Computational Models for Normative Multi-Agent Systems

Natasha Alechina[1], Nick Bassiliades[2], Mehdi Dastani[3], Marina De Vos[4], Brian Logan[1], Sergio Mera[5], Andreasa Morris-Martin[6], and Fernando Schapachnik[5]

1 **School of Computer Science, University of Nottingham**
Nottingham, NG8 1BB, UK
{nza,bsl}@cs.nott.ac.uk
2 **Dept. of Informatics, Aristotle University of Thessaloniki**
54124 Thessaloniki, Greece
nbassili@csd.auth.gr
3 **Dept of Information and Computer Sciences, University of Utrecht**
Princetonplein 5, 3584CC Utrecht, The Netherlands
M.M.Dastani@uu.nl
4 **Dept. of Computer Science, University of Bath**
Bath, BA2 7AY, UK
mdv@cs.bath.ac.uk
5 **Departamento de Computación, FCEyN, Universidad de Buenos Aires, Intendente Güiraldes 2160 – Ciudad Universitaria – C1428EGA, Buenos Aires, Argentina**
{smera,fschapachnik}@dc.uba.ar
6 **Dept. of Computer Science, University of Guyana**
Turkeyen, Greater Georgetown, Guyana
andreasa.morris@uog.edu.gy

Abstract

This chapter takes a closer look at computational logic approaches for the design, verification and the implementation of normative multi-agent systems. After a short overview of existing formalisms, architectures and implementation languages, an overview of current research challenges is provided.

1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence, I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Norm verification, Computational Architectures for Normative MAS, Programming Normative Systems

Digital Object Identifier 10.4230/DFU.Vol4.12111.71

1 Introduction

Multi-agent systems consist of interacting autonomous agents. While individual agents aim at achieving their own design objectives, the overall objectives of multi-agent systems can be guaranteed by regulating and organising the behaviour of individual agents and their interactions. There have been various proposals for regulating and organising the behaviours of individual agents. Some of these proposals advocate the use of coordination artifacts that are specified in terms of low-level coordination concepts such as synchronization [5, 28]. Other approaches are motivated by organisational models, normative systems, or electronic

Normative Multi-Agent Systems. *Dagstuhl Follow-Ups, Vol. 4.* ISBN 978-3-939897-51-4.
Editors: Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leendert W.N. van der Torre; pp. 71–92

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

institutions [35, 36, 49, 57, 62, 31]. These approaches have resulted in what is often called normative multi-agent systems. In such systems, the behaviours of individual agents are regulated by means of norms and organisational rules that are either used by individual agents to decide how to behave, or are enforced or regimented through monitoring and sanctioning mechanisms. In these systems, a social and normative perspective is conceived as a way to make the development and maintenance of multi-agent systems easier to manage. A plethora of social concepts (e.g., roles, groups, social structures, organisations, institutions, norms) has been introduced in multi-agent system methodologies (e.g. Gaia [88]), models (e.g. OperA [34]), specification and modelling languages (e.g. S-MOISE+ [58], INSTAL[25] and ISLANDER [36]) and computational frameworks (e.g. AMELI [35]).

With the significant advances in the area of autonomous agents and multi-agent systems in the last decade, promising technologies for the development and engineering of normative multi-agent systems have emerged. The result is a variety of programming languages, execution platforms, and computational tools and techniques that facilitate the development and engineering of normative multi-agent systems. This chapter provides an overview of computational tools that can be used to develop and build normative multi-agent systems. The main themes of this chapter are the languages for representing deontic notions (such as norms themselves, conflicts of norms, violations of norms, etc.) and the algorithms to implement tools capable of analyzing and verifying norms, and to implement normative system platforms capable of e.g. monitoring norm violations, and implementing agents capable of deliberating about norms.

2 Background

Computational models for norms are essential for the development of normative multi-agent systems. Such models can be used to specify and verify normative multi-agent systems, to implement normative multi-agent systems, or to allow software agents, electronic institutions and organisations to reason about norms at run-time. In this section, we provide an overview of the existing computational models of norms and how they are applied in multi-agent systems.

2.1 Norm Specification and Verification

This section focuses on the use of logic languages and automated reasoning to describe and understand the behavior of normative systems and analyze their properties. Coherence, in the sense of absense of conflicts, is one of the key properties of interest. Under this approach the object under study is typically the normative system itself. This means that the type of questions we can aim at answering using logic machinery are less related to particular executions of a set of normative agents (which is the focus of sections 2.2 and 2.3) and they are closer to understanding the behavior of universal properties or interactions within the system. It also involves being able to provide a mathematical proof of such properties (a proof whose extent, depending on the language and automated methods used, can cover a fragment of the models under consideration or the whole class of them).

Let's provide an example of the type of properties we mentioned and consider a legislator that is drafting a normative system that regulates the traffic of a city. During this process she can pose a series of very natural questions related to the quality of the normative system. For example, are these rules going to force a driver to, at the same time, grant and deny the right-of-way to another vehicle? Can a pedestrian and a vehicle be allowed to cross an intersection simultaneously, and therefore cause an accident? Is the system applicable to

all type of vehicles, or is there a "gap" that makes, for example, the speed limit not to be enforced on motorcycles?

The type of questions mentioned above involve dealing with the concept of obligation, permission and prohibition. *Deontic logic* is the field of logic concerned about these concepts and the formalisms we are going to talk about in this section naturally fall under this category. Other related concepts such as right, liberty, power, immunity, etc. can also be included in this definition. Legal theories, at least since Hohfeld in 1913 [54], logic-based approaches like [69] and more recently [80, 81, 75] have attempted to clarify these concepts.

The primal property that is subject of verification is what is known as *coherence*: whenever the set of rules is "contradictory" in any sense, sometimes referred to as "absence of conflicts". As stated in the literature (e.g., [52]), the problem cannot be simply reduced to logical consistency. To exemplify this, suppose you have a norm stating that it is forbidden to cross red lights, and another that says that it is forbidden but paying a fine is way of "fixing" it. Is there a conflict or not? Another example is a rule that imposes an obligation and a certain reparation as a way of "fixing" things if the obligation is not fulfilled. If another rule prohibits this same action, then there is another potential conflict.

The importance of absence of conflicts in a normative system depends of course on the context, given that different legal instruments operate at different levels of abstraction. Constitutions and parliamentary bills are often very general and their ambiguity is a necessary price to get enough consensus for their approval. However, as we descend the legal hierarchy we find everyday regulations having other requirements: they tend to be operational, describing how a set of actors (or agents) should behave in a daily manner. Ambiguities and gaps are usually a problem. More important, they are meant to be interpreted by common individuals, not specifically trained in Law, and it is seldom the case where conflicts that arise from them are resolved in a Court of Law. This does not mean that they are conflict-free. On the contrary, an important amount of time and effort is wasted in trying to overcome the deficiencies of everyday regulations. This is one place where doing logical verification of norms can add real value.

Much work has been done in exploring deontic-type logics and trying to cope with conflict detection. Many different languages have been proposed with motley theoretical and computational properties. There is no "one-size-fits-all" formalism, and there are many aspects (like complexity, expressiveness, conciseness, a language natural to the context of use, etc.) that live in opposite sides of the scales. Considering the strong computational flavor of this work, what follows aims at presenting the state-of-the-art formalisms currently being developed by the community, focusing on the ones that incorporated such concepts into working tools. There is much valuable work on the theoretical or philosophical side of deontic logic, but this falls outside the scope of this section. Besides a first generation of efforts that were aimed at specific normative sets, somehow hard-coded into the tools (e.g., [50, 20]), we describe what is considered as the current generation of practical formalisms.

The authors in [51] propose a language that deals with automated conflict detection in norms by using a tool that supports ontologies and translates normative propositions into a Prolog program. However, the analysis is restricted to logical contradiction, which has the limitations we already mentioned. In order to describe a real system, where rule interactions are naturally more involved, being able to deal with a more general concept such as coherence is a must.

