Explicit Representation of Social Norms for Social Robots

Fabio Maria Carlucci[1] and Lorenzo Nardi[1,2] and Luca Iocchi[1] and Daniele Nardi[1]

***Abstract*— As robots are expected to become more and more available in everyday environments, interaction with humans is assuming a central role. Robots working in populated environments are thus expected to demonstrate socially acceptable behaviors and to follow social norms. However, most of the recent works in this field do not address the problem of explicit representation of the social norms and their integration in the reasoning and the execution components of a cognitive robot.**

In this paper, we address the design of robotic systems that support some social behavior by implementing social norms. We present a framework for planning and execution of social plans, in which social norms are described in a domain and language independent form. A full implementation of the proposed framework is described and tested in a realistic scenario with non-expert and non-recruited users.

I. INTRODUCTION

Social robots are robots designed for effective social interactions with humans in populated environments. This research field is consequently gaining more and more attention in the areas of robotics and artificial intelligence.

Many approaches to develop social robots have been considered. In [1], a survey of socially interactive robots is presented, describing design methods and system components of several realizations of social robots. In this survey and in other recent papers about social robots, a lot of effort is put in the design of the physical appearance of the robot and in the development of low-level behaviors for human-robot interaction (such as, speech, gestures, navigation in public environments, etc.).

On the other hand, sociality of a robot comes also from the definition and the application of social norms (or rules), that define the modality of interactions with humans. In [2], social norms are mentioned as a design feature for a social robot, but no examples of architectures or robots explicitly defining these social norms are described. Conversely, in the community of multi-agent systems, the definition and explicit representation of social norms for agents has been very well developed (see for example, [3]).

In this paper, we address the problem of how the social norms can be represented and used by a social robot to execute a social behavior. We present a framework for planning and execution of social behaviors that has the following properties: 1) it allows for a clear separation between the specification of the domain and the specification of the social norms; 2) social norms can be expressed in a language and domain independent manner (so they can be reused for other tasks in other domains or with other reasoning formalisms); 3) the generation of social plans integrating domain description and social norms is completely automated; 4) different kinds of social behaviors can be obtained by simply enabling or disabling some social norms. The proposed framework has been fully implemented and extensively tested on a mobile robot in our department. The ability of producing different kinds of social behaviors by just enabling/disabling some norms allowed us to easily configure the system to test many different kinds of social behaviors.

In the following, after a brief description of the related works, our framework is presented, an example of the application of the proposed algorithm is illustrated, then implementation and experimental runs are described, while conclusions are drawn in the last section.

II. RELATED WORK

Different studies (e.g. [1]) support the idea that a social robot, in order to interact and collaborate with humans in a natural and friendly way, should attempt to manifest believable behaviors complying with the same social norms as humans do. Over the last years, different systems have been proposed to allow a robot performing some tasks in a socially acceptable way in environment shared with humans. However, most of them deal with specific behaviors for a particular task and they are hard to generalize.

Many systems implicitly model social norms in the architecture: this approach, while functional, has some drawbacks. For example, Partially Observable Markov Decision Processes (POMDP) are used to model human-robot interaction allowing the robot to act according to beliefs about the goals of the people it is interacting with based on observations [4]. In Human Aware Task Planner (HATP), social rules are used to define penalties to situations that are then used by the planning procedure to score different solutions and choose the "best" plan [5]. Answer Set programming (ASP) has been used to model an efficient planner for collaborative house keeping robots [6]. This system automatically extracts common sense information from the ConceptNet dataset and exploits it to better perform the given task. Finally, Human Aware Planning based on Constraint Based Planning (CBP) allows the planner module to take into account social norms when generating the plan [7].

In all these works, however, social norms are expressed in a domain dependent and language dependent formalism, i.e. they are embedded in the POMDP, HATP, ASP or CBP specifications together with the specification of the domain and of the task to be accomplished. Only in HATP, social rules are defined in a separate element with respect to the

1 Department of Computer, Control and Management Engineering, Sapienza University of Rome, Italy. Contact: iocchi@dis.uniroma1.it.
2 Institute for Geodesy and Geoinformation, University of Bonn, Germany.

This work has been partly carried out within the COACHES project within CHIST-ERA 4^{th} Call for Research projects, 2013, Adaptive Machines in Complex Environments (AMCE) Section, funded by MIUR Italy and by the European Commission under the grant FP7-610603-EUROPA2.

