Dynamic specification of open agent systems

ALEXANDER ARTIKIS, *Institute of Informatics and Telecommunications, National Centre for Scientific Research 'Demokritos', Athens 15310, Greece*
E-mail: a.artikis@iit.demokritos.gr; a.artikis@acm.org

Abstract

Multi-Agent Systems (MAS) where the agents are developed by different parties and serve different interests, are often classified as 'open'. The specification of open MAS is largely seen as a design-time activity. Moreover, there is no support for run-time specification modification. Due to environmental, social or other conditions, however, it is often required to revise the MAS specification during its execution. To address this requirement, we present an infrastructure for 'dynamic' open MAS specifications, that is, specifications that may be modified at run-time by the agents. We adopt a bird's eye view of an open MAS, as opposed to an agent's own perspective whereby it reasons about how it should act. The infrastructure consists of well-defined procedures for proposing a modification of the 'rules of the game', as well as decision-making over and enactment of proposed modifications. We evaluate proposals for rule modification by modelling a dynamic specification as a metric space, and by considering the effects of accepting a proposal on system utility. Furthermore, we constrain the enactment of proposals that do not meet the evaluation criteria. We employ the action language $C+$ to formalize dynamic specifications, and the 'Causal Calculator' implementation of $C+$ to execute the specifications. We illustrate our infrastructure by presenting a dynamic specification of a resource-sharing protocol.

Keywords: norm, norm change, norm-governed system, executable specification, organized adaptation.

1 Introduction

An 'open' Multi-Agent System (MAS) is a system in which the member agents are developed by different parties and serve different, often competing interests. The members of such systems are assumed to be autonomous, reflecting the independence of their users, and heterogeneous, reflecting the independence of their designers [83]. A few examples of this type of MAS are electronic marketplaces [80], virtual enterprises [46], virtual organizations [35] and digital media rights management applications [8].

In open MAS the behaviour of an agent Ag cannot be predicted in advance by the other agents, or the designer of the MAS [47], as these do not have access to the code or the internal state of Ag. Furthermore, it cannot be assumed that agents will behave as they are intended to behave because an agent may fail to conform to the MAS specification due to, say, a bug in the code of the agent, or because, as mentioned above, agents often serve competing interests, and so may choose not to conform to the MAS specification in order to achieve their individual goals [75] (this is what Minsky and Ungureanu [66] referred to as *inadvertent* and *malicious* violations, respectively).

In these cases it is meaningful to speak of obligations, permissions and perhaps other more complex normative relations that may exist between the agents, and to describe agent behaviour as governed by norms, which may be violated, accidentally or on purpose [52]. Among these normative relations, considerable emphasis has been placed on the representation of *institutional power* [53]. This is a standard feature of any normative system whereby designated agents, when acting in specified roles, are empowered by an institution to create specific relations or states of affairs. Consider, for example,

Vol. 22 No. 6, © The Author, 2011. Published by Oxford University Press. All rights reserved.
For Permissions, please email: journals.permissions@oup.com
Published online July 30, 2011 doi:10.1093/logcom/exr018

the case in which an agent is empowered by an institution to award a contract and thereby create a bundle of normative relations between the contracting parties.

Several approaches have been proposed in the literature for the specification of open MAS. The majority of these approaches offer 'static' MAS specifications, that is, there is no support for run-time specification modification. In some open MAS, however, environmental, social or other conditions may favour specifications of MAS that are modifiable during the system execution. Consider, for instance, the case of a malfunction of a large number of sensors in a sensor network, or the case of manipulation of a voting procedure due to strategic voting, or when an organization conducts its business in an inefficient manner. Therefore, we present in this article an infrastructure for 'dynamic' MAS specifications, that is, specifications that are developed at design-time but may be modified at run-time by the members of a system. We adopt a bird's eye view of a MAS, as opposed to an agent's own perspective whereby it reasons about how it should act. The presented infrastructure for dynamic specifications is an extension of our work on static specifications [5, 7], and is motivated by 'dynamic argument systems' [12]. These are argument systems in which, at any point in the disputation, agents may start a meta level debate, that is, the rules of order become the current point of discussion, with the intention of altering these rules.

Our infrastructure for dynamic specifications allows agents to alter the specification of a protocol P during the protocol execution. P is considered an 'object' protocol; at any point in time during the execution of the object protocol the participants may start a 'meta' protocol in order to decide whether the object protocol specification should be modified. Moreover, the participants of the meta protocol may initiate a meta–meta protocol to decide whether to modify the specification of the meta protocol, or they may initiate a meta–meta–meta protocol to modify the specification of the meta–meta protocol, and so on.

Unlike existing approaches on dynamic specifications, we place emphasis on the procedure with which agents initiate a meta protocol. We distinguish between successful and unsuccessful attempts to initiate a meta protocol by identifying the conditions in which an agent has the institutional power to propose a specification change. We evaluate an agent's proposal for specification change by modelling a dynamic specification as a *metric space* [14], and by taking into consideration the effects of accepting a proposal on system utility. We constrain the enactment of proposals that do not meet the evaluation criteria. Furthermore, we formalize procedures for role-assignment in a meta level, that is, we specify which agents may participate in a meta protocol, and the roles they may occupy in the meta protocol.

We employ a resource-sharing protocol to illustrate our infrastructure for dynamic specifications: the object protocol concerns resource-sharing while the meta protocols are voting protocols. In other words, at any time during a resource-sharing procedure the agents may vote to change the rules that govern the management of resources. The resource-sharing protocol was chosen for the sake of providing a concrete example. In general, the object protocol may be any protocol for open MAS, such as a protocol for coordination or e-commerce. Similarly, a meta protocol can be any procedure for decision-making over specification modification—argumentation, negotiation and so on.

We encode dynamic MAS specifications in executable action languages. In this article we employ the action language $C+$ [38], a formalism with explicit transition system semantics. The $C+$ language, when used with its associated software implementation, the 'Causal Calculator' (CCALC), supports a wide range of computational tasks of the kind that we wish to perform on MAS specifications.

The remainder of this article is structured as follows. First, we present the action language $C+$. Secondly, we review a static specification of a resource-sharing protocol, and show how CCALC can be used to prove properties of the static specification. Thirdly, we present a dynamic specification of the resource-sharing protocol and an infrastructure for modifying the protocol specification during the protocol execution. We then show how CCALC can be used to prove properties of the infrastructure for dynamic specifications, as well as support run-time activities by computing the normative relations current at each time. Finally, we summarize our work, outline the differences between the present work and earlier versions of this article, discuss related research and outline directions for further work.

2 The $\mathcal{C}+$ language

$\mathcal{C}+$, as mentioned above, is an action language with an explicit transition system semantics. We describe here the version of $\mathcal{C}+$ presented in [38].

2.1 *Basic definitions*

A *multi-valued propositional signature* is a set σ of symbols called *constants*, and for each constant $c \in \sigma$, a non-empty finite set $dom(c)$ of symbols, disjoint from σ, called the *domain* of c. For simplicity, in this presentation we will assume that every domain contains at least two elements.

An *atom* of signature σ is an expression of the form $c{=}u$ where $c \in \sigma$ and $u \in dom(c)$. A Boolean constant is one whose domain is the set of truth values $\{\mathsf{t}, \mathsf{f}\}$. When c is a Boolean constant we often write c for $c{=}\mathsf{t}$ and $\neg c$ for $c{=}\mathsf{f}$. A *formula* φ of signature σ is any propositional combination of atoms of σ. An *interpretation* I of σ is a function that maps every constant in σ to an element of its domain. An interpretation I *satisfies* an atom $c{=}u$ if $I(c){=}u$. The satisfaction relation is extended from atoms to formulas according to the standard truth tables for the propositional connectives. A *model* of a set X of formulas of signature σ is an interpretation of σ that satisfies all formulas in X. If every model of a set X of formulas satisfies a formula φ then X *entails* φ, written $X \models \varphi$.

2.2 *Syntax*

The representation of an action domain in $\mathcal{C}+$ consists of *fluent* constants and *action* constants.

- Fluent constants are symbols characterizing a state. They are divided into two categories: simple fluent constants and statically determined fluent constants. Simple fluent constants are related to actions by *dynamic laws*, that is, laws describing a transition from a state s_i to its successor state s_{i+1}. Statically determined fluent constants are characterized by *static laws*, that is, laws describing an individual state, relating them to other fluent constants. Static laws can also be used to express constraints between simple fluent constants. Static and dynamic laws are defined below.
- Action constants are symbols characterizing state transitions. In a transition $(s_i, \varepsilon_i, s_{i+1})$ from a state s_i to its successor state s_{i+1}, the transition label ε_i, also called an 'event', represents the actions performed concurrently by one or more agents or occurring in the environment. Transitions may be non-deterministic. Action constants are used to name actions, attributes of actions or properties of transitions as a whole.

An *action signature* $(\sigma^{\mathrm{f}}, \sigma^{\mathrm{a}})$ is a non-empty set σ^{f} of fluent constants and a non-empty set σ^{a} of action constants. An *action description* D in $\mathcal{C}+$ is a non-empty set of *causal laws* that define a transition system of a particular type. A causal law can be either a *static law* or a *dynamic law*. A static law is an expression

$$\textsf{caused } F \textsf{ if } G \tag{1}$$

where F and G are formulas of fluent constants. In a static law, constants in F and G are evaluated on the same state. A dynamic law is an expression

$$\textsf{caused } F \textsf{ if } G \textsf{ after } H \tag{2}$$

where F, G and H are formulas such that every constant occurring in F is a simple fluent constant, every constant occurring in G is a fluent constant and H is any combination of fluent constants and action constants. In a transition from state s_i to state s_{i+1}, constants in F and in G are evaluated on s_{i+1}, fluent constants in H are evaluated on s_i and action constants in H are evaluated on the transition itself. F is called the *head* of the static law (1) and the dynamic law (2).

The full $C+$ language also provides *action dynamic laws*, which are expressions of the form

$$\textsf{caused } \alpha \textsf{ if } H$$

where α is a formula containing action constants only and H is a formula of action and fluent constants. We will not use action dynamic laws in this article and so omit the details in the interests of brevity.

The $C+$ language provides various abbreviations for common forms of causal law. For example, a dynamic law of the form

$$\textsf{caused } F \textsf{ if } \top \textsf{ after } H \wedge \alpha$$

where α is a formula of action constants is often abbreviated as

$$\alpha \textsf{ causes } F \textsf{ if } H$$

The above abbreviation may be used to express the conditional effects of a set of actions. In the case where H is $\top$ the above is usually written as

$$\alpha \textsf{ causes } F$$

In this way we may express the unconditional effects of a set of actions. When presenting the resource-sharing protocol specification, we will often employ the causes abbreviation to express the effects of the agents' actions. We will also employ the $C+$ abbreviation

$$\textsf{default } F$$

which is shorthand for the static law

$$\textsf{caused } F \textsf{ if } F$$

expressing that F holds in the absence of information to the contrary.

