Normative Programs and Normative Mechanism Design (Extended Abstract)

Nils Bulling
Department of Informatics
Clausthal University of Technology
bulling@in.tu-clausthal.de

Mehdi Dastani
Intelligent Systems Group
Utrecht University
mehdi@cs.uu.nl

ABSTRACT

The environment is an essential component of multi-agent systems, which is often used to coordinate the behaviour of individual agents. Recently many programming languages have been proposed to facilitate the implementation of such environments. This extended abstract is motivated by the emerging programming languages that are designed to implement environments in terms of normative concepts such as norms and sanctions. We propose a formal analysis of normative environment programs from a mechanism design perspective. By doing this we aim at relating normative environment programs to mechanism design, setting the stage for studying formal properties of these programs such as whether a set of norms implements a specific social choice function in a specific equilibria.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Modal logic*

General Terms

Theory, Verification, Languages

Keywords

Normative Environment, Mechanism design, Programming Languages

1. INTRODUCTION

The overall objectives of multi-agent systems can be ensured by coordinating the behaviors of individual agents and their interactions. Existing approaches advocate the use of exogenous normative environments and organisational models to regulate the agents' behaviors and interactions [4, 5, 7]. Norm-based environments regulate the behavior of individual agents in terms of norms being enforced by means of regimentation and sanctioning mechanisms. Generally speaking, the social and normative perspective is conceived as a way to make the development and maintenance of multi-agent systems easier to manage, e.g., AMELI [4] and $\mathcal{M}$oise$^+$ [6].

This extended abstract departs from the normative environment programming perspective and proposes a formal analysis of normative environment programs by relating them to concurrent game structures (a well-known model used for modelling multi-agent systems) [2] and mechanism design. In our view, normative environment programs can be modelled as concurrent game structures where possible paths in game structures denote possible execution traces of the corresponding normative environment programs. This relation would set the stage for studying formal properties of normative environments such as whether a set of norms implements specific choice functions in specific equilibria. This also allows, for example, to analyse whether groups of agents are willing to obey the rules specified by a normative system. Such a formal analysis is closely related to the work presented in [1, 11], where norms are modelled by the deactivation of transitions, and the work presented in [9, 10], where social laws were proposed to be used in computer science to control agents' behaviours.

Cite as: Normative Programs and Normative Mechanism Design (Extended Abstract), Nils Bulling and Mehdi Dastani, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 1187-1188.

2. NORMATIVE PROGRAMS

The general setting of our programming framework is as follows. A normative multi-agent program consists of a normative environment program and a set of agents programs that when executed perform actions in the normative environment. In this framework, the programmed agents may or may not have access to the specified norms in the environment, their actions are performed simultaneously, and the actions' outcomes are determined by the normative environment programs.

We are interested in programming languages which are designed to implement normative environments in terms of norms and sanctions. These languages often provide programming constructs to specify 1) the (initial) state of an environment, 2) the outcomes of the agents' actions, and 3) norms and sanctions. The interpreter of such languages is based on a cyclic process that continuously monitors the agents' (observable) actions, determines the outcome of the actions, and imposes norms and sanction if necessary. Intuitively, the performance of agents' actions will change the environment state and possibly cause a violation of some specified norms. Imposing sanctions may in turn modify the environment state, which can be considered as a way to bring the violated state of the environment back to an optimal one. It is important to note that possible executions of a normative environment program depend on the agents'

actions and the interpreter of a normative environment programming language which selects an execution path among all possible ones. In order to relate the execution models of such normative environment programs to mechanism design and study their formal properties, the normative environment programming languages are required to have formal (operational) semantics. A candidate for a such a normative environment programming language is 2OPL [3].

3. NORMATIVE MECHANISM DESIGN

We propose to use *concurrent game structures* [2] as abstraction and as a formal model of normative environment programs. In such models it is assumed that all agents execute their actions synchronously. A combination of actions together with the current environment state determines the next state of the environment. An environment and a concurrent game structure are considered equivalent if the set of environment program executions coincides with the set of paths in the concurrent game structure. As program executions and paths are considered as the semantics of both normative environments and concurrent game structures, it is very natural to consider agents' preferences as relations on the sets of executions. In this way, an agent prefers some executions over others.

In social choice theory a *social choice function* assigns outcomes to given preference profiles (cf. e.g.[8]), where a preference profile consists of one preference for each participating agent. The task of a social choice function is to determine an outcome with respect to the preference profile. Various natural requirements are imposed on the social choice function in order to ensure e.g. fairness.

Mechanism design is concerned with creating a protocol or a set of standards for behaviours such that the outcome agrees with a social choice function provided that agents behave rationally–in some sense–according to their preferences. In game theoretic terms *behaving rationally* means to act according to some *solution concept* (e.g. the concept of Nash equilibria). If such a mechanism exists it is said that the mechanism implements the social choice function in an equilibrium (e.g., Nash equilibrium).

We define a *normative behaviour function* as a social choice function that assigns a set of "desired" environment executions to each preference profile. We refer to the outcomes as the *normative behaviours wrt a specific preference profile*. As a consequence, the aim of *normative mechanism design* is to come up with a *normative mechanism* or a *normative environment program* which imposes norms and sanctions based on the performed agents' actions such that agents–again following some rationality criterion according to their preferences–behave in such a way that the system executions stay within the normative outcome. Given an environment program and its corresponding concurrent game structure we are interested in the following question: Can we specify a set of norms and sanctions such that extending the environment programs with the norms and sanctions implements a normative behaviour function in an equilibrium (e.g. dominant or Nash)?

As said before, our work is closely related to [1, 11]. In the former, labelled Kripke structures are considered as models supposing that each agent controls some transitions. A norm is then considered as the deactivation of specific transitions. The main difference to our work is that adding norms and sanctions to an environment program in our framework can also "activate" new transitions in the underlying environment execution model. This is because the activation of transitions in our framework does depend on actions' pre- and postconditions.

4. CONCLUSIONS

In this extended abstract we are proposing normative mechanism design as a formal tool for analysing normative environment programs. We have argued how one can abstract from such programs and then apply methods from mechanism design to verify whether the restrictions imposed on the program agree with the behaviour the designer expects. More precisely, we have introduced normative behaviour functions for representing the "ideal" behaviour of the system with respect to different sets of agents' preferences. The latter has enabled us to apply concepts from game theory to identify agents' rational behaviour. These ideas can now be used to verify whether a programmed normative environment is sufficient to motivate agents to act in such a way that the behaviour described by the normative behaviour function is met.

5. REFERENCES

[1] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Normative system games. In *Proceedings of the AAMAS '07*, pages 1–8, New York, NY, USA, 2007. ACM.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.

[3] M. Dastani, D. Grossi, J.-J. Ch. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *Proceedings of KRAMAS 2008*, volume LNAI 5605, pages 16–31. Springer, 2009.

[4] M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, and J.L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of AAMAS 2004*, pages 236–243, New York, US, July 2004.

[5] D. Grossi. *Designing Invisible Handcuffs*. PhD thesis, Utrecht University, SIKS, 2007.

[6] J. F. Hübner, J. S. Sichman, and O. Boissier. $\mathcal{M}$oise$^+$: Towards a structural functional and deontic model for mas organization. In *Proceedings of AAMAS 2002*, pages 501–502. ACM, July 2002.

[7] A. J. I. Jones and M. Sergot. On the characterization of law and computer systems. In J.-J. Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. John Wiley & Sons, 1993.

[8] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[9] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings AAAI-92*, San Diego, CA, 1992.

[10] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[11] W. van der Hoek, M. Roberts, and M. Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis, 2007.