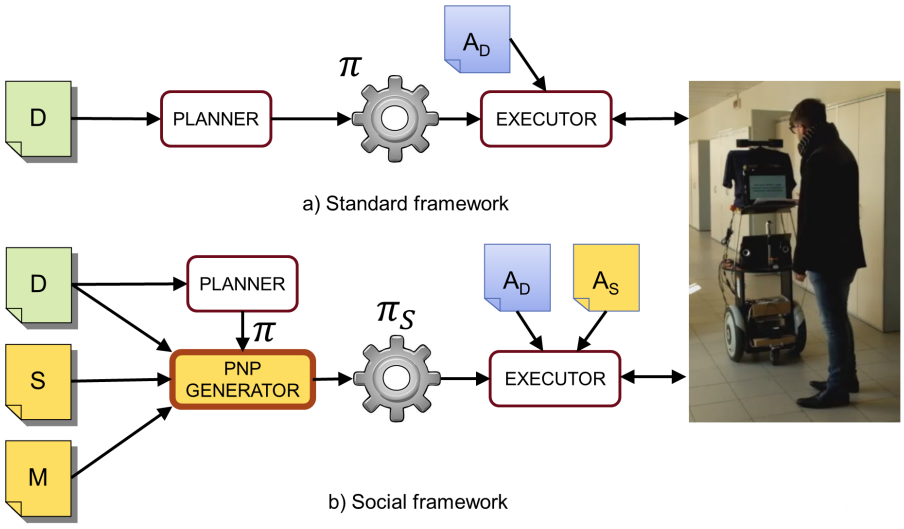

Fig. 1. Standard framework and social framework described in this paper. D: domain, A_D: domain actions, S: social norms, M: domain specifications, A_S: social actions, π: plan, π_S: social plan. Planner, Executor, D, and A_D are not changed between the two frameworks.

domain/task description, but they are still domain dependent. Thus, there is not a clear distinction between the domain/task description and a set of domain-independent social norms. In fact, the lack of explicit representation of the domain independent social norms might hinder the maintenance and the update of the social norms in these systems. Moreover, if a new domain is considered, the domain dependent norms must be rewritten.

In this paper, we address the above mentioned problems, by presenting a framework for planning and execution of social plans, in which social norms are described in a domain and language independent form. We believe that explicitly modeling general high level social norms allows for providing different social behaviors, while accomplishing different tasks, guaranteeing scalability to complex situations and adaptation to different scenarios. Language independent specifications of social norms could then be applied to different reasoning systems (POMDP, HATP, ASP, CBP, etc.) in order to generate social behaviors with different characteristics and to better compare which models/languages are most suited for the realization of cognitive social robots.

III. FRAMEWORK

In this section we describe the framework for explicit modeling of social norms and the corresponding execution model on a social robot.

We first describe the overall architecture depicted in Figure 1. We assume to have a classical planning + execution framework (Fig. 1 a)) that includes: D: a domain description, A_D: the implementation of the actions described in D, a PLANNER module that generates a plan π from the domain description D and an EXECUTOR module that executes the plan π activating the corresponding actions in A_D. Several formalisms for planning and execution can be used and they are not described in this paper.

Starting from such a standard planning + execution framework, we propose its transformation into a *social framework* (Fig. 1 b)), in which the elements of the standard framework are not changed. In particular, D and A_D remain unchanged in the social framework, as well as the PLANNER and the EXECUTOR modules, as long as they provide the requirements for applying the plan transformation algorithm described later in this section. More specifically, in this paper we describe an implementation of this framework in which any planner can be used, while the the EXECUTOR module in based on Petri Net Plans (described below).

The social framework contains some additional components: S: a set of social norms (or social rules), including the definition of social actions, that are domain independent and language independent; M: a set of domain-dependent and language-dependent specifications for applications of social norms S to the domain D; A_S: the implementation of the actions described in S; PNPGENERATOR : a module that generates a new *social plan* (i.e., a plan including social actions), from the plan π generated by the planner and the domain description augmented with the social norms and the application specifications $\langle D, S, M \rangle$.

All the elements in the proposed social framework are described in this section, focusing more on the elements that are specific of the social framework.

A. Domain Description and Planning

In this paper, the domain description D and the PLANNER module have been implemented using Answer Set Programming (ASP) language and solver [8]. The proposed framework does not rely on any particular feature of ASP, so other formalisms for domain description (e.g., PDDL languages) and other planners (e.g., STRIPS) could have been used here. Our solution follows the approach used in the context of a social robot by Erdem et al. [6], handling action descriptions, static axioms, and frame problem in a standard way as described by Lifschitz [9].