When it aids readability, we will express 'F if and only if G' as

$$\textsf{caused } F \textsf{ iff } G$$

This is a shorthand for the pair of static laws

$$\textsf{caused } F \textsf{ if } G$$

and

$$\textsf{default } \neg F$$

Finally, we will express the inertia of a fluent constant c over time as:

$$\textsf{inertial}\ c$$

This is an abbreviation for the *set* of dynamic laws of the form (for all values $u \in dom(c)$):

$$\textsf{caused}\ c{=}u\ \textsf{if}\ c{=}u\ \textsf{after}\ c{=}u$$

A $C+$ action description is a non-empty set of causal laws. Of particular interest is the sub-class of *definite* action descriptions. A $C+$ action description D is *definite* if:

- the head of every causal law of D is an atom or $\bot$; and
- no atom is the head of infinitely many causal laws of D.

The $C+$ action description in this article will be definite.

2.3 Semantics

It goes beyond the scope of the article to give a full account of the $C+$ language and its semantics. We trust that the $C+$ language, and especially its abbreviations, are sufficiently natural that readers can follow the presentation of the case study in later sections. Interested readers are referred to [38, 39] for further technical details. For completeness, we summarize here the semantics of *definite* action descriptions ignoring, as we are, the presence of action dynamic laws (and assuming that the domain of every constant contains at least two elements). We emphasize the transition system semantics, as in [74].

Every action description D of $C+$ defines a labelled transition system, as follows:

- States of the transition system are interpretations of the fluent constants σ^{f}. It is convenient to identify a state s with the set of fluent atoms satisfied by s (in other words, $s \models f{=}v$ if and only if $f{=}v \in s$ for every fluent constant f).

  Let $T_{\mathrm{static}}(s)$ denote the heads of all static laws in D whose conditions are satisfied by s:

  $$T_{\mathrm{static}}(s) =_{\mathrm{def}} \{F \mid \text{static law (1) is in } D, s \models G\}$$

  For a definite action description D, an interpretation s of σ^{f} is a *state of the transition system defined by* D, or simply, a *state of* D, when

  $$s = T_{\mathrm{static}}(s) \cup Simple(s)$$

  where $Simple(s)$ denotes the set of simple fluent atoms satisfied by s. (So $s - Simple(s)$ is the set of statically determined fluent atoms satisfied by s.)
- Transition labels of the transition system defined by D are the interpretations of the action constants σ^{a}.

  A *transition* is a triple (s, ε, s') in which s is the initial state, s' is the resulting state, and ε is the transition label. Since transition labels are interpretations of σ^{a}, it is meaningful to say that a transition label ε satisfies a formula α of σ^{a}: when $\varepsilon \models \alpha$ we sometimes say that the transition (s, ε, s') is of type α.

- Let $E(s,\varepsilon,s')$ denote the heads of all dynamic laws of D whose conditions are satisfied by the transition (s,ε,s'):

$$E(s,\varepsilon,s') =_{\text{def}} \{F \mid \text{dynamic law (2) is in } D, s' \models G, s \cup \varepsilon \models H\}$$

For a definite action description D, (s,ε,s') is a *transition of* D, or in full, a *transition of the transition system defined by* D, when s and s' are interpretations (sets of atoms) of σ^{f} and ε is an interpretation of σ^{a} such that:
  - $s = T_{\text{static}}(s) \cup \mathit{Simple}(s)$ (s is a state of D)
  - $s' = T_{\text{static}}(s') \cup E(s,\varepsilon,s')$

For any non-negative integer m, a *path* or *history of* D of length m is a sequence

$$s_0\,\varepsilon_0\,s_1 \ldots s_{m-1}\,\varepsilon_{m-1}\,s_m$$

where $(s_0,\varepsilon_0,s_1),\ldots,(s_{m-1},\varepsilon_{m-1},s_m)$ are transitions of D.

2.4 The Causal Calculator

The Causal Calculator (CCALC) (`http://userweb.cs.utexas.edu/users/tag/cc`) is a software implementation developed by the Action Group of the University of Texas for representing action and change in the $C+$ language, and performing a range of computational tasks on the resulting formalizations. The functionality of CCALC includes computation of 'prediction' (temporal projection) and planning queries. Action descriptions in $C+$ are translated by CCALC first into the language of *causal theories* [38] and then into propositional logic. The (ordinary, classical) models of the propositional theory correspond to paths in the transition system described by the original action description in $C+$. To compute an answer to a query, CCALC invokes a satisfiability (SAT) solver to find models of the propositional theory which also satisfy the query. A detailed account of CCALC's operation and functionality may be found in [1, 38, 56].

In the following sections we present a $C+$ action description, D^{RS}, expressing a specification of a resource-sharing protocol. More precisely, first we present a static specification of a resource-sharing protocol, and then we present an infrastructure for dynamic specification of a resource-sharing protocol. Moreover, we show how we use CCALC to execute a protocol specification.

3 A static resource-sharing protocol

We present a specification of a resource-sharing or *floor control* protocol in the style of [7]. In the field of Computer-Supported Co-operative Work the term 'floor control' denotes a service guaranteeing that at any given moment only a designated set of users (subjects) may simultaneously work on the same objects (shared resources), thus creating a temporary exclusivity for access on such resources. We present a 'chair-designated' Floor Control Protocol, that is, a distinguished participant is the arbiter over the usage of a specific resource. For simplicity we assume a single resource.

The protocol roles are summarized below:

- *Floor Control Server* (*FCS*), the role of the only participant physically manipulating the shared resource.
- *Subject* (*S*), the role of designated participants requesting the floor from the chair, releasing the floor, and requesting from the FCS to manipulate the resource.

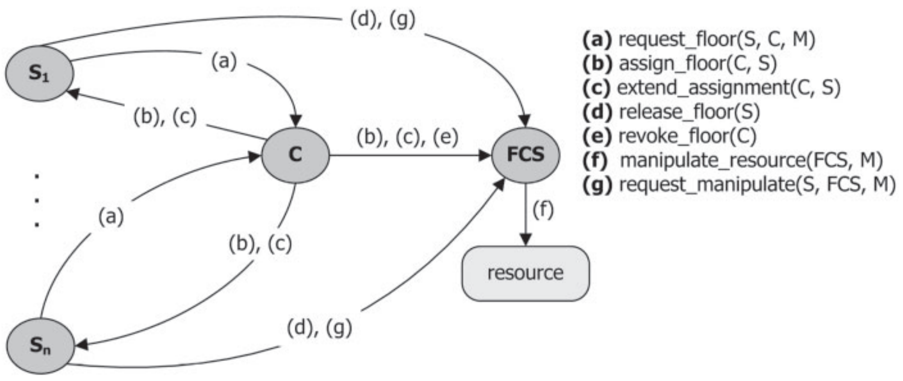
S₁
Sₙ
C
FCS
resource
(a)
(b), (c)
(d), (g)
(b), (c), (e)
(f)
(a) request_floor(S, C, M)
(b) assign_floor(C, S)
(c) extend_assignment(C, S)
(d) release_floor(S)
(e) revoke_floor(C)
(f) manipulate_resource(FCS, M)
(g) request_manipulate(S, FCS, M)

FIGURE 1. A chaired floor control protocol.

TABLE 1. A subset of the action signature of D^{RS} (Part A)

Variable	Domain
M	a set of resource manipulation types
Ag, S, S', C	a set of agent ids
Simple Fluent Constant	**Domain**
$role\_of(Ag)$	$\{subject, chair, fcs\}$
$holder(S)$, $sanctioned(Ag)$	Boolean
$requested(S)$	a set of resource manipulation types
$best\_candidate$	a set of agent ids
$c\_alloc(S)$	Z^+
Statically Determined Fluent Constant (Boolean)	
$powRequest(S,C)$, $powAssign(C,S)$, $powRequestMpt(S,FCS,M)$	
Action Constant σ^{act} (Boolean)	
$request\_floor(S,C,M)$, $assign\_floor(C,S)$, $request\_manipulate(S,FCS,M)$	

- *Chair* (C), the role of the participant assigning the floor for a particular time period to a subject, extending the time allocated for the floor, and revoking the floor from the subject holding it.

The floor can be either ‘granted’, denoting that a subject has exclusive access to the resource (by the chair), or ‘free’, denoting that no subject currently holds the floor. In both cases the floor may or may not be requested by a subject (for example, the floor may be granted to subject S' and requested by subject S'' at the same time).

Figure 1 provides an informal description of the possible interactions between the agents occupying the protocol roles. More details about these interactions will be given presently.

Table 1 shows a subset of the action signature of D^{RS}, that is, the $C+$ action description expressing the specification of the resource-sharing protocol. Variables start with an upper-case letter, and fluent and action constants start with a lower-case letter. The intended reading of the constants of the action signature will be explained below.

It has been argued [53] that the specifications of protocols for open MAS should explicitly represent the concept of institutional power (or, for short, ‘power’), that is, the characteristic feature of institutions whereby designated agents, often when acting in specific roles, are empowered, by the institution, to create or modify facts of special significance in that institution—*institutional facts*—usually by performing a specified kind of act. Searle [72], for example, has distinguished between *brute facts* and institutional facts. Being in physical possession of an object is an example of a brute fact (it can be observed); being the owner of that object is an institutional fact. The resource-sharing protocol specification explicitly represents the concept of institutional power. Moreover, we follow the standard, long-established distinction between institutional power, physical capability, permission and obligation (see [58] for illustrations of this distinction).

According to the resource-sharing protocol specification, all actions are physically possible at any time. In other examples the specification of physical capability could be different.

In this example, a subject S is empowered to request the floor from the chair C when S has no pending requests:

$$\begin{array}{l}\textsf{caused}\ powRequest(S,C)\ \textsf{iff}\\ \quad role\_of(S){=}subject,\\ \quad role\_of(C){=}chair,\\ \quad requested(S){=}null\end{array} \tag{3}$$

$C+$ abbreviations, including iff, were presented in Section 2. The *powRequest* fluent constant expresses the institutional power to request the floor, while the *role_of* fluent constant expresses the role an agent occupies. The *requested* fluent constant records an agent's requests for the floor—$requested(S){=}null$ denotes that S has no requests for the floor. *powRequest* is a statically determined fluent constant while *role_of* and *requested* are simple fluent constants (see Table 1).

Each simple fluent constant of D^{RS} is inertial, that is to say, its value persists by default from one state to the next. The constraint that a fluent constant f is inertial is expressed in $C+$ by means of the causal law abbreviation:

$$\textsf{inertial}\ f \tag{4}$$

Having specified the institutional power to request the floor, it is now possible to define the effects of this action: a request for the floor is eligible to be serviced if and only if it is issued by an agent with the institutional power to request the floor. Requests for the floor issued by agents without the necessary institutional power are ignored.

We chose to specify that a subject is always permitted to exercise its power to request the floor. Moreover, a subject S is permitted to request the floor even if S is not empowered to do so. In the latter case a request for the floor will be ignored by the chair (since S was not empowered to request the floor) but S will not be *sanctioned* since it was not forbidden to issue the request. In general, an agent is sanctioned when performing a forbidden action or not complying with an obligation. A few examples of sanctions will be shown presently (a more thorough treatment of sanctions may be found in [5, 7]). Finally, a subject is never obliged to request the floor.