BCL [46, 47] is a contract specification language based on defeasible logic that is meant for monitoring, allows to build executable versions, can detect conflicts among rules off-line and provides features like clause normalisation. Recently it incorporated support for some

temporal reasoning. On the other hand, it does not offer support for specifying background theories, which are very useful for restricting the class of models under consideration and greatly contribute to improve reasoning performance.

$\mathcal{CL}$ [78, 37] is another logical language based on dynamic logic that treats deontic operators as first-class citizens and has support for conditional obligations, permissions and prohibitions, as well as for (nested) contrary-to-duties and contrary-to-prohibitions. It provides its own coherence checker, CLAN [38]. Contextual information has to be encoded as deontic rules. Being based on dynamic logic, certain predicates are hard to express: for instance, it is easy to say that if a book is borrowed, a book should eventually be returned, but it would be difficult to formalize the borrowing of multiple books and their corresponding returning; the correct version being pretty involved.

`FL` [44, 45] is also a language and companion tool. The language is a wrapper for LTL with some features such as variables and fluents (called *intervals*) to ease expressions. Its main characteristic is that it uses existing LTL model checkers like SPIN [56], DiViNE [14] or NuSMV [23] to do the verification. Authors claim that as there are strong similarities between temporal software specifications and vast amounts of norms, it makes sense to reutilise tools that are being developed and optimised since at least a couple of decades instead of building new ones. On the other hand, it doesn't support deontic operators as first class citizens and somehow entails a notion of total consistency.

All of these tools face the challenge of scale: they have been tested with somehow realistic albeit small case studies. What is needed to cope with work load from real users?

For instance, there is the theoretical question of which complexity class are they in. Experience from other fields suggests this is not enough, as sometimes many problems become practically tractable although their theoretical worst case is not. What can be said about the analysis of norms? In other words, what is their "practical complexity"?

If it is about computing real cases, expressivity also comes into play. Can we say everything that is needed? Is there a simple way or contrived expressions cannot be avoided?

More importantly, a right balance between expressivity and complexity has to be achieved. What other fields have done to achieve the same goal seems promising in this case too. Build usable tools, apply them to real cases (also let others use them), understand what is lacking and what is overflowing, revise the underlying theory, repeat the cycle.

2.2 Computational Architectures for Normative Multi-Agent Systems

In the literature on multi-agent systems, there have been many proposals for specification languages and logics to specify and reason about normative multi-agent systems, virtual organisations, and electronic institutions (e.g., [62, 77, 17, 1]). How to specify, develop, program, and execute such normative systems was one of the central themes that were discussed and promoted during the AgentLink technical fora on programming multi-agent systems (see [30, 29] for the general report of these technical fora). In this section we focus on normative architectures; formalisms that were initially created to support a designer in specifying and verifying normative systems. The next section discusses the middleware and the programming languages needed for running normative systems. We only briefly discuss the ISLANDER and $\mathcal{M}$OISE$^+$ architectures in this section as they have a stronger emphasis on the programming part. They will be discussed in more detail in the next section.

All architectures in this section start with the assumption that the norms of the systems are centrally maintained. The agents themselves can be norm-aware or not. We will start with the more formal and logic-based frameworks before proceeding to the more pragmatic models.

OPERA [34, 72] is founded on the assumption that multi-agent systems imitate human societies. It therefore adopts an organisational structure for its norm modelling. It originates from the idea that agent societies cannot depend on individual agents, all with their own goals, to bring about the society's goals without a framework that specifically enforces this. The goals and norms of the society must be specified, independently of the participating agents since there is no guarantee that the agents will have a desire to achieve the society's goals. This also assures the autonomy of individual participating agents. Each of them can decide to ignore this organisation goal or even rebel against it. Agents are associated with roles or groups of roles within the organisation. The norms are defined on the roles rather than on the agents themselves. With each role an agent assumes, the agent will receive a set of responsibilities and capabilities. The norms are defined in order to allow the organisation to achieve its goals, irrespective of the agents' individual goals. The achievement of the organisation goals is directed by means of landmarks, i.e. states to be achieved, and scenes, a roadmap landmarks . These concepts are formalised using first order logic, allowing for verification of various properties, like for example "is it possible for the organisation to achieve its goals despite violation of the norms by some agents".

OperettA is an open-source tool that provides an organisation-oriented development environment. It was developed in order to support developers in designing and maintaining organization models based mainly on the OPERA framework. It provides specialised editors for all the main components of OPERA's operational model. The model once completed can be viewed as XML data. Additionally, OperettA supports the verification and simulation of the OPERA models.

The INSTAL framework [24] is a normative framework architecture with a formal mathematical model to specify, verify and reason about the norms that govern an open distributed system. The INSTAL approach has opted for an event-driven institutional approach: the norms are expressed on the events/actions of the participants rather than the normative state. Deviation from a norm results in a violation. The premise of the model is that events trigger the creation of institutional fluents. Inspired by Jones and Sergot's [63] account of institutional power and the notion of 'counts-as', the generation relation is used to explain the connection between actions and their interpretation in the context of the institution. The effects of events, actions or institutional events – in terms of the initiation or termination of brute facts [61] and institutional fluents – is described by the consequence relation. Thus, given an event and a state of the institutional model, represented as a set of (institutional) fluents, the next state can be determined by the transitive closure of the generation relation and the consequence relation. The system was extended to be able to deal with interacting normative systems [25].

The formal model and semantics is translated in an equivalent logic program under the answer set semantics [13]. Answer set programming is a declarative programming paradigm with an operational semantics. To support the designer, an associated action language, INSTAL, is provided. Designing a set of norms can be an erroneous process. To support the designer, an inductive logic programming system [26] was developed. On the of a set desired outcomes, also called use-cases, it revises the current specification to take these outcomes into account.

OCeAN (Ontology CommitmEnts Authorizations Norms) [39] is a metamodel that can be used to specify electronic institutions. Those institutions, thanks to a process of contextualization in a specific application domain, can be used in the design of different open systems by enabling the interaction of autonomous agents.

The fundamental concepts of this model, which need to be specified when designing

electronic institutions are: (i) an ontology for the definition of the concepts used in the communication and in the specification of the rules of the interaction; (ii) the definition of the events, the actions, and the institutional actions and events that may happen or can be used in the interaction among agents; (iii) the definition of the roles that the agents may play; (iv) an agent communication language (ACL) for the exchange of messages; (v) the definition of the institutional powers for the actual performance of institutional actions; (vi) a set of norms for the specification of obligations, prohibitions, and permissions.

This model has been formalized using the Discrete Event Calculus (DEC) [39], which is a version of the Event Calculus. Recently some parts of this model have been formalized using Semantic Web technologies [41, 40].

One of the early specification tools of multi-agent systems in terms of institutional rules and norms is ISLANDER [36] based on the concepts of electronic institutions as proposed in [74]. The same concepts are the foundation for the INST*AL* and OCEAN architectures discussed earlier. In ISLANDER institutions are constructed by means of a graphical user interface. The framework associates roles to agents. The roles offer standardised patterns of behaviour and agents can assume various roles when interacting with the electronic institutions. Scenes describe the normative paths that agents in their respective roles need to follow; the scene and the role describe what an agent can do, is permitted or obligated to do at any given time. Agent's behaviour is further restricted by normative rules, dictating which actions are permitted and which ones are not. The key aspect of ISLANDER approach is that norms can never be violated by the agents. A simulation tool SIMDEI [6] is provided for to animate and verify the institution before deployment.

$\mathcal{M}$OISE$^+$ [60] is a framework for specification by means of social and organisational concepts. This modelling language can be used to specify multi-agent systems through three organisational dimensions: structural, functional, and deontic.