The domain description contains static and dynamic knowledge about the capabilities of the robot, the environ-

(meet_person, S_greet)
(speak_to_person, S_face)
(speak_to_person, $S\_display\_text$)
(person_blocking $\wedge$ person_close, *S_stop*)
(person_blocking $\wedge$ person_close, *S_ask_to_pass*)
(crowded, *S_stop*)
(need_help $\wedge\neg$ person_close, *S_approach*)
(approach_person, *S_explain_approach*)
(need_help $\wedge$ person_close, *S_say_please*)
(need_help $\wedge$ person_close, *S_explain_reason_for_help*)
(received_help, *S_thank*)

TABLE I

LIST OF DOMAIN-INDEPENDENT SOCIAL NORMS.

ment, and the task to be accomplished. Static knowledge holds information that is meant to be relatively stable: common sense reasoning assumptions, domain constraints, the definition of all the available actions in A_D (with pre-conditions and post-conditions) and the static information on the environment (map topology and semantics). Dynamic knowledge is generated on the fly by the system and represents the current goal and the sensor fluents characterizing the initial state at plan generation time.

Notice also that, recalling the idea in [10], A_D will include both robot and human actions, where human actions are defined as actions that will be performed by humans to help the robot in achieving its goals. In this way, the PLANNER will compute plans in which the human actions are abstracted and explicitly represented. On the other hand, these actions are then implemented with corresponding robot behaviors (as described later and in more details in [10]), without the need of introducing this complexity in the domain description and in the planning module.

B. Petri Net Plans executor

The EXECUTOR module is implemented through the Petri Net Plans (PNP)[1] execution mechanism described in [11]. The input of the EXECUTOR module is a PNP, that contains the necessary constructs to properly combine actions and conditions. In particular, in the social framework, the PNP to be executed is generated by the PNPGENERATOR module, described later. The PNP engine is responsible for the execution of the plan represented in the PNP by properly activating the actions and checking for the relevant conditions on the basis of the PNP semantics. Moreover, the PNP-ROS bridge allows for an easy integration within ROS applications.

C. Social norms

Representation formalisms for social norms have been largely studied in the field of multi-agent systems These formalisms are typically very powerful to express several complex interactions among agents. In this work, the interaction between the robot and a user is not complex and long, therefore we have chosen a simple representation of social norms, as proposed in [12], based on propositional logic.

Given a propositional logic L defined on a set of atoms Δ, a social norm (as defined in [12]) is represented as a pair $(\phi, \psi) \in L \times L$, with the meaning that if ϕ is true, then ψ is mandatory, or, in other words, $\neg\psi$ is forbidden. In this paper, we consider a variation of this definition that refers to explicit actions that the robot has to do as a consequence of the application of the social norms. So, we consider a set A_S of social actions for the robot and define a social norm as a pair $(\phi, a) \in L \times A_S$, with the meaning that if ϕ is true, then it is mandatory for the robot to execute the action a.

The list of social norms for the robot is thus defined as $S = \{(\phi_1, a_1), \ldots, (\phi_n, a_n)\} \subset L \times A_S$. Some examples of social norms implemented in our system are illustrated in Table I. Different norms would be required for different situations and different configurations of the robot. Social actions are denoted with an initial label S_* to distinguish them from domain actions.

D. Specifications for application of social norms

M is defined as a set of specifications that relate actions and conditions in D and in S. This is needed since the social norms S are domain independent, and it is thus necessary to related the symbols used in S with the ones used in D. This is also needed because of our design goal of not modifying D to accommodate the social norms. Other possible solutions to integrate the social norms S with D would have been: 1) modify D to add elements that are necessary to include S, or 2) make S dependent on D. However, both these alternative solutions have drawbacks: 1) they requires the designer to manually modify D to adapt it to S (or viceversa); 2) they do not allow to use the same set of social norms for different domains and the same domain under different social norms. The solution presented in this paper avoid these two drawbacks, by only requiring to define the specifications M for applying a set of domain independent social norms S to an already available domain description D.

In M, the following types of specifications can be included, in which α is a logical formula over literals from either D or S, p is a literal in either D or S, d is an action in either A_D or A_S, and a_i is an action in A_S.