The chair's power to assign the floor is defined as follows:

$$\begin{array}{l}\textsf{caused}\ powAssign(C,S)\ \textsf{iff}\\ \quad role\_of(C){=}chair,\\ \quad \forall S'\ \neg holder(S'),\\ \quad best\_candidate{=}S\end{array} \tag{5}$$

The chair C is empowered to assign the floor to S if the floor is free, and S is the best candidate for the floor. The simple fluent constant *holder* expresses whether an agent has been allocated the floor. The simple fluent constant *best_candidate* denotes the best candidate for the floor. The definition of this constant is application specific. For instance, the best candidate could be the one with the earliest request, that with the most 'urgent' request (however 'urgent' may be defined), and so on.

The result of exercising the power to assign the floor to S is that the floor becomes granted to S for a specified time period. Sending an *assign_floor* message to an agent S without the power to assign the floor to S has no effect on the access rights of S.

In this example, the conditions in which the chair is permitted to assign the floor are expressed as follows:

$$\begin{array}{l} \textsf{caused}\ \mathit{perAssign}(C,S)\ \textsf{iff} \\ \qquad \mathit{role\_of}(C){=}\mathit{chair}, \\ \qquad \forall S'\ \neg \mathit{holder}(S'), \\ \qquad \mathit{best\_candidate}{=}S, \\ \qquad \mathit{c\_alloc}(S) < 3 \end{array} \tag{6}$$

The chair is permitted to assign the floor to S if the floor is free, S is the best candidate for the floor, and S has not been allocated the floor the last 3 (or more) times. A simple fluent constant $\mathit{c\_alloc}(S)$ is incremented by 1 when S is assigned the floor, and set to 0 when the floor is assigned to some other subject S'.

Note that, according to rules (5) and (6), the chair is not always permitted to exercise its power to assign the floor.

As mentioned above, an agent is subject to penalty when performing a forbidden action. In this case, a chair is subject to penalty when assigning the floor while being forbidden to do so. We record sanctions as follows:

$$\begin{array}{l} \mathit{assign\_floor}(C,S)\ \textsf{causes}\ \mathit{sanctioned}(C)\ \textsf{if} \\ \qquad \mathit{role\_of}(C){=}\mathit{chair}, \\ \qquad \neg \mathit{perAssign}(C,S) \end{array} \tag{7}$$

sanctioned is a simple fluent constant. The actual penalty associated with the violation of a prohibition, or non-compliance with an obligation, may come in different flavours. We will show a type of penalty in a later section.

In this example, the conditions in which the chair is obliged to assign the floor are the same as the conditions in which the chair is permitted to assign the floor.

Similarly we specify the power, permission and obligation to perform the remaining protocol actions, and the effects of these actions. For instance, a subject's power to request a manipulation of the shared resource is defined as follows:

$$\begin{array}{l} \textsf{caused}\ \mathit{powRequestMpt}(S,\mathit{FCS},M)\ \textsf{iff} \\ \qquad \mathit{role\_of}(\mathit{FCS}){=}\mathit{fcs}, \\ \qquad \mathit{holder}(S) \end{array} \tag{8}$$

The *powRequestMpt* fluent constant expresses the institutional power to request a resource manipulation. A subject S is empowered to request a resource manipulation of type M from the floor control server FCS if S is the holder of the resource.

The specification of the power, permission or obligation to request a resource manipulation, assign the floor or perform some other protocol action, should include a deadline stating the time by which the action of requesting a resource manipulation, assigning the floor, etc, should be performed. Including deadlines in the formalization lengthens the presentation and is omitted here for simplicity. Example formalizations of deadlines may be found in [7].

TABLE 2. A resource-sharing protocol specification

Action	Power	Permission	Obligation
$request\_floor(S,C,M)$	$requested(S)=null$	$\top$	$\bot$
$assign\_floor(C,S)$	$\forall S' \neg holder(S')$, $best\_candidate=S$	$\forall S' \neg holder(S')$, $best\_candidate=S$, $c\_alloc(S)<3$	$\forall S' \neg holder(S')$, $best\_candidate=S$, $c\_alloc(S)<3$
$request\_manipulate(S,FCS,M)$	$holder(S)$	$holder(S)$	$\bot$

To summarize, Table 2 presents the conditions in which a protocol participant has the institutional power, permission and obligation to perform an action. To simplify the presentation, in Table 2 we show only three protocol actions: *request_floor*, *assign_floor* and *request_manipulate*. Moreover, we do not display the *role_of* fluent constant and assume that S denotes an agent occupying the role of subject, C denotes an agent occupying the role of chair, and FCS denotes an agent occupying the role of floor control server.

4 Proving properties of the static resource-sharing protocol

The explicit transition system semantics of the $C+$ language enables us to prove various properties of the presented specification, which is expressed by means of the action description D^{RS}. Consider the following example.

PROPERTY 4.1
There is no protocol state in which the floor is free and the chair is not empowered to assign it to the best candidate.

PROOF. Assume a state s of the transition system defined by D^{RS} in which the best candidate for the floor is subject S, the floor is free, and the chair C is not empowered to assign the floor to S. In other words, for any S, C,

$$s \models best\_candidate=S \wedge \forall S' \neg holder(S') \wedge role\_of(C)=chair \wedge \neg powAssign(C,S)$$

Since s is a state of D^{RS}, it is an interpretation of σ^{f} such that $s=T_{\mathrm{static}}(s) \cup Simple(s)$, where $T_{\mathrm{static}}(s) =_{\mathrm{def}} \{F \mid \text{static law `}\mathsf{caused}\ F\ \mathsf{if}\ G\text{' is in } D^{RS},\ s \models G\}$ and $Simple(s)$ denotes the set of simple fluent atoms satisfied by s (see Section 2.3). From rule (5), and the fact that

$$s \models best\_candidate=S \wedge \forall S' \neg holder(S') \wedge role\_of(C)=chair$$

we have that $powAssign(C,S) \in T_{\mathrm{static}}(s)$. According to our initial assumption, however, in s the chair C is not empowered to assign the floor to S, that is, $powAssign(C,S) \notin s$, which implies that $s \neq T_{\mathrm{static}}(s) \cup Simple(s)$. Therefore, s is not a state of D^{RS}. ■

CCALC provides an automated means for proving properties (such as Property 4.1) of a protocol specification formalized in $C+$. We express the $C+$ action description D^{RS} in CCALC's input language and then query CCALC about D^{RS} in order to prove properties of the protocol specification.[1] Consider the following example.

PROPERTY 4.2
A chair is always sanctioned when it performs a forbidden assignment of the floor.

We instruct CCALC to compute all states s' such that

- (s, ε, s') is a transition of D^{RS},
- $s \models \neg perAssign(C,S) \wedge role\_of(C){=}chair$, and
- $\varepsilon \models assign\_floor(C,S)$.

For every state s' computed by CCALC we obtain

$$s' \models sanctioned(C)$$

This is due to rule (7).

Note that for some states s' computed by CCALC we obtain $s' \models holder(S)$, meaning that, in these cases, the chair C was empowered, although forbidden, to assign the floor to subject S (see, respectively, rules (5) and (6) for the specification of the power and permission to assign the floor). CCALC also computed states s' such that $s' \models \neg holder(S)$, that is, in these cases, the chair was not empowered to assign the floor to S.

We may prove further properties of the resource-sharing protocol specification, in the manner shown above, such as that an agent is permitted to perform at least one action in every protocol state, an agent is never forbidden and obliged to perform an action, non-compliance with an obligation always leads to a sanction, and so on. Further examples of proving properties of protocol specifications formalized in $C+$ will be presented in Section 6.

5 An infrastructure for a dynamic resource-sharing protocol

Being motivated by Brewka [12], we present an infrastructure that allows agents to modify (a subset of) the rules of a protocol at run-time. Regarding our running example, we consider the resource-sharing protocol as an 'object' protocol; at any point in time during the execution of the object protocol the participants may start a 'meta' protocol in order to potentially modify the object protocol rules—for instance, replace an existing rule-set with a new one. The meta protocol may be any protocol for decision-making over rule modification. For the sake of presenting a concrete example, we chose a voting procedure as a meta protocol, that is, the meta protocol participants take a vote on a proposed modification of the object protocol rules. The participants of the meta protocol may initiate a meta–meta protocol to modify the rules of the meta protocol, or they may initiate a meta–meta–meta protocol to modify the rules of the meta–meta protocol, and so on. For simplicity, in this example *all* meta protocols are voting procedures (in other systems each meta protocol may be a different decision-making procedure). In general, in a k-level infrastructure, level 0 corresponds to the main (resource-sharing, in this example) protocol, while a protocol of level n, $0{<}n{\leq}k{-}1$ (voting, in this example), is created, by the protocol participants of a level m, $0{\leq}m{<}n$, in order to decide whether to modify the protocol rules of level $n{-}1$. The infrastructure for dynamic (resource-sharing) specifications is displayed in Figure 2.

[1]The action description D^{RS} in the input language of CCALC is available at `http://users.iit.demokritos.gr/~a.artikis/drs.zip`.

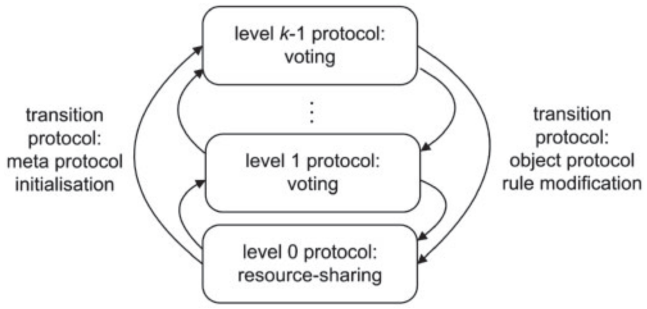
level k-1 protocol: voting
level 1 protocol: voting
level 0 protocol: resource-sharing
transition protocol: meta protocol initialisation
transition protocol: object protocol rule modification

FIGURE 2. A k-level infrastructure for dynamic specifications.

TABLE 3. A subset of the action signature of D^{RS} (Part B)

Variable	Domain
PL	Z^+
$SP, NSP, ASP, Motion$	a set of specification point ids
DoF_ID	a set of DoF ids
V	$\{for, against\}$
$Outcome$	$\{carried, not\_carried\}$
Simple fluent constant	**Domain**
$role\_of(Ag, PL)$	$\{subject, chair, fcs\}$
$dof(DoF\_ID, SP)$	a set of DoF values
$actual\_sp(PL)$	a set of specification point ids
$proposal(Ag, SP, PL)$, $properties(SP, PL)$	Boolean
$threshold\_d(PL)$, $threshold\_eu(PL)$, $eu(SP, PL)$	Z^+
Statically determined fluent constant	**Domain**
$powPropose(Ag, SP, PL)$, $powSecond(Ag, SP, PL)$, $perPropose(Ag, SP, PL)$, $oblPropose(Ag, SP, PL)$, $powDelcare(Ag, Motion, Outcome, PL)$	Boolean
$distance(SP, SP, PL)$	Z^+
Action constant σ^{act} (Boolean)	
$propose(Ag, NSP, PL)$, $second(Ag, NSP, PL)$, $vote(Ag, V, PL)$, $declare(Ag, Motion, Outcome, PL)$	

Apart from object and meta protocols, the infrastructure for dynamic specifications includes 'transition' protocols—see Figure 2—that is, procedures that express, among other things, the conditions in which an agent may successfully initiate a meta protocol (for instance, only the members of the board of directors may successfully initiate a meta protocol in some organizations), the roles that each meta protocol participant will occupy, and the ways in which an object protocol is modified as a result of the meta protocol interactions. The components of the infrastructure for dynamic specifications, level 0 protocol, level n protocol (n>0), and transition protocol, are discussed in Sections 5.1–5.4.