The systems presented in this section fall into two categories. The first group consists of OPER*A* and $\mathcal{M}$OISE$^+$. Both systems take an organisational approach and express high-level norms concerning a state of the system/organisation. The other architectures, INST*AL*, ISLANDER, OCEAN and [42, 82], take their inspiration from institutions where the norms are expressed at level of the actions of the participants. These are referred to as low-level norms. High-level norms can be used to represent *what* the agents should establish — in terms of a declarative description of a system state — rather than specifying *how* they should establish it. So far, to our knowledge, there is no system that allows for the specification of both high and low level norms.

All systems offer a variety of social, normative and organisational constructs to make the specification of the normative system easier. INST*AL* offers the least of them, providing a more concise model at the expense of the designer having to hard-code these constructs.

Of all the systems described above $\mathcal{M}$OISE$^+$ is the only architecture that is not grounded in a logical/mathematical formalism. This implies that the soundness and properties of the normative system cannot be analysed through formal analyses and verification mechanisms.

2.3 Programming Normative Multi-Agent Systems

The development of normative multi-agent systems requires programming languages that provide constructs to implement social and organisational concepts and abstractions in order to regulate the behaviour of individual agents. There are two ways to regulate the behavior of individual agents by means of such concepts. First, the programming languages for individual agents can be extended with constructs that allow the representation and reasoning about social and organisational concepts. Such constructs should allow multi-agent programmers

to implement agents that make their decisions not only based on their individual goals and beliefs, but also based on the existing social and organisational rules. The idea is that individual agents can be implemented in terms of cognitive and social abstractions such that their behaviours are determined upon reasoning about such abstractions.

Second, one can regulate the behaviour of individual agents exogenously by means of a program external to individual agent programs. The implementation of exogenous mechanisms requires abilities to monitor and control the behaviours of individual agents. The idea is to have an external organisation component that is able to monitor, evaluate and control the behaviour of individual agents. The question is what should be monitored and how the agents' behaviours can be evaluated and influenced. As the internal structure of individual agents cannot be assumed in general, their external behaviours (i.e., communication and interaction with the environment) are the only controllable entities. The organisation software entity can thus observe agents' external behaviour, evaluate them based on social and organisational rules, and determine how to respond to such behaviour. For example, if the organisation specification disallows certain agents to interact, then the organisation component should be able to block such interactions. This suggests that the agents' actions (e.g., communication, environment actions including sense actions) should be processed and managed through the external organisation component, i.e., the organisation component intermediates the interaction between agents as well as the interaction between agents and the environment.

In the previous section, we looked at different approaches to specify and verify normative systems. In this section, we focus on the implementation and the deployment of normative systems through the architectures or programming languages previously mentioned. We start with systems were norms are externalised before looking at programming normative agents.

2.3.1 Programming Normative Organisations

One of the early modelling languages for specifying institutions in terms of institutional rules and norms is ISLANDER [36]. In order to interpret institution specifications and execute them, a computational platform, called AMELI [35], has been developed. This platform implements an infrastructure that, on the one hand, facilitates agent participation within the institutional environment and supports their communication and, on the other hand, enforces the institutional rules and norms as specified in the institutional specification. The key aspect of ISLANDER/AMELI is that norms can never be violated by the agents. In other words, systems programmed via ISLANDER/AMELI make only use of regimentation in order to guarantee the norms to be actually followed. Another characteristic of this proposal is that norms concern communication actions and express which communication actions are permitted, obliged or forbidden.

A related modelling language is $\mathcal{M}$OISE$^+$ [60]. In contrast to ISLANDER [36], $\mathcal{M}$OISE$^+$ focuses more on organisational structures and specifies multi-agent systems through three organisational dimensions: structural, functional, and deontic. Another difference between $\mathcal{M}$OISE$^+$ and ISLANDER/AMELI is that the first is concerned with more high-level norms pertaining to declarative descriptions of the system while the latter considers norms as low-level procedures that directly refer to actions. Various programming frameworks have been proposed to implement and execute $\mathcal{M}$OISE$^+$ specifications. For example, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [58] is an organisational middleware that provides agents access to the communication layer and the current state of the specified organisation. Another characteristic feature of this middleware is that it allows agents to change the organisation and its specification, as long as such changes do not violate organisational constraints. In the artifact version of this framework,

ORG4MAS, various organisational artifacts are used to implement specific components of an organisation such as group and goal schema. In this framework, a special artifact, called reputation artifact, is introduced to manage the enforcement of the norms.

Like ISLANDER/AMELI, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ does not allow agents to violate organisational rules and norms, i.e., norms are regimented rather than being enforced by sanctions. In the artifact version of this framework, ORG4MAS, the enforcement of norms is assumed to be managed indirectly through a reputation mechanism, but it remains unclear how such a reputation system realizes sanctioning. Moreover, both AMELI and $\mathcal{S}$-$\mathcal{M}$OISE$^+$ lack a complete operational semantics that captures all aspects of normative systems, including the enforcement of norms. An explicit formal and operational treatment of norm enforcement is essential for a thorough understanding and analysis of computational frameworks of normative multi-agent systems. A great contribution of ISLANDER/AMELI and $\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$ is the repertoire of social and organisational concepts, which support high-level specification of multi-agent organisations.

In addition to the above approaches, there are some proposals that advocate the use of standard technologies for programming organisations and institutions, e.g., `powerJava` [11] and `powerJade` [10]. These proposals are designed to implement institutions in terms of roles. While `powerJava` extends Java with programming constructs to implement institutions, `powerJade` proposes similar extensions to the Jade framework. Like AMELI, $\mathcal{S}$-$\mathcal{M}$OISE$^+$, and ORG4MAS, these programming frameworks consider an institution as an exogenous coordination mechanism that manages the interactions between participating computational entities (objects in `powerJava` and agents in `powerJade`). However, unlike AMELI, $\mathcal{S}$-$\mathcal{M}$OISE$^+$, and ORG4MAS, the proposed programming frameworks provide only a limited set of social concepts such as role and power. A role is defined in the context of an institution (e.g., a student role is defined in the context of a school) and encapsulates capabilities, also called powers, that its players can use to interact with the institution and with other roles in the institution (e.g., a student can participate in an exam). For an object or an agent to play a role in an institution in order to gain specific abilities, they should satisfy specific requirements as well. In `powerJava` roles and organisations are implemented as Java classes. In particular, a role within an institution is implemented as an inner class of the class that implements the organisation. Moreover, the powers that a player of a role gains and the requirements that the player of the role should satisfy are implemented as methods of the class that implements the role. In `powerJade`, organisations, roles and players are implemented as subclasses of the Jade agent class. The powers that the player of a role gains and the requirements that a player of a role should satisfy are implemented as Jade behaviours (associated to the role).

A dedicated programming language that supports the implementation of normative multi-agent organisations is 2OPL (Organisation Oriented Programming) [31, 86]. Similar to the abovementioned computational frameworks, this programming framework considers an organisation as a software entity that exogenously coordinates the interaction between agents and their shared environment. In particular, the organisation is a software entity that manages the interaction between the agents themselves and between agents and the shared environment. 2OPL provides programming constructs to specify 1) the initial state of an organisation, 2) the effects of agents' actions in the shared environment, and 3) the applicable norms and sanctions. In contrast to AMELI, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ and ORG4MAS, norms in 2OPL can be either enforced by means of sanctions or regimented. In the first case, agents are allowed to violate norms after which sanctions are imposed. In the second case, norms are considered as constraints that cannot be violated. The enforcement of norms by sanctions is

a way to guarantee higher autonomy for the agents and higher flexibility for the multi-agent system. The interpreter of 2OPL is based on a cyclic control process. At each cycle, the observable actions of the individual agents (i.e., communication and environment actions) are monitored, the effects of the actions are determined, and norms and sanction are imposed if necessary. Comparing to other approaches, 2OPL has the advantage that it comes with a complete operational semantics such that normative organisation programs can be formally analysed by means of verification techniques [9]. This organisation oriented programming language is extended with programming constructs that support the implementation of concepts such as obligation, permission, prohibition, deadline, norm change, and conditional norm [86, 84, 85].