$\alpha \implies p$	α implies p
$before(d) \implies \alpha$	before executing d, α is true
$after(d) \implies \alpha$	after executing d, α is true
$during(d) \implies \alpha$	during execution of d, α is true
$conflict(d, a_i)$	actions d and a_i are conflicting

The specifications of the first kind are grouped as $M_D \subset M$. As described later, $D \cup M_D$ will be used for evaluating logical formulas in the plan transformation algorithm. In particular, in the implementation described in this paper, these formulas are expressed in ASP.

Some examples of specifications for application of the social norms to our example domain are illustrated in Table II. Althoug these specifications are domain dependent, they can be reused in similar domains (see also Sect. III-G).

E. PNP generator

The main component of our framework is the algorithm described in this section that provides for the automatic generation of the PNP that integrates the plan obtained by planning in the domain and the applications of the

[1] http://pnp.dis.uniroma1.it

human $\Longrightarrow$ $\neg$ person_close
person_close $\Longrightarrow$ meet_person
obstacle $\wedge$ person_close $\Longrightarrow$ person_blocking
many_people $\Longrightarrow$ crowded
before($H\_loadPaper$) $\Longrightarrow$ need_help
before($H\_openDoor$) $\Longrightarrow$ need_help
after($H\_loadPaper$) $\Longrightarrow$ received_help
after($H\_openDoor$) $\Longrightarrow$ received_help
after($S\_approach$) $\Longrightarrow$ person_close
during($S\_approach$) $\Longrightarrow$ approach_person
during(say) $\Longrightarrow$ speak_to_person
conflict($gotoDoor$, S_stop)
conflict($gotoPrinter$, S_stop)

TABLE II

SPECIFICATIONS FOR APPLICATION OF SOCIAL NORMS.

social norms. The PNPGENERATOR module is implemented through Algorithm 1 described here.

The main procedure of the algorithm executes the following steps: 1) generation of a linear PNP from the linear plan π; 2) application of the social norms; 3) transformation of the human actions in PNP instantiating the corresponding templates; 4) new application of the social norms. The $apply\_social\_norms$ procedure is described in the remaining part of the algorithm.

The application of the social norms is repeated at Step 4 in order to consider actions that are introduced at Step 3 for replacing the human actions. In this way, the social norms can be applied both to the human actions (before the transformation, since after that they are replaced) and to the actions introduced by the transformation (that were not present before the transformation).

In this algorithm, we use a queue Q of instances of actions, i.e. of pointers to the beginning of instances of actions in the plan being built. Notice that it is necessary to refer to instances of actions since an action (in particular the social actions) may occur several times in the plan. In practice this is implemented by storing in Q the start place of the PNP action. In the algorithm, $d_i \in Q$ expresses both the particular instance of the action d_i (used in the PNP_* procedures) and its name (used to verify specifications in M).

In the application of the social norms, we use $prec(d_i)$ and $postc(d_i)$ to denote the formulas that describe respectively the preconditions and the postconditions of action d_i, as described in the domain description D. The navigation of the list S in forward order for the 'begin' specifications and in reverse order for the 'after' specifications guarantees that, if multiple actions are applicable to a step in the plan, they are applied in the order they compare in S. Other possible solutions (not discussed in this paper) to consider the order in which specifications must be applied are possible.

The application of the social norms is performed according to the semantics of the specifications for the application of the social norms. Thus, the 'before', 'after', or 'during' specifications add a new social action respectively before, after or in parallel with the current action. This process is better illustrated in the example in the next section. While the 'conflict' specifications add interrupts in the plan to interrupt a domain action when a social norm requires it.

This algorithm uses a set of functions to manipulate PNPs that are described in the following.

Algorithm 1: PNP generator

Input: π: plan generated by the ASP solver,
$\langle D, S, M\rangle$: domain description and social norms
Data: Q: a queue of instances of actions
Output: π_S: PNP integrating π with social actions