Table 3 shows a set of fluent and action constants of the action signature of D^{RS} that will be presented in the following sections. In order to distinguish between the protocol states of different protocol levels, we add a parameter, when necessary, in the representation of action and fluent constants, expressing the protocol level *PL*. For example, $role\_of(Ag, PL)$ expresses the role *Ag* occupies in level *PL*. It is unnecessary to modify the syntax of action and fluent constants that are part of the specification of a single protocol level (such as the action constant *request_floor* that concerns only level 0).

5.1 Level 0

For illustration purposes we chose a resource-sharing protocol—the specification of which was presented in Section 3—as a level 0 protocol. A protocol specification consists of the 'core' rules that are always part of the specification, and the *Degrees of Freedom* (*DoF*), that is, the specification components that may be modified at run-time. (A DoF can be seen, for example, as Vreeswijk's 'partial protocol specification' [86].)

A protocol specification with l DoF creates an l-dimensional specification space where each dimension corresponds to a DoF. A point in the l-dimensional specification space, or *specification point*, represents a complete protocol specification—a *specification instance*—and is denoted by an l-tuple where each element of the tuple expresses a 'value' of a DoF. Consider, for example, the resource sharing protocol with three DoF: the specification of the best candidate for the floor, the permission to assign the floor, and the permission to request a resource manipulation. The specification of these three protocol features may change at run-time—for instance, the best candidate may be determined randomly, on a first-come, first-served basis, priority may be given to subjects requesting a particular manipulation type, or to subjects that have not been sanctioned (these are possible values of the best candidate DoF). Regarding the second DoF, it may be forbidden to assign the floor to subjects that have been allocated the floor the last 3, 6 or 9 times. Finally, in this example, the holder may be permitted to request any type of resource manipulation from the FCS, or it may be permitted to request only the resource manipulation type expressed when it applied to the chair for the floor (these are possible values of the third DoF). In the resource-sharing example with these three DoF, a specification point is, for instance:

$$(fcfs, 3, any\_type)$$

According to the above specification point, the best candidate for the floor is determined on first-come, first-served basis (*fcfs*), the maximum number of permitted consecutive allocations of the floor to a subject is 3, while the holder is permitted to request any type of resource manipulation (*any_type*).

In this example, the first DoF has 4 values, the second DoF has 3 values and the third DoF has 2 values. Consequently, the specification space contains $4 \times 3 \times 2 = 24$ specification points. In other examples we could have chosen different DoF (and/or DoF values), such as the specification of the permission and the obligation to request the floor, or perform any other protocol action. The classification of a specification component as a DoF is application specific.

There are various reasons for which the agents may change at run-time the value of one or more DoF and thus the specification point. In the resource-sharing example, when the population of a system increases, the agents may decide to lower the limit of allowed consecutive allocations of the floor; when the number of agents violating the laws increases, it may be decided to give priority to agents that have not been sanctioned when computing the best candidate for the floor; etc. Furthermore, an agent may attempt to change at run-time the value of a DoF in order to satisfy its own private goals.

In any case, the value of one or more DoF, and thus the specification point, may change at run-time by means of a meta protocol. A discussion about meta protocols is presented in the following section.

We encode a protocol's specification points in $C+$ as follows:

$$\begin{array}{l}\textsf{caused } \mathit{dof}(\mathit{bc},\mathit{sp}_1){=}\mathit{fcfs}\\ \textsf{caused } \mathit{dof}(\mathit{per\_assign},\mathit{sp}_1){=}9\\ \textsf{caused } \mathit{dof}(\mathit{per\_mpt},\mathit{sp}_1){=}\mathit{any\_type}\end{array} \tag{9}$$

$$\begin{array}{l}\textsf{caused } \mathit{dof}(\mathit{bc},\mathit{sp}_2){=}\mathit{rmt}\\ \textsf{caused } \mathit{dof}(\mathit{per\_assign},\mathit{sp}_2){=}9\\ \textsf{caused } \mathit{dof}(\mathit{per\_mpt},\mathit{sp}_2){=}\mathit{expressed\_type}\end{array} \tag{10}$$

The simple fluent constant *dof* records the value of a DoF—$\mathit{dof}(\mathit{bc},\mathit{sp}_1){=}\mathit{fcfs}$, for example, denotes that, when the protocol's specification point is sp_1, the best candidate (*bc*) for the floor is determined on a first-come, first-served basis. *per_assign* denotes the DoF concerning the permission to assign the floor while *per_mpt* denotes the DoF concerning the permission to request a resource manipulation. *rmt* is a value of the best candidate DoF, indicating that priority is given to subjects requesting a particular resource manipulation type. The above formalization shows two example specification points: $\mathit{sp}_1{=}(\mathit{fcfs},9,\mathit{any\_type})$ and $\mathit{sp}_2{=}(\mathit{rmt},9,\mathit{expressed\_type})$.

Each DoF value is defined by a set of rules—consider the following example formalization:

$$\begin{array}{l}\textsf{caused } \mathit{perRequestMpt}(S,\mathit{FCS},M) \textsf{ if}\\ \quad \mathit{actual\_sp}(0){=}\mathit{ASP},\\ \quad \mathit{dof}(\mathit{per\_mpt},\mathit{ASP}){=}\mathit{any\_type},\\ \quad \mathit{role\_of}(\mathit{FCS},0){=}\mathit{fcs},\\ \quad \mathit{holder}(S)\end{array} \tag{11}$$

$$\begin{array}{l}\textsf{caused } \mathit{perRequestMpt}(S,\mathit{FCS},M) \textsf{ if}\\ \quad \mathit{actual\_sp}(0){=}\mathit{ASP},\\ \quad \mathit{dof}(\mathit{per\_mpt},\mathit{ASP}){=}\mathit{expressed\_type},\\ \quad \mathit{role\_of}(\mathit{FCS},0){=}\mathit{fcs},\\ \quad \mathit{holder}(S),\\ \quad \mathit{requested}(S){=}M\end{array} \tag{12}$$

Rule (11) defines the *any_type* value of the DoF concerning the permission to request a resource manipulation type, while rule (12) defines the *expressed_type* value of this DoF. The *perRequestMpt* fluent constant expresses the permission to request a resource manipulation of a particular type (for instance, storing files of a particular type on a shared storage device, or executing applications of a particular type on a shared processor). According to rule (11), a subject S is permitted to request any resource manipulation type M if S is the holder of the resource. According to rule (12), the holder S of the resource is permitted to request only the manipulation type denoted when it applied to the chair for the floor (see the last condition of rule (12)—recall that the *requested* fluent constant records the subjects' requests for the floor). The simple fluent constant $\mathit{actual\_sp}(\mathit{PL})$ records the actual specification point of protocol level PL. Rules (11) and (12) are replaceable in the sense that the participants of the resource-sharing protocol may activate/deactivate one of them, at run-time, by changing the specification point in a way that the value of the DoF concerning the permission to request a resource manipulation type is modified. For example, moving from specification point sp_1 to sp_2 deactivates rule (11) and activates rule (12). The ways in which a specification point may change at run-time are presented next.

5.2 *Level n*

To provide a concrete example, we chose a three-level infrastructure for dynamic specifications. Moreover, both levels 1 and 2 are voting procedures, such as that presented in [67]. A presentation of a voting procedure specification is beyond the scope of this article. Briefly, we assume a simple procedure including a set of voters casting their votes, 'for' or 'against' a particular motion, which would be in this example a proposed specification point change in level $n-1$, and a chair counting the votes and declaring the motion carried or not carried, based on the *standing rules* of the voting procedure—simple majority, two-thirds majority, etc.

Our infrastructure allows for the modification of the specification of all protocol levels apart from the top one. Consequently, we define DoF for all protocol levels apart from the top one. For the protocol of level 1—voting—we chose to set as a DoF the standing rules of the voting procedure. In other words, a level 2 protocol may be initiated in order to decide whether level 1 voting should become, say, simple majority instead of two-thirds majority. Note that the voting procedures of levels 1 and 2 may not always have the same set of rules. For example, at a particular time-point level 2 voting may require a simple majority whereas level 1 voting may require a two-thirds majority (as mentioned above, the standing rules of level 1 constitute a DoF and thus the specification of this part of the protocol may change at run-time).

5.3 *Transition protocol*

In order to change the specification point of level m, $m \geq 0$ (for example, to change the value of the best candidate DoF from *fcfs* to *rmt*), that is, in order to start a protocol of level $m+1$, the participants of level m need to follow a 'transition' protocol—see Figure 2. The infrastructure for dynamic specifications presented in this article requires two types of transition protocol: one leading from the resource-sharing protocol to a voting one (level 0 to level 1 or 2), and one leading from one voting protocol to the other (level 1 to level 2). We will only present the first type of transition protocol; the latter type of protocol may be specified in a similar manner. An example transition protocol leading from resource-sharing to voting can be briefly described as follows. A subject S of the resource-sharing protocol proposes that the specification point of this level (or of level 1) changes. If S is empowered to make such a proposal, then the modification is directly accepted, without the execution of a meta protocol, provided that another subject exercises its power to second the proposal, and no other subject exercises its power to object to the proposal. If S is not empowered to make the proposal, or if the proposal is not seconded, then it is ignored. If the proposal is seconded, and there is an objection, then an argumentation procedure commences, the topic of which is the proposed specification point change, the proponent of the topic is S, the subject that made the proposal, and the opponent is the subject that objected to the proposal. The argumentation procedure is followed by a meta protocol (level 1 or 2) in which a vote is taken on the proposed specification point change.

In this example transition protocol we have specified the power to propose a specification point change as follows:

$$\begin{aligned}
&\textsf{caused}\ \mathit{powPropose}(Ag, NSP, PL)\ \textsf{if} \\
&\quad \mathit{role\_of}(Ag, 0) = \mathit{subject}, \\
&\quad \mathit{actual\_sp}(PL) = ASP,\ ASP \neq NSP, \\
&\quad \mathit{protocol}(PL{+}1) = \mathit{idle}, \\
&\quad \mathit{properties}(NSP, PL)
\end{aligned} \tag{13}$$

An agent Ag is empowered to propose that the specification point of protocol level PL becomes NSP if the following conditions are satisfied. First, Ag occupies the role of subject in level 0. In this example, the chair of the resource-sharing protocol (level 0) is not empowered to propose a change of the specification point. Secondly, the actual specification point, *ASP*, is different from *NSP*. Thirdly, there is no protocol taking place in level *PL+1*. A *protocol*(*PL*) simple fluent constant records whether a protocol of level *PL* is idle or executing. A protocol for changing the specification point of level *PL*, that is, a protocol of level *PL+1*, may commence only if there is no other protocol of level *PL+1* taking place.