Rule Responder [76] is an open source framework for creating virtual organizations as multi-agent systems that support collaborative rule-based agent networks on the Semantic Web, where independent agents engage in conversations by exchanging event messages and cooperate to achieve (collaborative) goals. Rule Responder agents communicate in conversations that allow implementing different agent coordination and negotiation protocols. It utilizes messaging reaction rules from Reaction RuleML for communication between the distributed agent inference services, employing various semantically annotated primitives, such as speech acts, or message content information, such as deontic organizational norms, purposes or goals and values etc., which are represented as ontological concepts, using RDF Schema or OWL. The (semi-)autonomous agents use rule engines and Semantic Web rules to describe and execute derivation and reaction logic which declaratively implements the organizational semiotics and the different distributed system/agent topologies with their negotiation/coordination mechanisms. The supported reasoning engines, with their languages, in Rule Responder are, among others, Prova [66] with a Prolog-like language, OO jDREW [12] with RuleML/POSL, and DR-Device [15] with a defeasible logic rule language. Compared to the systems and frameworks presented above, Rule Responder provides the necessary flexible infrastructure to build normative organizations, being agnostic to the modelling/programming language that is used to implement social and organisational concepts. Any of the reasoning engines supported by Rule Responder can be used in an adhoc manner for this purpose. One such example, presented in the next subsection, is DR-DEVICE-M [65]. Furthermore, Rule Responder is extensible enough to allow the incorporation of more specific-to-the task normative reasoning engines, such as the ones presented throughout this section.

2.3.2 Programming Norm Aware Agents

Normative programming frameworks and middleware to support the development of normative multi-agent organisations are often designed to inter-operate with existing BDI-based agent programming languages, and there has been considerable recent work on approaches to programming agents which operate as part of a normative system.

For example, $\mathcal{J}$-$\mathcal{M}$OISE$^+$ [60] is designed to inter-operate with the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [58] middleware and allows Jason [18] agents to access and update the state of an $\mathcal{S}$-$\mathcal{M}$OISE$^+$ organization. Similarly, the JaCaMo programming framework combines the Jason, Cartago [79], and $\mathcal{S}$-$\mathcal{M}$OISE$^+$ platforms. In JaCaMo, the organisational infrastructure of a multi-agent system consists of organisational artefacts and agents that together are responsible for the management and enactment of the organisation. An organisational artefact employs a normative program which in turn implements a MOISE$^+$ specification. A programming language for the implementation of normative programs as well as a translation of MOISE$^+$ specifications into normative programs is described in [59]. JaCaMo provides similar functionality to $\mathcal{J}$-$\mathcal{M}$OISE$^+$ in allowing Jason agents to interact with organisational artefacts,

e.g., to take on a certain role. However, while these approaches allow a developer to program e.g., when an agent should adopt a role, the Jason agents have no explicit mechanisms to reason about norms and their deadlines and sanctions in order to adapt their behaviour at run time.

In [2], the the notion of *norm-aware* agents is introduced. Norm-aware agents deliberate on their goals, norms and sanctions before deciding which plan to select and execute. A norm-aware agent will violate a norm (accepting the resulting sanction) if it is in the agent's overall interests to do so, e.g., if meeting an obligation would result in an important goal of the agent becoming unachievable. Norm-aware agents are related to the notion of *deliberate normative agents* in [22], and are capable of *behaving according to a role specification in a normative organization* and *reasoning about violations* in the sense of [87]. Programming norm-aware agents in conventional BDI-based agent programming languages is difficult, as they lack support for deliberating about goals, norms, sanctions and deadlines. To address this, Alechina et al. [2] introduce the norm-aware agent programming language N-2APL. N-2APL is based on 2APL [27] and provides support for beliefs, goals, plans, norms, sanctions and deadlines. Alechina et al. show that N-2APL agents are rational in the sense of committing to a set of plans that will achieve the agent's most important goals and obligations by their deadlines while respecting its most important prohibitions. N-2APL agents are designed to interoperate with the 2OPL architecture for normative systems [31]. However, as the idea of organisational artefacts based on normative programs is closely related to the 2OPL architecture, Alechina et al. speculate that N-2APL agents could also be used in the JaCaMo framework.

Another approach that integrates norms in a BDI-based agent programming architecture is proposed in [70]. This extends the AgentSpeak(L) architecture with a mechanism that allows agents to behave in accordance with a set of non-conflicting norms. As in N-2APL, the agents can adopt obligations and prohibitions with deadlines, after which plans are selected to fulfil the obligations or existing plans are suppressed to avoid violating prohibitions. However, unlike N-2APL, [70] does not consider scheduling of plans with respect to their deadlines or possible sanctions.

Defeasible deontic logic is a representation and reasoning formalism for normative agent knowledge [65, 4, 48, 71]. It combines the non-monotonicity of defeasible logic with the normative modalities of deontic logic; with the former offering a computational environment of the latter. Defeasible and deontic logic provide the means for modeling multi-agent systems, where each agent is characterized by its own cognitive profile and normative system, as well as policies, which define privacy requirements for a user, access permissions for a resource, individual rights etc.

There are a few systems that implement defeasible deontic logics. One of them is DR-DEVICE-M [65], an extension of the DR-DEVICE Semantic Web-aware defeasible reasoning engine [15], with a mechanism for expressing modal and deontic logic operators. DR-DEVICE-M is based on work presented in [48], the main difference being that it adopts a theory transformation for turning a modal defeasible theory into a non-modal one, while [48] is based on a meta-program formalization of defeasible logic [68]. Furthermore, DR-DEVICE-M supports an extensible agent type definition scheme, where the agent is declared in the rule base, while modality interactions are dealt with parametrically, via external agent-type definition files. Compared to the programming frameworks discussed above, DR-DEVICE-M, as well as other similar defeasible deontic logic reasoning engines [3, 67, 71], cannot fully implement a programming environment for norm aware agents, but can play the role of the reasoning mechanism about norms.

3 Challenges

This section provides a list of issues that are currently challenging the development of normative multi-agent systems. These challenges concern computational systems for norms that can be used to specify, verify, and implement norms in multi-agent systems.

3.1 Verification of Norms

In its most general definition norm verification systems take a set of norms and some properties that they are suppose to comply with, and verify whether they do or do not. Ideally, they should provide counter examples if they do not comply. The term *offline* is sometimes used to differentiate this procedure from a similar one that some systems need to do at run-time (see Section 3.7). Many times the property of interest is absence of conflicts, and the set of norms represents a contract, a law, or some other legal artefacts.

Verification of norms face the community with the need to advance in at least two foundational issues:

- Finding a common family of formalisms where different interpretations of legal concepts, and the consequences of such interpretations, could be represented and discussed over a shared background. Even when formal semantics are provided, it is not always clear how to establish a fair comparison between existing proposals, or to understand what these formalisms have in common.
- Starting to consider not only expressiveness but also complexity and performance, so we can also build actual tools that help us produce better real-world norms. Significant research has been done in the direction of capturing the intended meaning of deontic modalities and fulfilling the expressiveness requirements for modest-sized normative systems. But the question of how computationally tractable the proposed systems are is usually overlooked.

These concepts should be discussed both in general and in specific contexts. This is particularly important because even if the general problem is hard, there could be "niche markets" where practical contributions to state-of-the-art and society could be made, like two-party contracts.

3.1.1 Need for Common Playground

Clarifying the meaning of legal concepts is a worthwhile endeavor. It's important to know if, for example, empowering someone to marry two individuals that have immunity to marriage is equivalent to having the liberty of bringing about that two individuals are married when they have the freedom of being married.

