. 14
(6)	$\Diamond DO_{d'}\ remove\_organ(d',y) \wedge \big[(\neg DO_{d'}\ remove\_organ(d',y) \wedge \neg V_1 \wedge \neg DONE_d\ certify\_dead(d,y))$ until $((DO_d\ certify\_dead(d,y) \wedge \bigcirc\Box\neg V_1) \vee (\neg DO_d\ certify\_dead(d,y) \wedge \bigcirc(DO_{d'}\ remove\_organ(d',y) \wedge V_1)))\big]$	
	$V_1 = viol(d, DO_d\ certify\_dead(d,y), DO_{d'}\ remove\_organ(d',y))$	(5)
(7)	$\neg\Diamond DO_{d'}\ remove\_organ(d',y)$	(Π)
(8)	$DO_d\ certify\_dead(d,y) \wedge \neg violation \rightarrow \bigcirc\Box\neg V_1$	(6),(7)
(9)	$\bigcirc\Box\varphi \rightarrow \bigcirc\varphi$	($taut$)
(10)	$DO_d\ certify\_dead(d,y) \wedge \neg violation \rightarrow \bigcirc\neg V_1$	(8),(9)
(11)	at $\pi_0 \wedge \neg violation \rightarrow \bigcirc\neg V_1$	(4),(10)
(12)	$P_d(DO_d\ certify\_dead(d,y) < DO_{d'}\ remove\_organ(d',y))$	(5)
(13)	at $\pi_0 \wedge \neg violation \rightarrow \bigcirc\neg viol(d,\alpha,\delta)$ for all viol-predicates other than V_1	(VC)
(14)	at $\pi_0 \wedge \neg violation \rightarrow \bigcirc\neg violation$	(11),(13)
(15)	$\neg violation$ invof π_0	(14)

And so, after checking the norm compliance of π_0 we continue with checking whether the next actions (starting with π_5 and so on, see the formalised protocol in §3.1 and the full proof in [1]) do not violate the norms. After deriving that $\neg violation$ is an invariant of all the protocol steps we can derive, by theorem 1, that the protocol does not violate any of the norms, see (111).

	$\vdots$	
(110)	$\neg violation$ invof π_{33}	
	No norms concerning filling in donor form (VC)	
(111)	$\neg violation$ invof $\mathscr{M}_{\Pi}$	(1),(3),(15),...,(110)

In a similar fashion we can prove that a liveness property as specified in definition 3 holds for Π. Where

$$
\begin{aligned}
\text{at}\,\alpha_e \;&\equiv\; \text{at}\,\pi_e \\
goal \;&\equiv\; criteria(y) \wedge \neg contra\text{-}indication(y) \wedge (statement\_permission(y) \vee other\_statement(z,y) \vee relative\_permission(y)) \wedge \\
&\qquad (\neg non\text{-}natural\_dead(y) \vee DA\_permission(p,remove\_organs)) \\
&\qquad \rightarrow know\_permission(d,remove\_organ(y)) \\
&\bigwedge\; \neg\,(criteria(y) \wedge \neg contra\text{-}indication(y) \wedge (statement\_permission(y) \vee other\_statement(z,y) \vee relative\_permission(y)) \wedge \\
&\qquad (\neg non\text{-}natural\_dead(y) \vee DA\_permission(p,remove\_organs))) \\
&\qquad \rightarrow know\_no\_permission(d,remove\_organ(y))
\end{aligned}
$$

This *goal* represents that the protocol is supposed to make sure that the agent obtains the knowledge whether it has the permission for the organ transplantation or not, after ending the protocol run. By proving these safety and liveness properties we show that Π is not only norm-compliant, but also that Π actually achieves the goal for which it is designed (that is, to determine whether you are allowed to extract the patients organs for transplantation).

[4] Remember that not being able to derive this permission would have meant that there existed a prohibition on this action, see §3.2

5 Conclusions & Future Work

In this paper we discussed a formal approach on norm compliance of protocols based on the verification of programs. We give a view of how these techniques can be used, after some adaptation and extension, to verify that a (knowledge-based) protocol is norm-compliant. We also show, as an example, how norm compliance of a knowledge-based protocol (actually used in the medical domain) can be proven.

Please note that norm compliance of the protocols used by the agents is only a step towards the implementation of norms in MAS. Protocols are guidelines and agents are, therefore, not necessarily constrained to follow the protocol. A more direct enforcement is needed instead. Norms can be enforced either by the use of violation detection and sanctioning these violations [23], or by directly constraining the agents such that they can only do actions that do not violate norms.

Currently our formal method is suited for verification of single sequential protocols. We plan to extend our $\mathscr{L}_{TP_{\Pi}}$ language to prove norm compliance of parallel protocols (such as interaction protocols). We also plan to extend the $\mathscr{L}_{TP}$ language with operators from epistemic logic in order to improve expressiveness of knowledge and beliefs of agents following a protocol. Moreover, we are very interested in seeing how this extended approach can, for instance, be used for the checking of security and authentication protocols.