Fourthly, the specification instance corresponding to specification point *NSP* of level *PL* satisfies a set of properties—*properties* is a simple fluent constant. The properties that a specification instance should satisfy are application specific. We may require, for example, that an agent in never forbidden and obliged to perform the same action, a floor control mechanism is 'safe' and 'fair' [24], and so on. In this example, *properties* is false for specification points of the form $(rmt, *, any\_type)$ ($*$ denotes any value of the second DoF), as these points correspond to specification instances that are, in some sense, 'inconsistent': priority for access to the resource is given to subjects with an expressed request of resource manipulation M (the value of the best candidate DoF is *rmt*), while it is permitted to actually request from FCS a different type of manipulation M' (the value of the DoF concerning the permission for actual resource manipulation is *any_type*).

CCALC is an automated reasoning tool allowing for proving protocol properties—recall that in Section 4 we proved properties of a specification instance of the resource-sharing protocol by means of CCALC's query computation. Assuming that a protocol's specification points are known before the commencement of the protocol execution, we may determine at design-time, with the use of CCALC, whether the specification instance corresponding to each specification point of a protocol level satisfies a set of desirable properties. Accordingly, we may set the value of the *properties* fluent constant (which is part of the transition protocol specification), thus avoiding, at run-time, the (time-consuming) task of proving protocol properties.

In other examples the specification of the power to propose a specification point change could have different, or additional conditions further constraining how a protocol specification may change at run-time. For instance, it may be required that, for any protocol level, the specification point does not change 'too often', or there may be an upper limit on the number of specification point changes (proposed by an agent). In Section 5.4 we present a way of further constraining the process of specification point change. Other ways of achieving that, such as the ones described above, could have been formalized.

A proposal for specification point change needs to be seconded by another subject having the institutional power to second the proposal in order to be directly enacted, or initiate the argumentation procedure leading to voting. We chose to specify the power to second a proposal for specification point change as follows:

$$\begin{array}{l}\textsf{caused}\ powSecond(Ag, NSP, PL)\ \textsf{if}\\ \quad role\_of(Ag, 0){=}subject,\\ \quad proposal(Ag', NSP, PL),\\ \quad Ag \neq Ag'\end{array} \tag{14}$$

Ag is empowered to second any proposal for specification point change made by some other Ag'. The *proposal* simple fluent constant records proposals made by empowered agents (subjects, in this example).

We have specified that any subject is empowered to object to a proposal for specification point change.

Exercising the power to object to a proposal for specification point change initiates an argumentation procedure, the topic of which is the proposed change. It goes beyond the scope of this article to present a specification of an argumentation procedure; see [6] for an example formalization of such a procedure.

The completion of the argumentation taking place in the context of a transition protocol initiates a meta protocol. The latter protocol is a voting procedure concerning a proposed specification point change. The agents participating in this procedure and the roles they occupy are determined by the transition protocol that results in the voting procedure. We chose to specify that the chair of the resource-sharing protocol becomes the chair of the voting procedure. Furthermore, the agents occupying the role of voter, thus having the power to vote, are the subjects that have not been sanctioned for exhibiting ‘anti-social’ behaviour, that is, performing forbidden actions or not complying with obligations:

$$\begin{array}{l}\textsf{caused } \mathit{role\_of}(\mathit{Ag},\mathit{PL}) = \mathit{voter} \textsf{ if}\\ \quad \mathit{role\_of}(\mathit{Ag},0) = \mathit{subject},\\ \quad \mathit{protocol}(\mathit{PL}) = \mathit{executing},\ \mathit{PL} > 0,\\ \quad \neg \mathit{sanctioned}(\mathit{Ag})\end{array} \tag{15}$$

The value of a *protocol*(*PL*) constant becomes *executing*, in the case where $PL>0$, at the end of the argumentation of the transition protocol that led to level *PL*. We chose not to relativize the constant recording sanctions to a protocol level. Therefore, the simple fluent constant *sanctioned* records ‘anti-social’ behaviour exhibited at any protocol level. Depending on the employed treatment of sanctions, ‘anti-social’ behaviour may be permanently recorded, thus permanently depriving a subject of participating in a meta level, or it may be temporarily recorded, thus enabling subjects to participate in a meta level after a specified period has elapsed from the performance of forbidden actions or non-compliance with obligations.

Clearly, meta level role assignment may be specified in other ways. For example, it may be the case that agents that recently joined the system are not admitted in the meta level, or that priority is given to subjects that have had the least time accessing the shared resources, etc. In general, meta level role assignment can be as complex as required by the application under consideration.

The fact that an agent may successfully start a protocol of level $m+1$ by proposing a change of the specification point of level m, does not necessarily imply that the specification point of level m will be changed. It is only if the motion of level $m+1$—which is the proposed specification point for level m—is carried, that the specification point of level m will be changed. Consider the following rule expressing the outcome of a voting procedure of level $m+1$, that takes place in order to change the specification point in level m to *NSP*:

$$\begin{array}{l}\mathit{declare}(\mathit{VC},\mathit{Motion},\mathit{carried},\mathit{PL}+1) \textsf{ causes } \mathit{actual\_sp}(\mathit{PL}) = \mathit{NSP} \textsf{ if}\\ \quad \mathit{powDeclare}(\mathit{VC},\mathit{Motion},\mathit{carried},\mathit{PL}+1),\\ \quad \mathit{Motion} = \mathit{NSP}\end{array} \tag{16}$$

Exercising the power to declare the motion carried in level $PL+1$ results in changing the specification point in level *PL* to *NSP*, provided that the motion concerned the adoption of *NSP*. If the chair of the voting procedure *VC* did not declare the motion carried, or was not empowered to make the declaration, then the specification point would not have changed in level *PL*.

Exercising the power to declare the motion carried in a meta level m may have additional effects. For example, we may explicitly specify whether or not deactivating a rule, by changing the specification point of protocol level $m-1$, results in removing the effects of the rule that were produced prior to the rule deactivation. A discussion about specification change operations is presented in the last section of the article.

5.4 *Constraining the process of run-time specification modification*

As already mentioned, all protocol levels apart from the top one have DoF and, therefore, their specification may be modified at run-time. In this section, we present ways of evaluating a proposal for specification point change. Moreover, we present ways of constraining the enactment of proposals that do not meet the evaluation criteria.

One way of evaluating a proposal for specification point change is by modelling a dynamic protocol specification as a *metric space* [14]. More precisely, we compute the 'distance' between the proposed specification point and the actual point. We constrain the process of specification point change by permitting proposals only if the proposed point is not too 'far' from/'different' to the actual point. The motivation for formalizing such a constraint is to favour gradual changes of a system specification. In what follows we describe how we may compute the distance between two specification points, and the conditions in which a proposal for specification point change is considered permitted.

We may follow two steps to compute the distance between two specification points sp and sp', each represented as an l-tuple, where each element of the tuple expresses a DoF value. First, we may transform sp and sp' to l-tuples of non-negative integers qsp and qsp', respectively. To achieve that, we may define an application-specific function v, that 'ranks' each DoF value, that is, associates every DoF value with a non-negative integer. The i-th element of qsp, qsp_i, has the value of $v(sp_i)$, where sp_i is the i-th element of sp (respectively, the i-th element of qsp', qsp'_i, has the value of of $v(sp'_i)$, where sp'_i is the i-th element of sp'). Secondly, we may employ a *metric* (or *distance function*), such as the Euclidean metric, to compute the distance between qsp and qsp'. A list of metrics may be found in [14]. Depending on the employed metric, we may add weights on the DoF—for instance, we may require that the computation of the distance between qsp and qsp' is primarily based on the best candidate DoF rather than the other two DoF. The distance between sp and sp' is the distance between qsp and qsp'.

Alternatively, we may compute the distance between two specification points by defining an application-specific metric that does not necessarily rely on a quantification DoF values. A mathematical basis for measuring distances between specification points may be found in [51], for example.

We may constrain run-time specification point change as follows:

$$\begin{aligned}
&\textsf{caused}\ \mathit{perPropose}(Ag, NSP, PL)\ \textsf{iff}\\
&\qquad \mathit{powPropose}(Ag, NSP, PL),\\
&\qquad \mathit{actual\_sp}(PL) = ASP,\\
&\qquad \mathit{distance}(NSP, ASP, PL) \leq \mathit{threshold\_d}(PL)
\end{aligned} \tag{17}$$

$\mathit{perPropose}(Ag, NSP, PL)$ is a statically determined fluent constant denoting whether agent Ag is permitted to propose that the specification point of level PL becomes NSP. *distance* is a statically determined fluent constant computing the distance between any two specification points of a protocol level. The definition of *distance* includes a $C+$ representation of a chosen metric. $\mathit{threshold\_d}(PL)$ is a simple fluent constant recording the maximum distance that a proposed specification point should have from the actual point in level PL. Different protocol levels may have different *threshold_d* values and different metrics. According to rule (17), an agent Ag is permitted to propose that the specification point of level PL becomes NSP, if and only if Ag is empowered to make this proposal, and the distance between NSP and the actual specification point of level PL, ASP, is less or equal to a specified threshold.

Note that the maximum allowed distance between the actual point and a proposed point may change during the system execution. Consider, for example, snapshots (a)–(d) of the Euclidean

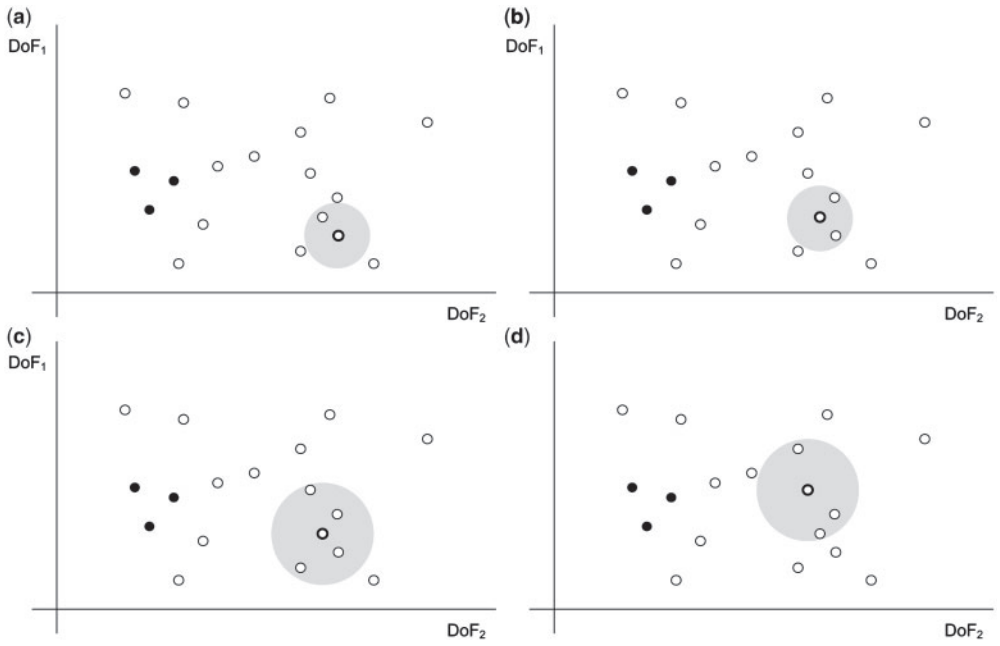

FIGURE 3. Run-time change of actual specification point and maximum allowed distance between the actual specification point and a proposed specification point.