1 $\pi_S, Q \leftarrow PNP\_genLinear(\pi)$;
2 $\pi_S \leftarrow apply\_social\_norms(D, S, M, \pi_S, Q)$;
3 $\pi_S, Q \leftarrow PNP\_transformHumanActions(\pi_S)$;
4 $\pi_S \leftarrow apply\_social\_norms(D, S, M, \pi_S, Q)$;
5 **return** π_S ;
6
7 **Procedure** $apply\_social\_norms(D, S, M, \pi_S, Q) : \pi_S$
8 **while** $Q \neq \emptyset$ **do**
9 $d_i \leftarrow$ pop(Q);
10 **foreach** *'*$before(d_i) \Longrightarrow \alpha$*' in* M **do**
11 **foreach** $(\phi_j, a_j) \in S$ *(forward order), s.t.* $D \cup M_D \models \alpha \wedge prec(d_i) \Longrightarrow \phi_j$ **do**
12 $\pi_S \leftarrow PNP\_addBefore(\pi_S, d_i, a_j)$;
13 push(Q, a_j);
14 **foreach** *'*$after(d_i) \Longrightarrow \alpha$*' in* M **do**
15 **foreach** $(\phi_j, a_j) \in S$ *(reverse order), s.t.* $D \cup M_D \models \alpha \wedge postc(d_i) \Longrightarrow \phi_j$ **do**
16 $\pi_S \leftarrow PNP\_addAfter(\pi_S, d_i, a_j)$;
17 push(Q, a_j);
18 **foreach** *'*$during(d_i) \Longrightarrow \alpha$*' in* M **do**
19 **foreach** $(\phi_j, a_j) \in S$, *s.t.* $D \cup M_D \models \alpha \Longrightarrow \phi_j$ **do**
20 $\pi_S \leftarrow PNP\_addParallel(\pi_S, d_i, a_j)$;
21 push(Q, a_j);
22 **foreach** *'*$conflict(d_i, a_j)$*' in* M **do**
23 $B \leftarrow \{d_i\} \cup \{a \in \pi_S | a$ *in parallel with* $d_i\}$;
24 // B = actions in parallel with d_i), $(\phi_j, a_j) \in S$
$\pi_S \leftarrow PNP\_addInterrupt(\pi_S, B, \phi_j, a_j)$;
25 push(Q, a_j);
26 **return** π_S ;

$\pi_S, Q \leftarrow PNP\_genLinear(\pi)$ generates a linear PNP with the sequence of actions in π by applying the sequence operator in PNP and initializes the queue Q with all the instances of the actions generated in π_S.

$\pi_S, Q \leftarrow PNP\_transformHumanActions(\pi_S)$ replaces every human action in π_S with an instantiated template for making human actions executable, as explained in [10]. This template realizes a human robot collaboration scheme driven by the robot in which it waits and establishes an interaction with a human, it asks him/her for help and it waits until s/he perform the task s/he has been asked. This procedure also initializes the queue Q with all the instances of the actions added in π_S.

$\pi_S \leftarrow PNP\_addBefore(\pi_S, d_i, a_j)$ adds action a_j before action d_i using the sequence operator in PNP.

$\pi_S \leftarrow PNP\_addAfter(\pi_S, d_i, a_j)$ adds action a_j after action d_i using the sequence operator in PNP.

$\pi_S \leftarrow PNP\_addParallel(\pi_S, d_i, a_j)$ adds action a_j in parallel to action d_i using the fork-join operators in PNP. If d_i is already in parallel with other actions, a_i will be added to the already existing fork-join construct.

$\pi_S \leftarrow PNP\_addInterrupt(\pi_S, B, \phi_j, a_j)$ adds an interrupt to all the actions in B controlled by the condition ϕ_j, adds the action a_j after the interrupt ϕ_j, adds a transition labeled with $\neg\phi_j$ after the action a_j, and connect this last transition with the node before the fork of the actions in B or to the input place of d_i, if it is the only action in B.

As a result of this process the final plan π_S, represented as a PNP, obtained by the knowledge represented in $\langle D, S, M\rangle$ is sent to the PNP executor. Note that the algorithm may not terminate if there are cyclic specifications. Although it would be possible to detect this situation, in this paper we will not consider this issue assuming acyclic specifications.

F. Action Implementation

A_D and A_S denote the actual implementations of the actions considered in D and in S. Different solutions can be adopted to represent action implementation. Here we use the ROS actionlib[2] specification to implement all the required actions. This choice allows for a direct implementation on the robot using the ROS framework.

G. Portability to other domains

The main feature of the proposed architecture is its portability to other domains. When a new domain is considered, several elements of the current domain can be reused. A domain is generally characterized by a description of the robotic platform, of the environment and of the task to be accomplished. Consequently, $\langle D, S, M\rangle$ contains rules and specifications that refer to all these sub-components. These rules are portable to another domain whenever there is a common intersection in the characteristics of the robot, the environment, or the task. For example, for executing a different task with the same robot in the same environment, all the rules in $\langle D, S, M\rangle$ that are not specific of the task can be reused.

