A Verification Framework for Normative Multi-Agent Systems

Lăcrămioara Aştêfânoaei¹, Mehdi Dastani², John-Jules Meyer², and Frank S. de Boer¹

¹ CWI, Amsterdam, The Netherlands
² Universiteit Utrecht, The Netherlands

Abstract. This paper presents a programming language that facilitates the implementation of coordination artifacts which in turn can be used to regulate the behaviour of individual agents. The programming language provides constructs inspired by social and organisational concepts. The operational semantics of the language is prototyped in Maude, a rewrite logic software. Properties of the coordination artifacts are model-checked with the Maude LTL model-checker.

1 Introduction

One of the challenges in the design and development of multi-agent systems is to coordinate and control the behaviour of individual agents. Some approaches aim at achieving this objective by means of exogenous coordination artifacts, which are designed and built in terms of concepts such as action synchronisation and resource access relation [12]. Other approaches advocate the use of social and organisational concepts (e.g., norms, roles, groups, responsibility) and mechanisms (monitoring agents’ actions and sanctioning mechanisms) to organise and control the behaviour of individual agents [3]. Yet other approaches aim at combining these by proposing organisation-based coordination artifacts, i.e., coordination artifacts that are designed and developed in terms of social and organisational concepts [4,5]. In such combined approaches, a multi-agent system is designed and developed in terms of an organisation artifact and the constituting individual agents. In order to ensure that the developed multi-agent systems achieve their overall design objectives and satisfy some global desirable properties, one has to verify the organisation artifact that constitutes the coordination and control part of the multi-agent system.

In this paper, we present a verification framework for normative multi-agent systems in which individual agents are coordinated and controlled by norm-based organisation artifacts. Such artifacts refer to norms as a way to signal when violations take place and sanctions as a way to respond (by means of punishments) in the case of violations. Basically, a norm-based artifact observes the actions performed by the individual agents, determines their effects in the environment (which is shared by all individual agents), determines the violations caused by performing the actions, and possibly, imposes sanctions.

© Springer-Verlag Berlin Heidelberg 2008
We first present a programming language that is designed to facilitate the implementation of norm-based organisation artifacts. The operational semantics of the language makes it easy to prototype it in Maude [6], a rewriting logic software. Rewriting logic can be used as a computational framework for modular programming language design and formal analysis [7]. It has the benefit that there is no gap between semantics and implementation. This has a great advantage over the situations where one needs to implement an interpreter in order to execute the semantics of the designed language. It is also the case that rewriting logic implementations like Maude offer prototype parsers for free since they support user-definable syntax. Another benefit is rapid prototyping of programming language designs. This makes it easier to experiment with new language constructions since one needs only to define and not also to implement them. Furthermore, rewriting logic offers a suite of generic tools for formal analysis, for instance, the Maude theorem prover and LTL model-checker, which can be used to prove properties of the language definitions.

2 Programming Normative Multi-Agent Systems

In this section, we present a programming language that facilitates the implementation of normative multi-agent systems. Individual agents are assumed to be implemented in a programming language, not necessarily known to the multi-agent system programmer, who is assumed to have a reference to the (executable) programme of each agent. Most noticeably, it is not assumed that the agents are able to reason about the norms of the system since we do not make any assumptions about the internals of individual agents. Agents perform their actions in an external environment which is part of and controlled by the organisation. Actions are assigned pre- and post-conditions. If the pre-condition holds in the current state of the environment (the execution of an action is enabled), then the multi-agent system organisation determines the effect of the action by updating the state with the facts which represent the post-condition. We consider norms as being represented by counts-as rules [8], which ascribe "institutional facts" (e.g. "a violation has occurred") to "brute facts" (e.g. "the agent is in the train without a ticket"). In our framework, brute facts constitute the factual state of the multi-agent organisation, which is represented by the environment (initially set by the programmer), while institutional facts constitute the normative state of the multi-agent organisation. The institutional facts are used with the explicit aim of triggering system’s reactions (e.g. sanctions). Sanction rules determine what brute facts will be brought about by the system as a consequence of normative facts. Typically, such brute facts are sanctions, such as fines.

2.1 Syntax

In order to represent brute and institutional facts in our normative multi-agent system programming language, we introduce two disjoint sets of first-order atoms <b-atoms> and <i-atoms> to denote these facts. Moreover, we use <ident> to