specification space with two DoF shown in Figure 3. Black circles denote specification points representing specification instances that do not satisfy a set of protocol properties—consequently, an agent is not empowered to propose that the actual specification point becomes one of those points (see Section 5.3). White circles, on the other hand, denote specification points representing specification instances that satisfy the required protocol properties. The circle with the thick line denotes the actual specification point. The grey area denotes the maximum allowed distance between the actual specification point and a proposed point—this is expressed by *threshold_d*. Initially—snapshot (a)—only one specification point, *sp*, is within the grey area, that is, it is permitted, under certain circumstances, to move only to *sp*. Then—snapshot (b)—the actual specification point moves to *sp*. At this time, two specification points are within the grey area. Following this specification point change, the value of *threshold_d* increases—snapshot (c)—that is, the size of the grey area increases, offering more options for permitted specification point change. In some systems, designated agents, such as 'institutional agents' [11], may increase, temporarily perhaps, the value of *threshold_d* in order to allow for a greater system specification change, possibly as a result of sensing a substantial change of environmental, social or other conditions. In other systems, changing the value of *threshold_d* may be realized in a manner similar to that used for changing the actual specification point. Snapshot (d) shows that the actual specification point moves to a point that could not have been directly reached had the value of *threshold_d* not increased, assuming that agents abide by the rules governing specification point change. Note that, in general, the members of a system may adopt *any* specification point that satisfies certain protocol properties—in the present example the agents may adopt any point depicted as a white circle. Moving outside of the grey area, however, is forbidden and the agent that proposed such a specification point change may be subject to penalty.

The designers of a system may further constrain the process of run-time specification modification by permitting a proposal for specification point change only if the expected system utility associated with the proposed point is ‘acceptable’ (in a sense to be specified below). Consider the following formalization:

$$\begin{array}{l} \textsf{caused}\ \mathit{perPropose}(Ag,NSP,PL)\ \textsf{if} \\ \quad \mathit{powPropose}(Ag,NSP,PL), \\ \quad \mathit{actual\_sp}(PL){=}ASP, \\ \quad \mathit{distance}(NSP,ASP,PL) \leq \mathit{threshold\_d}(PL), \\ \quad \mathit{eu}(NSP,PL) \geq \mathit{threshold\_eu}(PL) \end{array} \tag{18}$$

$$\begin{array}{l} \textsf{caused}\ \mathit{perPropose}(Ag,NSP,PL)\ \textsf{if} \\ \quad \mathit{powPropose}(Ag,NSP,PL), \\ \quad \mathit{actual\_sp}(PL){=}ASP, \\ \quad \mathit{distance}(NSP,ASP,PL) \leq \mathit{threshold\_d}(PL), \\ \quad \mathit{eu}(NSP,PL) > \mathit{eu}(ASP,PL) \end{array} \tag{19}$$

$$\textsf{default}\ \neg \mathit{perPropose}(Ag,NSP,PL) \tag{20}$$

The simple fluent constant *eu*(*NSP*, *PL*) expresses the expected system utility associated with the specification point *NSP* of level *PL*, that is, the system utility expected to be achieved under the specification instance corresponding to *NSP*. The utility of a system may be defined in various ways; in a resource-sharing protocol, for instance, the system utility may be defined in terms of the average time for servicing a request for the floor. An approach to defining system utility in a resource allocation setting, and estimating the expected system utility, may be found in [82], for example. The simple fluent constant *threshold_eu*(*PL*) expresses the minimum allowed expected system utility in level *PL*. The first three conditions of rules (18) and (19) are the same as the conditions of rule (17). According to rules (18)–(20), an agent *Ag* is permitted to propose that the specification point of level *PL* becomes *NSP*, if and only if *Ag* is empowered to make this proposal, *NSP* is not too ‘far’ from the actual point *ASP*, and

- the expected system utility associated with *NSP* exceeds a specified threshold, or
- the expected system utility associated with *NSP* is greater than the expected system utility associated with *ASP*.

Rules (18)–(20) are but one possible way to formalize the permission to propose a specification point change. We could have equally chosen to adopt, say, only rules (18) and (20), or rules (19) and (20).

Care should be taken when specifying the thresholds for distance and expected system utility. For example, it might be the case that, for certain values of these thresholds, it is never permitted to move from most specification points to any other point. CCALC is very useful in selecting the appropriate values for these thresholds. We may use CCALC to prove properties of a transition protocol under certain values of the distance and expected system utility thresholds, such as that, under certain circumstances it is permitted to move from some/all specification points to another point. In the next section, we demonstrate the use of CCALC for proving properties of an infrastructure for dynamic protocol specification, including a transition protocol specification.

The system utility expected to be achieved under a specification instance depends on various conditions such as the size of the population of a system, the frequency of rule violation, and the available resources. Such conditions may fluctuate during a system’s lifetime. Consequently, the system utility expected to be achieved under a specification instance may change over time.

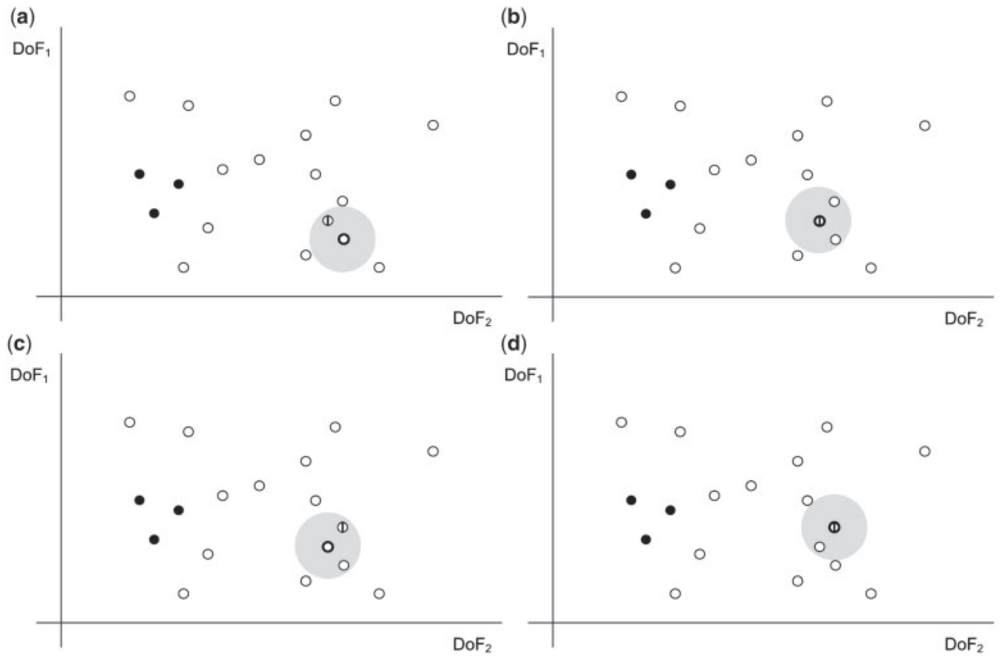

FIGURE 4. Run-time change of desired and actual specification points.

Figure 4 illustrates this issue; this figure shows four new snapshots of the Euclidean specification space presented in Figure 3. Recall that black circles denote specification points representing specification instances that do not satisfy a set of protocol properties, white circles denote specification points representing specification instances that do satisfy the required protocol properties, the circle with the thick line denotes the actual specification point, and the grey area denotes the maximum allowed distance between the actual specification point and a proposed point. The circle with the vertical line denotes the 'desired' specification point, that is, the specification point corresponding to the specification instance with the maximum expected system utility. To avoid clutter, we do not show the expected system utility of each specification point. The snapshots of Figure 3 show three specification point changes. First, the system members change the actual specification point in a way that it coincides with the desired point. Then, the desired specification point changes—such a change could be the result of a change in environmental conditions, for example. (In the present $C+$ formalization, we have written simple rules to compute the expected system utility associated with each specification point, and thus determine the desired point. In other examples, institutional agents, or other types of agent may compute the expected system utility associated with each specification point, and therefore determine the desired point, under different environmental conditions.) Finally, the system members change once more the actual point in a way that it reaches the desired one.

Apart from the desired point, the threshold concerning expected system utility (*threshold_eu*) may change over time, using institutional agents or some other means.

Similar to rules (18)–(20), we may express the permission to *second* a proposal for specification point change, or the *obligation to object* to such a proposal. Moreover, when the expected system utility associated with the actual specification point is below the specified threshold (*threshold_eu*), we could impose an *obligation to propose* a specification point change that, if accepted (the proposal), would change the actual point in a way that the associated expected system utility is increased. Below is a way of formalizing such an obligation:

$$\begin{aligned}&\textsf{caused}\ \mathit{oblPropose}(Ag,NSP,PL)\ \textsf{if}\\ &\quad \mathit{perPropose}(Ag,NSP,PL),\\ &\quad \mathit{actual\_sp}(PL)=ASP,\\ &\quad eu(ASP,PL)<\mathit{threshold\_eu}(PL)\end{aligned} \tag{21}$$

The statically determined fluent constant *oblPropose* expresses the obligation to propose a specification point change. According to the above rule, an agent *Ag* is obliged to propose that the specification point of level *PL* becomes *NSP* if *Ag* is permitted to propose the adoption of *NSP* (see rules (18)–(20) for an example definition of the permission to propose a specification point change), and the expected system utility associated with the actual specification point of level *PL*, *ASP*, is less than the *threshold_eu*(*PL*) value.

Note that *Ag*'s obligation to propose a specification point change may be terminated, even if *Ag* does not discharge it: a specification point with greater expected system utility may be adopted due to the proposal of some other agent. In this case the last condition of rule (21) may cease to hold and thus *Ag* will no longer be obliged to propose a specification point change.

6 Proving properties of the dynamic resource-sharing protocol

Specifying a dynamic protocol in $C+$ allows us to prove various properties of the specification. (Recall that in Section 4 we proved properties of a static resource-sharing protocol specification.) We may prove properties of the specification instances of level 0 and level n, $n>0$ and the transition protocols. Below we present a few example properties proven of the presented dynamic resource-sharing protocol, by means of CCALC query computation.

Recall that the $C+$ action description D^{RS}, expressing the dynamic resource-sharing protocol, defines a three-level infrastructure; level 0 (resource-sharing) has three DoF, the specification of the best candidate for the floor, the permission to assign the floor, and the permission to request a resource manipulation. Level 1 voting has a single DoF, the standing rules of the voting procedure, while level 2 voting has no DoF. To conduct computational experiments, one has to make specific choices for a set of parameters. For a concrete illustration we will present here experiments in which, for example, the distance between two specification points is computed with the use of a weighted Manhattan metric. Arbitrary values were chosen for the threshold distance between the actual specification point and a proposed point, as well as the threshold expected system utility. Moreover, the only specification points representing inconsistent, in some sense, specification instances are the level 0 points of the form $(rmt, *, any\_type)$ (see Section 5.3). Other choices concerning the experimental parameters could of course have been made, and the experiments repeated for those.