Although the community has not yet reach final consensus over the meaning of prohibition, obligation and permission, there is at least some level of agreement over their basic properties. Introduced in 1913 by Hofheld [54], other modalities such as *claim right*, *power*, *freedom* and *immunity* are still less clear. Many attempts have been made at formalizing the Hofhelian concepts (e.g., [64, 69, 81, 80]). However, semantic-wise, most of them went no further than structured language, leaving many questions unresolved. For instance, [80] introduces the concept of directed modalities to express sentences like "It is obligatory that Tom pays Mary $1000 in order to advance Mary's interests". If Mary is using the money to pay a blackmailer or to buy cancer-causing cigarettes, is she advancing her interests? According to whom? Can Tom deny the payment claiming that she would not use the money "to advance her interests"? [81] is more precise about which operator combinations are consistent given a few assumptions about the underlying logic, but because it only considers some basic modalities,

and because the logic is not fixed, we still don't know if, for example, being empowered but forbidden makes any sense.

In order to fully exploit the benefits of a formal language, formal semantics are also needed. A legal concept is fully understood if, given a legal corpora that uses it in combination with others, we can set apart legal from illegal behaviors. Such behaviors are usually shaped as narratives of events, i.e., traces. Given a state of affairs and a legal corpora, only some combinations of actions and events are legal: (future) traces again. So if we want to focus our efforts in practical approaches for improving the quality of real-life normative systems, what's needed are trace-friendly semantics.

There are currently many deontic proposals tailored for different situations (e.g., monitoring, conflict analysis, judging, etc). As mentioned in Section 2.1, the dream of a one-size-fits-all, experience suggest, is only Utopian. However, we could aim for founding logics on a family of related mathematical structures, forming a common base where comparisons become easier. Much as when giving Kripke semantics to modal logics allowed to easily compare axiomatic systems whose logic relation at that point was unknown [43].

Comparing again with Software Engineering, there are Architecture Description Languages and methodologies tailored for specific purposes such as avionics, web systems, finances, etc. In the same vein, maybe we need different deontic languages for trade contracts (e.g., supporting some numeric capabilities) than for finding conflicts in criminal law, where different ways of expressing intention become important. This is one of the challenges that need to be addressed: for each domain of interest (e.g., business contracts, civil law, SOAP, etc.) determine the expressivity requirements, which types of conflicts need to be detected and find tools that handle them properly.

3.1.2 Dealing with Complexity

Computational complexity is a topic on its own right, and we discuss it in the next section, but there is also the complexity that humans face when trying to understand large and complex pieces of information. Deontic formalisms willing to deal with real-world sized corpora should allow to model different levels of abstraction and to apply compositional reasoning [16]. These are powerful tools to cope with large, interrelated, complex and sometimes difficult to attain sets of concepts.

At the micro level, doing detailed intra-norm conflict analysis is definitively a must, but also to be able to understand complex inter-norm interactions. To exemplify, we would like to know if sending an email *counts as* placing an order in much the same way as sending a signed order form does, or if a given contract presents some contradiction surrounding those two practices. Zooming out, we would also like to know if placing an order surpassing one's paying capacity is considered illegal by some national law, abstracting away the particulars of how the order was placed. Compositional reasoning seems like the right tool, allowing to concentrate into details, checking their local correctness and then analyzing global correctness from clearly abstracted local properties.

However, care should be exercised when reasoning compositionally as this small but representative example shows: if some higher-ranked norm states that email orders are considered void for amounts over a limit, the particulars of how the order was placed cannot be abstracted away. The topic of compositional reasoning and abstraction in the deontic world, we believe, deserves further analysis.

3.1.3 The Quest for Tools

The predominant efforts carried out so far on normative systems, and deontic systems in general, share a strong philosophical bias. Identifying legal concepts and structuring them into operators, analyzing their interactions, expressiveness and derived paradoxes drove the main research agenda of the last decades. Even though there was valuable work in the direction of building real tools ([78, 46, 83, 51], etc.), this effort is far from being completed. Computer Science as a discipline, and in particular Computational Logic, has evolved significantly and today is much closer to provide usable tools that can cope with scales that match real-life cases. The area of modal logic, and specifically model checking, has proven to be very successful at this respect (e.g, [53]). It is a very good opportunity to make the most of this momentum and apply it for building usable tools for legal drafting and norm verification.

There is a very well-known usability problem in tools that deal with formal systems. Reconciling a smooth and natural user experience and the right level of abstraction with a rigorous formal language is a very difficult issue to overcome. The most frequent result are tools that demand very specialized technical skills to the end user. There are many similarities between software specification and legislative sciences [45], but there is a difference here: even though the final user for both communities is probably a non-technical person, people that *build* software, the developers, are inherently closer to formal verification. On the contrary, people that *build* laws, the legislators, are probably not familiar with the concept. For the time being, we see real tools aiding the legislative process only in the presence of a strong side-by-side collaboration between specialized technicians and legislators. The first major difficulty that has to be solved, we claim, is not really offering a pleasant experience for the non-technical user, but designing tools that can beat real-world sized problems.

We mentioned scalability, but the fundamental aspect we think has been largely overlooked for formal legal systems is computational tractability. Although some work has been done where practical considerations are analyzed (e.g., [8, 78]), many proposed systems analyze only the philosophical aspects, and no practical short-term role for them in the legislative process seems to be devised. And going farther from theoretical complexity, the ability to count on mature, highly tuned, inference tools is key. It is well-known that a full-fledged inference tool can many times provide better empirical results working with a reasonably complex logic than a young, made-from-scratch tool developed for a theoretically less complex logic. The work done in SMT-solvers, theoretically intractable, but successful in many real-life cases (e.g., [33]) is a good example. This point nicely connects to the considerations we made in the previous section: the ability to compare logics, and therefore know how much they overlap, will allow us to reuse off-the-shelf inference tools that have decades of development effort on their backs.

3.2 Operational Semantics for Norms

Operational semantics are used to formally specify the meaning of programming languages. They allow the specification of program executions and their formal verifications. It is therefore essential that programming languages for normative multi-agent systems (both for normative organizations and norm-aware agents) come with their operational semantics. Operational semantics require an operationalization of the normative concepts that are involved in the programming languages. An operationalization of normative concepts assumes a computational representation and a reasoning engine.

The expressivity of norms in most agent-based programming languages (e.g., 2OPL and N-2APL) is limited to atomic cases. In some of these programming languages, norms are

represented by atomic expressions such as $O(i, \phi)$ (agent i has an obligation to bring about ϕ, where ϕ is a conjunction of literals) such that reasoning with these concepts is limited to reasoning about ϕ. In some programming languages the deontic concepts are evaluated on states and operationalized by ensuring that each state either satisfies the deontic expressions (e.g., for $O(i, \phi)$ ensuring that each state satisfies ϕ) or otherwise be considered as a state in which agent i is in violation. In these programming languages, one can specify sanctions that should be imposed when specific violation occurs. A sanction ψ (often a conjunction of literals) is imposed by updating the state with ψ. In some other agent-based programming languages, the deontic expressions are extended with deadlines, e.g., the extended expression $\langle O(i, \phi), d, \psi \rangle$ indicates that agent i is obliged to bring about ϕ before deadline d otherwise sanction ψ should be imposed. The deadline in such an expression determines the temporal scope of the deontic element. Such extended deontic expressions are evaluated on paths (a sequence of states) and operationalized by ensuring that each execution path ending with a state satisfying deadline d has a state that satisfies the deontic element, or otherwise the state satisfying d should be updated with sanction ψ.

An important challenge in this respect is to have more expressive but computationally tractable fragments of normative concepts that can be operationalized and incorporated in the compilers or interpreters of agent-based programming languages. Also, the set of deontic concepts can be extended to include a variety of social and normative concepts such as responsibility, delegation of responsibility, trust, commitments, roles and their enactments. The incorporation of these concepts requires their operationalization which in turn requires computational representation and reasoning engines. A particular difficulty in this regard is the possibility of conflicts that can emerge between various normative concepts (e.g., agent i is both obliged and prohibited to bring about ϕ or agent i is responsible for ϕ but has no permission to control or change it) such that any operationalization of these concepts should be able to detect and resolve such conflicts. Another challenge is the operationalization of deontic concepts applied to groups of individuals, e.g., group obligation and collective responsibility. In particular, when and how such concepts can be created, maintained, released, and how to cope with violations and sanctions towards individual agents. Finally, normative concepts can be applied to both states and actions, including communication actions (e.g., agent i is obliged to bring about ϕ or to perform action α). State- and action-based norms can interact and it is a challenge to propose programming constructs to implement both kinds of normative concepts while governing their interactions.