IV. Example

In this section we provide a full example of the use of the proposed framework. We consider a mobile robot with no mechanical arms that has the task of delivering papers output by a printer to a location that is behind a closed door. The task can be accomplished only with the help of humans, since the robot is not able neither to grab papers from the printer nor to open doors.

The domain description D models this problem considering the following available actions: $gotoPrinter$, $gotoDoor$, $passDoor$, $H\_loadPaper$, $H\_openDoor$. H_* actions are human actions, i.e. actions that are executed by humans to help the robot (as described in the previous section). Given this domain description, the ASP planner is able to generate a plan corresponding to the following sequence of actions: $\pi = \langle gotoPrinter, H\_loadPaper, gotoDoor, H\_openDoor, passDoor\rangle$. This plan does not include social actions as the ones described in Section III-C.

Let us now describe how the social norms described in Tables I and II are applied to this plan (line 2). Consider the domain action $H\_loadPaper$, the specification 'before($H\_loadPaper$) $\implies$ need_help' in M, and the social norm '(need_help $\wedge\neg$ person_close, *approach*)' in S (line 11 of the algorithm). Here, the logical entailment $D \cup M_D \models \alpha \wedge prec(d_i) \implies \phi_j$ is true. Indeed the formula 'need_help $\wedge$ human $\implies$ need_help $\wedge\neg$ person_close' can be derived by $D \cup M_D$ (in particular, from the statement in M_D 'human $\implies$ $\neg$ person_close'). Consequently, according to the social norm, the social action $S\_approach$ must be executed before the execution of $H\_loadPaper$ and thus this action is added in the plan π_S and in the queue Q of actions to be processed (lines 12–13). When the algorithm will consider the specification 'after($H\_loadPaper$) $\implies$ received_help' and the social norm '(received_help, S_thank)' (line 15), then the social action S_thank will be added after $H\_loadPaper$ (line 16).

At some later stage of the execution of the algorithm, the action $S\_approach$ will be extracted by the queue (line 9) and processed. Let us consider now the specification 'after($S\_approach$) $\implies$ person_close' in M and the social norm '(meet_person, S_greet)' in S. The logical entailment $D \cup M_D \models \alpha \wedge postc(d_i) \implies \phi_j$ (line 15) is true. Since 'person_close $\implies$ meet_person' is actually present in M_D, the action S_greet is added after $S\_approach$ (line 16). This is repeated also for the social norms '(need_help $\wedge$ person_close, S_say_please)' and '(need_help $\wedge$ person_close, $S\_explain\_reason\_for\_help$)' that are enabled after $S\_approach$ and thus the corresponding social actions are also added after $S\_approach$. As explained before, visiting the social norms in reverse order for the 'after' specifications guarantees to maintain the correct order of the corresponding social actions in the plan. In this case S_greet will precede S_say_please that will precede $S\_explain\_reason\_for\_help$. Similarly, when the specification 'during($S\_approach$) $\implies$ approach_person' in M and the social norm '(approach_person, $S\_explain\_approach$)' in S are considered (line 19), the algorithm will add the action $S\_explain\_approach$ in parallel with $S\_approach$ (line 20). The same process will be performed for applying the social norms to the other actions of the linear plan generated by the ASP planner.

After transforming all the human actions in PNP using the template defined in [10] (line 3), the process of applying the social norms is repeated to consider actions that are introduced by the template (line 4). For example, the action say is in the template of the human actions to ask for help to a human. For all the instances of the action say, the specification 'during(say) $\implies$ speak_to_person' in M enables two social norms in S, '(speak_to_person, S_face)' and '(speak_to_person, $S\_display\_text$)' and thus the corresponding actions S_face and $S\_display\_text$ are added in parallel with say.

At the end of this process, a PNP that implements all of the social norms listed in Tables I and II is generated.

[2] http://wiki.ros.org/actionlib

V. IMPLEMENTATION AND EXPERIMENTS

The framework described in this paper has been fully implemented and tested on a robot executing tasks in our Department. The task used in the experiments is the one illustrated in the previous example.

For this task, the robot is able to navigate in the environment (using standard navigation tools), to detect people close to it (through the analysis of laser scans), to detect if a white paper is placed on a black tray (through an easy image processing procedure), to detect if a door is open (again through laser analysis), and to speak to the persons. At this moment, in order to eliminate any bias due to the performance of the speech understanding system, the utterances of the users were confirmed by an operator.