The mechanism for run-time specification modification presented in the previous section offers refined control to the designer of a multi-agent system. For instance, the mechanism guarantees that a specification point expressing a specification instance that does not satisfy a set of application-dependent criteria will never be adopted. Consider the following example.

PROPERTY 6.1
A specification point of the form $(rmt, *, any\_type)$ will never be adopted.

We instruct CCALC to compute all paths $s_0 \varepsilon_0 s_1 \ldots s_{m-1} \varepsilon_{m-1} s_m$ of D^{RS} such that

- $s_0 \models actual\_sp(0)=ASP \wedge$
  $(dof(bc,ASP) \neq rmt \vee dof(per\_mpt,ASP) \neq any\_type) \wedge$
  $\forall Ag, SP\ \neg proposal(Ag,SP,0)$ and
- $s_m \models actual\_sp(0)=NSP \wedge$
  $dof(bc,NSP)=rmt \wedge dof(per\_mpt,NSP)=any\_type$

Recall that $dof(D,SP)$ records the value of a DoF D under specification point SP, while *proposal* records proposals for specification point change made by empowered agents (see Section 5). In this case, we are interested in computing all paths from a state in which the actual specification point of protocol level 0—resource-sharing—is not of the form $(rmt, *, any\_type)$ and no proposal for specification point change has been made, to a state in which the actual specification point of level 0 is of the form $(rmt, *, any\_type)$.

CCALC finds no solution to this query. As mentioned above, a specification point of the form $(rmt, *, any\_type)$ represents an inconsistent specification instance. Consequently, the last condition of rule (13), expressing the power to propose the adoption of a specification point, is not satisfied. Since an agent is never empowered to propose the adoption a specification point of the form $(rmt, *, any\_type)$, a meta-protocol concerning such a specification point change will never be initiated and, therefore, a point of this form will never be adopted.

The proposed mechanism for run-time specification modification further guarantees that, under certain circumstances, the specification of a protocol level will never change ‘dramatically’, and that a specification change will always improve the expected system utility, or maintain a standard of expected system utility. Consider the following examples.

PROPERTY 6.2
A specification point that is too ‘far’ from/‘different’ to the actual specification point will never be adopted as long as all agents occupying the role of subject comply with the transition protocol rules.

We instruct CCALC to compute all paths $s_0 \varepsilon_0 s_1 \ldots s_{m-1} \varepsilon_{m-1} s_m$ of D^{RS} such that

- $s_0 \models actual\_sp(PL)=ASP \wedge \forall Ag, SP\ \neg proposal(Ag,SP,PL)$ and
- $s_m \models actual\_sp(PL)=NSP \wedge NSP \neq ASP$.

In other words, we are interested in computing all paths from a state in which the actual specification point of protocol level PL is ASP and no proposal for specification point change has been made, to state in which the actual specification point of protocol level PL changes to NSP.

In every solution in which all subjects comply with the transition protocol prohibitions, the distance between the new specification point, NSP, and the previous point, ASP, does not exceed the specified threshold for level PL. See rules (18)–(20) for the specification of permission in the transition protocol.

PROPERTY 6.3
A specification change will always improve the expected system utility or maintain a standard of expected system utility as long as all agents occupying the role of subject comply with the transition protocol rules.

We instruct CCALC to compute the same query as in the case of Property 6.2. In every solution in which all subjects comply with the transition protocol prohibitions, the system utility expected to be achieved under the new specification point is greater than the system utility expected to be achieved under the previous specification point, or is greater than the specified threshold.

Note that in Properties 6.2 and 6.3 we make no assumptions about the behaviour of the agents outside the transition protocol—for example, we do not assume that subjects comply with the resource-sharing protocol rules.

The circumstances under which the proposed mechanism for run-time specification modification guarantees that a specification change will always improve the expected system utility, or maintain a standard of expected system utility, and that the specification of a protocol level will never change 'dramatically', depend on the specification of the transition protocol. For example, it may be the case that these guarantees are offered even when only a small subset of the agents comply with the transition protocol rules.

We may further prove that the mechanism for run-time specification modification guarantees that the specification of a protocol level does not change too 'often', that the modification process terminates within a fixed number of steps, or that the specification of a protocol level always changes according to the decision taken at the meta level. Consider the following example.

PROPERTY 6.4
Exercising the power to declare the outcome of the voting procedure of level n carried always changes the specification point of level $n-1$.

We instruct CCALC to find all states s' such that

- (s,ε,s') is a transition of D^{RS},
- $s \models \mathit{powDeclare}(\mathit{VC},\mathit{NSP},\mathit{carried},\mathit{PL}) \wedge \mathit{actual\_sp}(\mathit{PL}-1)=\mathit{ASP}$, and
- $\varepsilon \models \mathit{declare}(\mathit{VC},\mathit{NSP},\mathit{carried},\mathit{PL})$.

For every state s' computed by CCALC we obtain

$$s' \models \mathit{actual\_sp}(\mathit{PL}-1)=\mathit{NSP}$$

which denotes that the specification point of level PL has changed (in all solutions $\mathit{ASP} \neq \mathit{NSP}$). Rule (16) expresses the effects of exercising the power to declare the outcome of a voting procedure carried.

7 Animating the dynamic resource-sharing protocol

Apart from proving properties of a dynamic protocol specification, CCALC's query computation—in particular the computation of 'prediction' (temporal projection) queries, such as the queries used to prove Properties 4.2 and 6.4—allows us to calculate, at run-time, the agents' powers, permissions and obligations. Such information may be publicized to the members of a system, and may be provided by a central server or in various distributed configurations. (Further discussion of these architectural issues is outside the scope of this article.) In this section, we present an example execution (run) of the dynamic resource-sharing protocol, and the results obtained by CCALC's prediction query computation.

The narrative of events of the presented run is displayed in Table 4. The events of transition protocols, *propose*, *second*, *object*, level 1 and level 2 protocols, *vote* and *declare*, are indented. The last argument of a level 1 or 2 event indicates the protocol level in which the event took place.

TABLE 4. Run of dynamic resource-sharing protocol

Time	Action
4	$propose(sub_3, sp_{26}, 1)$
10	$object(sub_1, sp_{26}, 1)$
17	$second(sub_5, sp_{26}, 1)$
	transition protocol argumentation
28	$vote([sub_2, sub_3, sub_4, sub_5, sub_6], for, 2)$
30	$vote(sub_1, against, 2)$
31	$declare(c, sp_{26}, carried, 2)$
35	$propose(sub_5, sp_{17}, 0)$
36	$second(sub_3, sp_{17}, 0)$
39	$object(sub_2, sp_{17}, 0)$
	transition protocol argumentation
51	$vote([sub_3, sub_4, sub_6], for, 1)$
53	$vote([sub_1, sub_2], against, 1)$
54	$declare(c, sp_{17}, carried, 1)$
55	$request\_floor(sub_1, c, app_A)$
56	$request\_floor(sub_2, c, app_A)$
58	$request\_floor(sub_5, c, app_A)$
60	$request\_floor(sub_3, c, app_A)$
62	$assign\_floor(c, sub_3)$

To avoid clutter, we group the votes that are in favour of (respectively, against) a motion. In the initial state of the presented run:

- The actual specification point of level 0 is $(fcfs, 3, any\_type)$, that is, the best candidate for the floor is determined on a first-come, first-served basis, the maximum number of permitted consecutive allocations of the floor is 3, while the holder is permitted to request any type of resource manipulation (see rule (11)).
- The actual specification point of level 1 is (3_4m), that is, a 75% majority is required (recall that level 1 has a single DoF).

Level 2 voting does not have a DoF. In these experiments level 2 voting requires a 75% majority. Details about the choices we made concerning the remaining experimental parameters were given in the previous section.

The present example includes a chair c and six subjects sub_1–sub_6. sub_3 and sub_5 aim to change the specification point of level 0 in a way that the value of the best candidate DoF becomes *random*, that is, the best candidate for the floor is chosen randomly from the list of subjects having pending floor requests. Before attempting to change the specification point of level 0, sub_3 and sub_5 attempt to change the specification point of level 1 in a way that level 1 voting requires simple majority as opposed to 75% majority. Therefore, fewer votes will be required in level 1 when sub_3 and sub_5 propose to change the specification point of level 0. The proposal for changing the specification point of level 1 takes place at time-point 4—specification point sp_{26} expresses that the standing rules require simple majority. At that time sub_3 is empowered to make the proposal (see rule (13)). Furthermore, sub_3 is permitted to exercise its power (see rules (18)–(20)) because: (i) the distance, in level 1, between the actual specification point (75% majority) and the proposed point sp_{26} (simple majority) is less than the chosen threshold ($threshold\_d(1)$), and (ii) the expected system utility associated with sp_{26} is greater than the corresponding threshold ($threshold\_eu(1)$). sub_3's proposal is followed by an objection, a secondment and an argumentation. Then level 2 voting commences; the motion is the adoption of sp_{26} in level 1. sub_2–sub_6 vote for the motion while sub_1 votes against it. At time-point 31 the motion of level 2 is declared carried (recall that level 2 requires 75% majority) and thus the specification point of level 1 becomes sp_{26} (see rule (16)), meaning that the standing rules of level 1 change to simple majority.

sub_5 proposes at time-point 35 the adoption of specification point sp_{17}=(*random*,*3*,*any_type*) in level 0. sub_5 is empowered to make the proposal at that time. However, sub_5 is not permitted to exercise its power because the expected system utility of sp_{17} is less than *threshold_eu*(*0*), and less than the expected system utility associated with the actual point (*fcfs*,*3*,*any_type*) of level 0. Consequently, sub_5 is sanctioned for performing a forbidden action and thus cannot participate in level 1 to vote (see rule (15)). sub_3, sub_4 and sub_6 vote for the motion of level 1—the adoption of sp_{17} in level 0—while sub_1 and sub_2 vote against it. At time-point 54 the motion of level 1 is declared carried (recall that level 1 now requires a simple majority) and thus the specification point of level 0 becomes sp_{17}, meaning that the best candidate for the floor is chosen randomly from the list of subjects having pending floor requests.

Following the specification point change of level 0, sub_1, sub_2, sub_5 and sub_3 exercise their power to request the floor, all of them requiring to run applications of type *A* on the shared processor (see the third argument of *request_floor*). At time 62 *c* exercises its power (see rule (5)) to assign the floor to sub_3.

8 Summary, related and further work

We presented an infrastructure for dynamic specifications for open MAS, that is, specifications that are developed at design-time but may be modified at run-time by the members of a system. Any protocol for open MAS may be in level 0 of our infrastructure, whereas any protocol for decision-making over specification change may be in level n, $n > 0$. The level 0/level n/transition protocols can be as complex/simple as required by the application in question.