3.3 Programming Norm Change

Norm change is an interesting but complex phenomenon. There are many aspects related to norm change: the entity/authority that can issue a norm change (e.g., can every agent issue a norm change?), the types of norms that can be changed (e.g., can a concrete norm applied to a specific agent change or can a general norm applied to a set of agents change too? do we allow constitutive and regulative norms to change?), the constraints that norm change mechanism should satisfy (e.g., under which conditions a norm can be adopted and what are the consequences?), and how to handle norm conflicts as a consequence of norm change. Other dynamic aspects of norms are known as legal annulment and legal abrogation. Legal abrogation refers to cancellation of a norm except for effects that were obtained already before the cancellation, and legal annulment refers to the total cancellation of a norm including all (earlier) effects of that norm.

Most aspects have been extensively investigated in the literature. For example, Artikis [7] has presented an infrastructure allowing agents to modify a protocol (a set of laws) at

runtime, Bou et al. [19] and Campos et al. [21] have investigated how norms in electronic institutions can change at runtime, and Oren et al. [73] have considered how powers can be used to create, delete and modify norms. Despite these research works on norm change not much attention has been given to programming languages that support the development and implementation of normative multi-agent systems in which norms can change at runtime. Such programming languages require constructs to create, charge, and discharge norms. It should be noted that it is often difficult, if not impossible, to operationalize phenomena such as legal annulment.

A proposal to facilitate norm change by means of dedicated programming constructed is presented in [32]. In this work, norm change is considered at the level of both norm instances and norm schemes. Norm instances are detached norms, i.e., norms that are already issued and directed to specific agents (e.g., agent i is obliged to fulfill its payment before next Wednesday) whereas norm schemes are general norms that can be applied and instantiated to specific agents (e.g., payments should be fulfilled within one week after signing the contract). Both norm instances and norm schemes are specified by means of rules which can be applied at run-time to modify norms. In this framework, changing a norm scheme may have consequences for the existing norm instances. For example, a norm scheme that states that a payment should be fulfilled before one week after signing the contract can be changed to a norm where the deadline is extended with one more week. The question is what needs to be done with the norm instances that are already issued before a norm change takes place. A related and more general problem in this respect is the problem of organisation change. Multi-agent organisations aim at controlling and coordinating the behaviour of individual agents by means of a plethora of concepts including normative concepts. Besides normative concepts, multi-agent organisations uses concepts such as roles, scenes, capabilities, services and databases that can be changed at runtime. A full-fledge programming language that supports the implementation of multi-agent organisation should therefore provide constructs and mechanisms that can be operationalized to manage the organisational changes.

3.4 Norm Adaptation and Emergence

Norm emergence and adaptation in open normative systems is related to norm change discussed issues discussed previously. Like society, norms continuously change. Some norms get discarded while new ones get introduced. Designers of these systems cannot foresee all possible changes that might occur to the system. The evolution of the system should not be prescribed nor regimented. In some systems, for instance, participating agents should be able to collectively decide which norms can abandoned or adapted. This could be the case for norms that are constantly violated by a majority of the agent population. While agents can bring up the adoption of a norm to a vote based on the behaviour of the participants.

3.5 Scalable Architectures for Normative Systems

A key challenge for large, distributed normative systems is scalability. In a normative multi-agent system, the interaction between agents and the environment is governed by a normative exogenous organisation, which is specified by a set of conditional norms with their associated deadlines and sanctions. The normative organisation monitors environmental changes resulting from the activities of agents to: determine any obligations to be fulfilled or prohibitions that should not be violated by the agents; determine any violations based on the deadlines of the detached obligations and prohibitions; and impose any corresponding sanctions. The role of the exogenous organisation is thus to continuously 1) monitor the

behaviour of agents, 2) evaluate the effects they bring about in the environment, 3) determine norms that should be respected, 4) check if any norm is violated and 5) take necessary measures (i.e., imposing sanctions) when norms are violated. This continuous process is often implemented by a so-called control cycle [31]. For large, open, distributed normative MAS, implementing the control cycle is non-trivial. State information may be distributed, and updated asynchronously by many agents, which may join or leave the system at any time. Such systems demand scalable architectures for behaviour and norm monitoring. One approach is to distribute such functions, however this entails additional complexity, e.g., transactional approaches to state update. For very large scale systems, 100% monitoring may be infeasible, and probabilistic guarantees of the detection of norm violations may be more appropriate. Such approaches, in turn, may impact the design of sanctions, and, when applied to behaviour monitoring, the design of the conditional norms which specify the normative exogenous organisation itself.

3.6 Algorithms for Compliance Checking

By this, we mean algorithms for checking whether external behaviour of agents conforms to the system's norms.

Compliance checking will require more or less computationally demanding algorithms depending on the structure of norms (e.g. do we need to monitor for bad states/bad events or do we need to analyse histories in order to detect a violation?). For some systems, existing algorithms can be re-used. For example algorithms for checking database integrity, or methods for checking business process compliance, or on-the-fly LTL model-checking [55]. For more complex norms we may need to design new algorithms or come up with non-trivial modifications of existing ones.

3.7 Normative Conflict Detection and Resolution

We addressed conflict detection at Section 3.1. The types of systems mentioned there usually need only to point out conflicts and leave resolution in the hands of a human being. There are some systems, however, that need to both detect and resolve conflicts at run-time, without human intervention, autonomous normative agents being an example.

It is seldom the case that offline mechanisms, which are usually more resource consuming, are fit for the purpose of run-time detection of conflicts. Lightweight detection mechanisms are needed, and those probably require changes in normative languages (see Section 3.2) to relax some assumptions or provide more information for the automatic resolution of conflicts. The whole topic of how a norm-aware agent should resolve conflicts at runtime requires further investigation, and probably domain-specific solutions instead of a general one.

3.8 Programming Languages for Norm-Aware Agents

Current approaches to normative agent programming such as N-2APL [2] have significant limitations, and many basic questions remain unanswered. For example, if the priorities of norms are commensurable, norm-aware deliberation is intractable — are there contexts where commensurable priorities are essential or contexts where 'best effort' (i.e., potentially intractable) deliberation is acceptable? What sorts of deadlines should be associated with norms — a single point in time or an interval of time? Should deadlines be interpreted as 'soft' or 'hard', or are both required? How can an agent developer estimate the time required to execute a plan in order to find out if an obligation can be discharged by its deadline? Should plan execution times rather be expressed as probabilistic profiles?

Additional challenges arise from overlaps between programming languages for implementing norm-aware agents and the operational semantics assumed for norms (see Section 3.2). For example, current approaches to normative agent programming have focussed on state-based norms, and would need to be extended to support action-based norms and their interactions with state-based norms. Similarly, existing approaches lack support for norm-aware deliberation about norms that apply to groups of agents, or normative interactions between agents, such as delegation of responsibility.

4 Conclusions

Looking at other fields that reached maturity, it is common to see an iterative pattern, each cycle consisting of an expansion phase, where new concepts are discovered, new entities are named and new methods are proposed, followed by a period of synthesis, where some of those are pruned or at least classified and people start to agree on which problems are best suited to be worked with which tools.

As was previously mentioned, the field of Computational Models for Normative Multi-Agent Systems involves a plethora of social concepts like roles, groups, social structures, organisations, institutions, norms, etc. that have been introduced in multi-agent system methodologies, models, specification and modelling languages, and computational frameworks.