The framework discussed in this paper uses a theorem proving method to verify the norm compliance of protocols. This is known to be labour-intensive. We are currently considering the use of model-checking, instead.

References

1. H. Aldewereld, J. Vázquez-Salceda, F. Dignum, and J.-J.Ch. Meyer. Proving norm compliancy of protocols in electronic institutions. Technical Report UU-CS-2005-010, Institute of Information and Computing Sciences, Utrecht University, 2005.
2. G. Boella and L. Lesmo. Deliberative normative agents. In C. Dellarocas and R. Conte, editors, *Workshop on Norms and Institutions in Multi-Agent Systems*, pages 15–25. ACM-AAAI, ACM Press, 2000.
3. M. Boman. Norms in artificial decission making. *Artificial Intelligence and Law*, 7(1):17–35, 1999.
4. J. Broersen, F. Dignum, V. Dignum, and J.-J. Ch. Meyer. Designing a Deontic Logic of Deadlines. In *7th Int. Workshop on Deontic Logic in Computer Science (DEON'04)*, Portugal, May 2004.
5. C. Castelfranchi, F. Dignum, C. Jonker, and J. Treur. Deliberative Normative Agents: Principles and architecture. In *Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, 1999.
6. F. Dignum, J. Broersen, V. Dignum, and J.-J. Ch. Meyer. Meeting the Deadline: Why, When and How. In *3rd Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS)*, Maryland, April 2004.
7. F. Dignum, D. Morley, and E.A. Sonenberg. Towards socially sophisticated BDI agents. In *DEXA Workshop*, pages 1134–1140, 2000.

8. V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic*. SIKS Dissertation Series 2004-1. SIKS, 2004. PhD Thesis.
9. M. Estava. *Electronic Institutions: from specification to development*. PhD thesis, Universitat Politècnica de Catalunya, 2003.
10. D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. Accepted for the 1st International Symposium on Normative Multiagent Systems (NorMAS2005), 2005.
11. D. Grossi and F. Dignum. From abstract to concrete norms in agent institutions. In M. G. Hinchey, J. L. Rash, W. F. Truszkowski, and et al., editors, *Formal Approaches to Agent-Based Systems: Third International Workshop, FAABS 2004*, Lecture Notes in Computer Science, pages 12–29. Springer-Verlag, April 2004.
12. D. Grossi, F. Dignum, and J-J. Ch. Meyer. Contextual taxonomies. In J. Leite and P. Toroni, editors, *Proceedings of CLIMA V Workshop, Lisbon, September*, pages 2–17, 2004.
13. H. L. A. Hart. *The Concept of Law*. Clarendon Press, Oxford, 1961.
14. Fred Kröger. *Temporal Logic of Programs*, volume 8 of *EACTS monographs on theoretical computer science*. Springer-Verlag, 1987.
15. F. López y Lopez. *Social Power and NormsL Impact on Agent Behaviour*. PhD thesis, Faculty of Engineering and Applied Science, Univ. of Southampton, 1997.
16. F. López y Lopez, M. Luck, and M. d'Inverno. A framework for norm-based inter-agent dependence. In *Proceedings of The Third Mexican International Conference on Computer Science*, pages 31–40. SMCC-INEGI, 2001.
17. J.-J. Ch. Meyer and R.J. Wieringa. Deontic logic: A concise overview. In *Deontic Logic in Computer Science: Normative System Specification*, pages 3–16. John Wiley & Sons Ltd., Chichester, UK, 1994.
18. L. Royakkers and F. Dignum. Giving permission implies giving choice. In E. Schweighofer, editor, *8th International Conference and Workshop on Database and Expert Systems Applications*. Toulouse, France, 1997.
19. M.J. Sergot, F. Sadri, R.A. Kowalski, and F. Kriwaczek. The british nationality act as a logic program. *Communications of the ACM*, 29(5):370–386, May 1986.
20. Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
21. A. Soeteman. *Logic in Law: Remarks on logic and rationality in normative reasoning, especially in law*. Kluwer Academic Publishers, 1989.
22. M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, October 1976.
23. J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Implementing norms in multiagent systems. In G. Lindemann, J. Denzinger, I.J. Timm, and R. Unland, editors, *Multiagent System Technologies*, LNAI 3187, pages 313–327. Springer-Verlag, 2004.
24. J. Vázquez-Salceda and F. Dignum. Modelling electronic organizations. In V. Marik, J. Muller, and M. Pechoucek, editors, *Multi-Agent Systems and Applications III*, LNAI 2691, pages 584–593. Springer-Verlag, 2003.
25. Mahadevan Venkatraman and Munindar P. Singh. Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.