The users participating in the experiments were not selected from us before, but they were just people passing in the corridor of our Department. The motivation for this choice is that it is likely that a user accepting to participate to the experiment would have been biased towards doing what the robot was asking for. Therefore, users were not instructed about what to do and not even told that they would have been asked to do something by the robot. In this way, we obtained results in a realistic scenario and indeed we experienced that many people did not want to interact with the robot. Since the start of the test was controlled by us, we have tried to avoid repetition of the test with the same person. [3]

A total number of 70 runs during three days of experiments have been executed with different configurations of the social norms. The evaluation of each run has been done by an operator by assigning to it one of the values reported below. The table includes also the percentage of runs for which we obtained this outcome.

A	Successful execution of the task with human help	42 %
B	Unsuccessful execution of the task, but user was willing to help	6 %
C	User answered that s/he was not willing to help	10 %
D	No answer to the robot's request	42 %

These results clearly do not allow for an assessment of the effectiveness of the social norms, which indeed was not a goal of these experiments. Instead the execution all these runs demonstrate the flexibility of our system to easily produce different social behaviors.

Plan	Actions	Places	Transitions	Edges
π	12	37	38	76
π_S	34	123	110	248

As for the complexity of the generated plans, the above table shows the size of the PNP described in the example, before and after the application of the social norms. It is evident that the size (and thus the complexity) of the resulting PNP, being about 3 times bigger, would have made the task of manually writing and maintaining it very ineffective.

This example demonstrates that our system can generate very complex plans and, accordingly, execute complex and robust robot social behaviors. Moreover, our system can automatically generate many variations, by applying different norms, to test different behaviors of the robot. Finally, the system is ready-to-use and non-expert users can easily ran a robot task through a simple GUI.

[3] Videos and implementation details are available in `https://sites.google.com/site/socialrobotplanning/`.

VI. CONCLUSIONS

In this paper we have described a framework for planning and execution of social plans, in which social norms are described in a domain and language independent form. A full implementation of the proposed framework is described and it gave us the possibility of making several tests in a realistic scenario with non-expert and non-recruited users.

The proposed framework allows for extensive user studies, where different social norms are enabled/disabled in order to evaluate their effectiveness for the task under test. This evaluation is certainly an important future direction of our work. Moreover, the application to multiple different scenarios, as well as the implementation of the proposed idea to other cognitive architectures, would allow to extend the approach and to improve in general the development of social robots with explicit representation of social norms.

REFERENCES

[1] T. Fong, I. R. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems*, vol. 42, no. 3-4, pp. 143–166, 2003.

[2] C. Bartneck and J. Forlizzi, "A design-centred framework for social human-robot interaction," in *Proceedings of the Ro-Man2004*, 2004, pp. 591–594.

[3] G. Boella, L. van der Torre, and H. Verhagen, "Introduction to normative multiagent systems," *Computation and Mathematical Organizational Theory*, vol. 12, no. 2–3, pp. 71–79, 2006.

[4] F. Broz, I. R. Nourbakhsh, and R. G. Simmons, "Planning for human-robot interaction using time-state aggregated pomdps," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 2008, pp. 1339–1344.

[5] S. Alili, R. Alami, and V. Montreuil, "A task planner for an autonomous social robot," in *Distributed Autonomous Robotic Systems 8*, 2009, pp. 335–344.

[6] E. Erdem, E. Aker, and V. Patoglu, "Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution," *Intelligen Service Robotics*, 2012.

[7] S. Tomic, F. Pecora, and A. Saffiotti, "Too cool for school - adding social constraints in human aware planning," in *Proc. of 9th International Workshop on Cognitive Robotics*, 2014.

[8] V. Lifschitz, "Answer set programming and plan generation," *Artificial Intelligence*, 2002.

[9] ——, "What is answer set programming?." in *AAAI*, vol. 8, 2008, pp. 1594–1597.

[10] L. Nardi and L. Iocchi, "Representation and execution of social plans through human-robot collaboration," in *Fifth International Conference on Social Robotics (ICSR 2014)*, 2014, pp. 266–275.

[11] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara, "Petri net plans - A framework for collaboration and coordination in multi-robot systems," *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 3, pp. 344–383, 2011.

[12] G. Boella, G. Pigozzi, and L. van der Torre, "Normative framework for normative system change," in *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.