If our Computational Models are to advance, it might be time to start classifying the various models and tools. Which tools are best for designing/implementing on-line monitoring? And for off-line checking? For programming action-based semantics? And for society-imposed regulations? How different tools and methodologies compare? Any of them solves the open issues?

This chapter leans toward providing some information to answer such type of questions, hopping to serve the practitioner by presenting a clear view of what we have and what we still need.

References

1 T. Ågotnes, W. van der Hoek, and M. Wooldridge. Robust normative systems. In Padgham, Parkes, Muller, and Parsons, editors, *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 747–754, Estoril, Portugal, May 2008. IFAMAAS/ACM DL.

2 Natasha Alechina, Mehdi Dastani, and Brian Logan. Programming norm-aware agents. In Vincent Conitzer, Michael Winikoff, Lin Padgham, and Wiebe van der Hoek, editors, *Proc. of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, volume 2, pages 1057–1064, Valencia, Spain, June 2012. IFAAMAS.

3 Grigoris Antoniou, Nikos Dimaresis, and Guido Governatori. A modal and deontic defeasible reasoning system for modelling policies and multi-agent systems. *Expert Systems with Applications*, 36(2, Part 2):4125 – 4134, 2009.

4 Grigoris Antoniou, Thomas Skylogiannis, Antonis Bikakis, Martin Doerr, and Nick Bassiliades. Dr-brokering: A semantic brokering system. *Knowledge-Based Systems*, 20:61–72, 2007.

5 F. Arbab, L. Astefanoaei, F. de Boer, M. Dastani, J.-J.Ch. Meyer, and N. Tinnermeier. Reo connectors as coordination artifacts in 2APL systems. In *Proceedings of the 11th Pacific Rim International Conference on Multi-Agents (PRIMA 2008)*, volume LNCS 5357, pages 42–53. Springer, 2009.

6 Josep Lluís Arcos, Marc Esteva, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Engineering open environments with electronic institutions. *Eng. Appl. of AI*, 18(2):191–204, 2005.

7 Alexander Artikis. Dynamic protocols for open agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 97–104, 2009.

8 Alexander Artikis, Marek Sergot, and Jeremy Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Logic*, 10:1:1–1:42, January 2009.

9 L. Astefanoaei, M. Dastani, J.-J. Ch. Meyer, and F. Boer. On the semantics and verification of normative multi-agent systems. *International Journal of Universal Computer Science*, 15(13):2629–2652, 2009.

10 M. Baldoni, G. Boella, M. Dorni, R. Grenna, and A. Mugnaini. powerJADE: Organizations and roles as primitives in the JADE framework. In *In of WOA 2008: Dagli oggetti agli agenti, Evoluzione dell'agent development: metodologie, tool, piattaforme e linguaggi*, pages 84–92, 2008.

11 M. Baldoni, G. Boella, and L. Van Der Torre. Roles as a coordination construct: Introducing powerJava. In *In Proceedings of 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, volume 150 (1), pages 9–29. Electronic Notes in Theoretical Computer Science, 2005.

12 Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer. The oo jdrew reference implementation of ruleml. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3791 of *Lecture Notes in Computer Science*, pages 218–223. Springer Berlin Heidelberg, 2005.

13 Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

14 J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer Berlin / Heidelberg, 2006.

15 Nick Bassiliades, Grigoris Antoniou, and Ioannis P. Vlahavas. A defeasible logic reasoner for the semantic web. *Int. J. Semantic Web Inf. Syst.*, pages 1–41, 2006.

16 S. Berezin, S. Campos, and E. Clarke. Compositional reasoning in model checking. *Compositionality: The Significant Difference*, pages 81–102, 1998.

17 G. Boella and L. van der Torre. Substantive and procedural norms in normative multiagent systems. *Journal of Applied Logic*, 6:152–171, 2008.

18 Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

19 Eva Bou, Maite López-Sánchez, and Juan A. Rodríguez-Aguilar. Adaptation of autonomic electronic institutions through norms and institutional agents. In *Proc. of the 7th International Conf. on Engineering Societies in the Agents World (ESAW)*, pages 300–319, 2006.

20 Joost Breuker, Emil Petkov, and Radboud Winkels. Drafting and validating regulations: The inevitable use of intelligent tools. In *AIMSA '00: Proceedings of the 9th International Conference on Artificial Intelligence*, pages 21–33, London, UK, 2000. Springer-Verlag.

21 Jordi Campos, Maite López-Sánchez, Juan Antonia Rodríguez-Aguilar, and Marc Esteva. Formalising situatedness and adaptation in electronic institutions. In *Proceedings of Coordination, Organizations, Institutions and Norms in Agent Systems (COIN@AAAI)*, pages 126–139, Berlin, Germany, 2009. Springer.

22 C. Castelfranchi, F. Dignum, Catholijn M. Jonker, and Jan Treur. Deliberative normative agents: Principles and architecture. In N. R. Jennings and Y. Lesperance, editors, *Intel-*

ligent Agents VI. Proceedings of the 6th International Workshop on Agent Theories and Architectures and Languages and ATAL'99., LNAI, pages 364–378. Springer Verlag, 2000.

23 A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4):410–425, 2000.

24 Owen Cliffe, Marina De Vos, and Julian Padget. Answer set programming for representing and reasoning about virtual institutions. In Katsumi Inoue, Satoh Ken, and Francesca Toni, editors, *Computational Logic for Multi-Agents (CLIMA VII)*, volume 4371 of *LNAI*, pages 60–79, Hakodate, Japan, May 2006. Springer.

25 Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and reasoning about multiple institutions. In Pablo Noriega, Javier Vázquez-Salceda, Guido Boella, Olivier Boissier, Virginia Dignum, Nicoletta Fornara, and Eric Matson, editors, *Coordination, Organization, Institutions and Norms in Agent Systems II – AAMAS 2006 and ECAI 2006 Int'l Workshops, COIN 2006 Hakodate, Japan, May 9, 2006 Riva del Garda, Italy, August 28, 2006*, volume 4386 of *LNCS*, pages 67–85. Springer Berlin / Heidelberg, 2007.

26 Domenico Corapi, Marina De Vos, Julian Padget, Alessandra Russo, and Ken Satoh. Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11:783–799, 2011.

27 M. Dastani. 2APL: a practical agent programming language. *International Journal of Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

28 M. Dastani, F. Arbab, and F.S. de Boer. Coordination and composition in multi-agent systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 439–446. 2005.

29 M. Dastani and J. Gomez-Sanz. Programming multi-agent systems. *The Knowledge Engineering Review*, 20(2):151–164, 2006.

30 M. Dastani and J.J. Gomez-Sanz. Agentlink iii technical forum group, programming multiagent systems. `http://people.cs.uu.nl/mehdi/al3promas.html`.

31 M. Dastani, D. Grossi, J.-J. Ch. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *Proc. Workshop on Knowledge Representation for Agents and Multi-Agent Systems*, LNCS 5605, pages 16–31, 2009.

32 Mehdi Dastani, John-Jules Meyer, and Nick Tinnemeier. Programming norm change. *Journal of Applied Non-classical Logics*, Published online, 2012.

33 Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54:69–77, September 2011.

34 V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Utrecht University, SIKS, 2004.

35 M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, and J.L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, pages 236–243, New York, US, July 2004.

36 Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, pages 1045–1052, Bologna, Italy, 2002.

37 Stephen Fenech, Gordon J. Pace, and Gerardo Schneider. Automatic conflict detection on contracts. In *ICTAC '09: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing*, pages 200–214, Berlin, Heidelberg, 2009. Springer-Verlag.

38 Stephen Fenech, Gordon J. Pace, and Gerardo Schneider. Clan: A tool for contract analysis and conflict discovery. In Zhiming Liu and Anders P. Ravn, editors, *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 90–96. Springer, 2009.

39 N. Fornara and M. Colombetti. Specifying artificial institutions in the event calculus. In V. Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, Information science reference, chapter XIV, pages 335–366. IGI Global, 2009.

40 Nicoletta Fornara. Specifying and monitoring obligations in open multiagent systems using semantic web technology. In A. Elçi, M. Tadiou Kone, and M. A. Orgun, editors, *Semantic Agent Systems: Foundations and Applications*, volume 344 of *Studies in Computational Intelligence*, chapter 2, pages 25–46. Springer-Verlag, 2011.

41 Nicoletta Fornara and Marco Colombetti. Representation and monitoring of commitments and norms using owl. *AI Commun.*, 23(4):341–356, 2010.

42 A. Garcia-Camino, P. Noriega, and J. A. Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Proc. of the 4th Int'l Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2005)*, pages 667–673, New York, NY, USA, 2005.

43 R. Goldblatt. Mathematical modal logic: a view of its evolution. *Journal of Applied Logic*, 1(5-6):309–392, 2003.

44 Daniel Gorín, Sergio Mera, and Fernando Schapachnik. Model Checking Legal Documents. In *Proceedings of the 2010 conference on Legal Knowledge and Information Systems: JURIX 2010*, pages 111–115, December 2010.

45 Daniel Gorín, Sergio Mera, and Fernando Schapachnik. A Software Tool for Legal Drafting. In *FLACOS 2011: Fifth Workshop on Formal Languages and Analysis of Contract-Oriented Software*, pages 1–15. Elsevier, 2011.

46 G. Governatori and D.H. Pham. Dr-contract: An architecture for e-contracts in defeasible logic. *International Journal of Business Process Integration and Management*, 4(3):187–199, 2009.

47 Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*, AI'07, pages 486–496, Berlin, Heidelberg, 2007. Springer-Verlag.

48 Guido Governatori and Antonino Rotolo. Bio logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17:36–69, 2008. 10.1007/s10458-008-9030-4.

49 D. Grossi. *Designing Invisible Handcuffs*. PhD thesis, Utrecht University, SIKS, 2007.

50 N. Den Haan. Tracs: A support tool for drafting and testing law. In *Jurix 92: Information Technology and Law*, pages 63–70, 1992.

51 S. Hagiwara and S. Tojo. Discordance Detection in Regional Ordinance: Ontology-based Validation. In *Proc. of the 2006 Conference on Legal Knowledge and Information Systems: JURIX 2006: The Nineteenth Annual Conference*, pages 111–120. IOS Press, 2006.

52 Jörg Hansen, Gabriella Pigozzi, and Leendert W. N. van der Torre. Ten philosophical problems in deontic logic. In Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

53 K. Havelund, M. Lowry, and J. Penix. Formal analysis of a space-craft controller using SPIN. *IEEE Transactions on Software Engineering*, pages 749–765, 2001.

54 W.N. Hohfeld. Some fundamental legal conceptions as applied in judicial reasoning. *Yale Lj*, 23:16, 1913.

55 Gerard J. Holzmann. On-the-fly model checking. *ACM Comput. Surv.*, 28(4es):120, 1996.

56 Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

57 J.F. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents: Giving the organisational power back to the agents. *International Journal of Autonomous Agents and Multi-Agent Systems*, 20:369–400, 2010.

58 J.F. Hübner, J.S. Sichman, and O. Boissier. $\mathcal{S}-\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In *Proceedings of the international workshop on Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer, 2006.

59 Jomi Fred Hübner, Olivier Boissier, and Rafael H. Bordini. From organisation specification to normative programming in multi-agent organisations. In Jürgen Dix, João Leite, Guido Governatori, and Wojtek Jamroga, editors, *Computational Logic in Multi-Agent Systems, 11th International Workshop, CLIMA XI, Lisbon, Portugal, August 16-17, 2010. Proceedings*, volume 6245 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2010.

60 Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multiagent systems using the $\mathcal{M}$OISE$^+$ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

61 John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.

62 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.

63 Andrew J.I. Jones and Marek Sergot. A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4):121, 1996.

64 S. Kanger and H. Kanger. Rights and parliamentarism. *Theoria*, 32(2):85–115, 1966.

65 Efstratios Kontopoulos, Nick Bassiliades, Guido Governatori, and Grigoris Antoniou. A modal defeasible reasoner of deontic logic for the semantic web. *Int. J. Semantic Web Inf. Syst.*, 7(1):18–43, 2011.

66 Alexander Kozlenkov, Rafael Penaloza, Vivek Nigam, Loic Royer, Gihan Dawelbait, and Michael Schroeder. Prova: Rule-based java scripting for distributed web applications: A case study in bioinformatics. In *EDBT Workshops'06*, pages 899–908, 2006.

67 Ho-Pun Lam and Guido Governatori. The making of spindle. In *Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, RuleML '09, pages 315–322, Berlin, Heidelberg, 2009. Springer-Verlag.

68 M. J. Maher and G. Governatori. A semantic decomposition of defeasible logics. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 299–305, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

69 D. Makinson. On the formal representation of rights relations. *Journal of philosophical Logic*, 15(4):403–425, 1986.

70 Felipe Rech Meneguzzi and Michael Luck. Norm-based behaviour modification in BDI agents. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 177–184. IFAAMAS, 2009.

71 Donald Nute, editor. *Defeasible Deontic Logic: Essays in Nonmonotonic Normative Reasoning*, volume 263 of *Synthese Library*. Kluwer Academic Publishers, Dordrecht, Holland, 1997.

72 Daniel Okouya and Virginia Dignum. Operetta: a prototype tool for the design, analysis and development of multi-agent organizations. In *AAMAS (Demos)*, pages 1677–1678. IFAAMAS, 2008.

73 N. Oren, M. Luck, and S. Miles. A model of normative power. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 815–822, 2010.

74 Pablo Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

75 Gordon J. Pace and Fernando Schapachnik. Types of rights in two-party systems: A formal analysis. In Burkhard Schäfer, editor, *JURIX*, volume 250 of *Frontiers in Artificial Intelligence and Applications*, pages 105–114. IOS Press, 2012.

76 Adrian Paschke and Harold Boley. Rule responder: Rule-based agents for the semantic-pragmatic web. *International Journal on Artificial Intelligence Tools*, 20(06):1043–1081, 2011.

77 H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57:91–115, 1996.

78 Cristian Prisacariu and Gerardo Schneider. $\mathcal{CL}$: An action-based logic for reasoning about contracts. In *WoLLIC '09: Proceedings of the 16th International Workshop on Logic, Language, Information and Computation*, pages 335–349, Berlin, Heidelberg, 2009. Springer-Verlag.

79 Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Give agents their artifacts: the A&A approach for engineering working environments in MAS. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*. IFAAMAS, 2007.

80 G. Sartor. Fundamental legal concepts: A formal and teleological characterisation*. *Artificial Intelligence and Law*, 14(1):101–142, 2006.

81 M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic (TOCL)*, 2(4):581–622, 2001.

82 V. Torres Silva. From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. *JAAMAS*, 17(1):113–155, 2008.

83 Ellis Solaiman, Carlos Molina-jimenez, and Santosh Shrivastava. Model checking correctness properties of electronic contracts. In *Proceedings of the International conference on Service Oriented Computing (ICSOC03)*, pages 303–318. Springer-Verlag, 2003.

84 N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Roles and norms for programming agent organizations. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proceedings of the Eight International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 121–128. IFAMAAS/ACM DL, 2009.

85 N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Programming norm change. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 957–964. IFAMAAS/ACM DL, 2010.

86 N. Tinnemeier, M. Dastani, J.-J. Ch. Meyer, and L. van der Torre. Programming normative artifacts with declarative obligations and prohibitions. In *Proceedings of IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 145–152. IEEE Computer Society, 2009.

87 M. Birna van Riemsdijk, Koen V. Hindriks, and Catholijn M. Jonker. Programming organization-aware agents: A research agenda. In *Proceedings of the Tenth International Workshop on Engineering Societies in the Agents' World (ESAW'09)*, volume 5881 of *LNAI*, pages 98–112. Springer, 2009.

88 F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.