We employed $C+$, an action language with explicit transition system semantics to formalize dynamic specifications. Moreover, we employed CCALC, an automated reasoning tool for proving properties of the specifications and assimilating narratives of events. On the one hand, therefore, we may provide design-time services, proving properties, such as termination, of the various protocols of our infrastructure, and on the other hand, we may offer run-time services, calculating the system state current at each time, including the powers, permissions and obligations of the agents.

This article is a significantly extended and updated version of [3, 4]. We presented here in greater detail the way we may use $C+$ to formalize dynamic specifications and the way we may use CCALC to execute such specifications, we employed more illustrative and extensive examples of our infrastructure for dynamic specifications, and present, in what follows, a considerably more extensive comparison of our work with related approaches. Furthermore, we formulated transition protocols in a very different way than our previous work. In [3, 4] we modelled a dynamic specification as a metric space in order to forbid agents to propose the adoption of a specification point that is too 'far' from/'different' to a desired specification point. The rationale for such a constraint was that the system designers could control, in some sense, the process of run-time specification modification by indicating which specification point is desired and, according to the aforementioned constraint, requiring that agents adopt specification points 'close'/'similar' to the desired one. In this article, we used metric spaces in a different way: we modelled a dynamic specification as a metric space in order to forbid agents to propose the adoption of a specification point that is too ‘far’ from/‘different’ to the *actual* specification point. In this way we favour gradual changes of a system specification. Moreover, we placed additional constraints (not related to metric spaces) on transition protocols: agents are forbidden to propose the adoption of a specification point under which the expected system utility is below a certain threshold. The presented formalization of transition protocols offers a finer level of control to the system designers and is applicable to a much wider range of MAS than that presented in our earlier work.

Chopra and Singh [16] have presented a way of adapting ‘commitment protocols’ [77–79, 83] according to context, or the preferences of agents in a given context. They formalize protocols and ‘transformers’, that is, additions/enhancements to an existing protocol specification that handle some aspect of context or preference. Depending on the context or preference, a protocol specification is complemented, at *design-time*, by the appropriate transformer thus resulting in a new specification. Unlike Chopra and Singh, we are concerned here with the *run-time* adaptation of a protocol specification and, therefore, we developed an infrastructure—meta protocols, transition protocols—to achieve that.

Several approaches have been proposed in the literature for run-time specification change of norm-governed MAS. Serban and Minsky [73], for example, have presented a framework for law change in the context of ‘Law-Governed Interaction’ (LGI) [61, 62, 64, 66, 87]. LGI is an abstract regulatory mechanism that satisfies the following principles: statefulness, that is, the regulatory mechanism is sensitive to the history of interaction between the regulated components, decentralization, for scalability, and generality, that is, LGI is not biased to a particular type of law. LGI is an abstraction of a software mechanism called Moses [63, 65] which can be used to regulate distributed systems. Moses employs regimentation devices that monitor the behaviour of agents, block the performance of forbidden actions and enforce compliance with obligations.

It has been argued [52] that regimentation is rarely desirable (it results in a rigid system that may discourage agents from entering it [68]), and not always practical. In any case, violations may still occur even when regimenting a MAS (consider, for instance, a faulty regimentation device). For all of these reasons, we have to allow for non-compliance and sanctioning and not rely exclusively on regimentation mechanisms.

Serban and Minsky were concerned in [73] with architectural issues concerning law change. They presented a framework with which a law change is propagated to the distributed regimentation devices, taking into consideration the possibility that during a ‘convergence period’ various regimentation devices operate under different versions of a law, due to the difficulties of achieving synchronized, atomic law update in distributed systems.

There are several other approaches in the literature concerned with architectural issues of run-time specification change—[15, 25, 40] are but a few examples. These issues—various distributed configurations for computing the normative relations current at each time—are beyond the scope of this article, and will be considered in future work.

Bou *et al*. [10, 11] have presented a mechanism for the run-time modification of the norms of an ‘electronic institution’ [26–30, 36, 70]. These researchers have proposed a ‘normative transition function’ that maps a set of norms (and goals) into a new set of norms: changing a norm requires changing its parameters, or its effect, or both. The ‘institutional agents’, representing the institution, are observing the members’ interactions in order to learn the normative transition function, so that they (the institutional agents) will directly enact the norms enabling the achievement of the ‘institutional goals’ in a given scenario. Unlike Bou *et al*., we do not necessarily rely on designated agents to modify norms. We presented an infrastructure with which any agent may (attempt to) adapt the system specification. This does not exclude the possibility, however, that, in some applications, designated agents are given, under certain circumstances, the institutional power to directly modify the system specification.

Identifying *when* (to propose) to change a system specification during the system execution, as done in the work of Bou and colleagues with respect to the 'institutional goals', is a fundamental requirement for adaptive MAS. Addressing this requirement, however, is out of the scope of this article.

Boella *et al.* [9] have developed a formal framework for representing norm change—see [13] for a recent survey on formal models of norm change. The framework of norm change presented in [9] is produced by replacing the propositional formulas of the Alchourrón, Gärdenfors, Makinson (AGM) framework of theory change [2] with *pairs* of propositional formulas—the latter representing norms—and adopting several principles from input/output logic [59]. The resulting framework includes a set of postulates defining norm change operations, such as norm contraction.

Governatori *et al.* [43–45] have presented variants of a Temporal Defeasible Logic [41, 42] to reason about different aspects of norm change. These researchers have represented meta norms describing norm modifications by referring to a variety of possible time-lines through which the elements of a norm-governed system, and the conclusions that follow from them, can or cannot persist over time. Governatori *et al.* [43–45] have formalized norm change operations according to which norms are removed with all their effects, as well as operations according to which norms are removed but all or some of their effects propagate if obtained before the modification.

$C+$, along with proposed extensions [21] of this language, allows for the formalization of relatively complex norm change operations. We expect that the expressiveness of $C+$ is adequate for representing norm change operations for a wide range of software MAS. The infrastructure presented in this article, however, may be formalized using other languages. We chose $C+$ because it enables the formal representation of the (direct and indirect) effects of actions and default persistence (inertia) of facts, has a transition system semantics and thus there is a link to a wide range of other formalisms and tools based on transition systems (later in this section we describe a way to exploit this link by combining $C+$ with standard model checkers), and has direct routes to implementation.

Rubino and Sartor [71] have presented a taxonomy of 'source norms', that is, norms empowering the members of a system to modify a set of other norms. The so-called 'agreement-based source norms' may be viewed as the norms of a meta protocol in our infrastructure, in the sense that they allow for norm change based on the deliberation of a group of agents. Rubino and Sartor employ the logic of the PRATOR system for defeasible argumentation [69] to express source norms. The formalization of the ideas presented in [71], however, is restricted; consequently, this line of work cannot be used yet to support dynamic MAS.

Run-time specification change has long been studied in the field of argumentation. Loui [57], for example, identified the need to allow for the run-time modification of argumentation protocol rules by means of meta argumentation. Vreeswijk [86] also investigated forms of meta argumentation. The starting point for this work was two basic observations. First, that there are different protocols appropriate for different contexts (for example, quick and shallow reasoning when time is a constraint; restricted number of counter-arguments when there are many rules and cases; etc). Secondly, that 'points of order', by which a participant may steer the protocol to a desired direction, are standard practice in dispute resolution meetings. Vreeswijk then defined a formal protocol for disputes in which points of order can be raised to allow (partial) protocol changes to be debated. A successful 'defence' meant that the parties in the dispute agreed to adopt a change in the protocol, and the rules of dispute were correspondingly changed.

As already mentioned, our work is motivated by Brewka's dynamic argument systems [12]. Like Vreeswijk's work, these are argument systems in which the protagonists of a disputation may start a meta level debate, that is, the rules of order become the current point of discussion, with the intention of altering these rules. Unlike Vreeswijk's work, there may be more than one meta argumentation level.

A key difference between our work and Brewka's approach, and more generally, a key difference between our work and related research, including all approaches discussed in this section, is that we formalize the transition protocol leading from an object protocol to a meta protocol. More precisely, we distinguish between successful and unsuccessful attempts to initiate a meta protocol (exercising the institutional power to propose/second/object to a specification change versus proposing/seconding/objecting to a specification change without the necessary power), evaluate proposals for specification change by modelling a specification as a metric space, and by taking into consideration the effects of accepting a proposal on system utility, constrain the enactment of proposals that do not meet the evaluation criteria, and formalize procedures for role-assignment in a meta level.

In this section, we focused on the facilities offered by related approaches with respect to system specification *change*. A comparison of our work with commitment protocols (including [18, 32–34, 84]), LGI, electronic institutions, Brewka's argument systems, as well as other approaches for specifying norm-governed systems, from the viewpoint of *static* specifications, may be found in [5, 6].

We have been concerned with a particular aspect of 'organized adaptation' [85]: the run-time modification of the 'rules of the game' of norm-governed systems. Clearly there are other aspects of (organized) adaptation such as the run-time alteration of the (trading and other) relationships between agents, the assignment of roles to agents, and the goals of a system. [19, 22, 23, 31, 48–50, 54, 60, 76, 81] are but a few examples of studies of adaptive systems.

We employed CCALC for the provision of run-time services—for instance, to compute the system state current at each time. CCALC, however, can become inefficient when considering action descriptions defining large transition systems. (CCALC is not the only means by which $C+$ action descriptions can be executed. In [37, 38] it is shown how $C+$ action descriptions can be translated into the formalism of (extended) logic programs.) We have also used versions of the Event Calculus (EC) [55] to specify and execute dynamic specifications [3]. EC is a simple and flexible formalism that is efficiently implemented for narrative assimilation. Our Prolog EC implementation, however, does not offer facilities for proving properties of a specification or planning. More importantly, we also lose the explicit transition system semantics which we see as a very important advantage of the $C+$ formulation. A comparison of EC and $C+$ with respect to executable MAS specifications can be found in [7].

A direction for further work is to employ a single formalism for efficient execution (narrative assimilation and proving properties) of (dynamic) MAS specifications. Some first steps are reported by Craven [20] who investigates methods for efficient EC-like query evaluation for (a subset of) the $C+$ language, and for integrating action descriptions in this language with standard model checking systems (specifically NuSMV [17]). Norm-governed system properties expressed in temporal logics such as Computation Tree Logic can then be verified by means of standard model checking techniques on a transition system defined using the $C+$ language.

Acknowledgements

This article is a significantly updated and extended version of [3, 4]. We would like to thank the reviewers and participants of the Eighth International Conference on Autonomous Agents and

Multi-Agent Systems, the Twelfth International Conference on Artificial Intelligence and Law, as well as the reviewers of the Journal of Logic and Computation, who gave us useful feedback. We would also like to thank Jeremy Pitt and Lloyd Kamara for their comments on various drafts of the article.

References

[1] V. Akman, S. Erdogan, J. Lee, V. Lifschitz, and H. Turner. Representing the zoo world and the traffic world in the language of the Causal Calculator. *Artificial Intelligence*, **153**, 105–140, 2004.

[2] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, **50**, 510–530, 1985.

[3] A. Artikis. Dynamic protocols for open agent systems. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 97–104. ACM, 2009.

[4] A. Artikis. Formalising dynamic protocols for open agent systems. In *Proceedings of International Conference on Artificial Intelligence & Law (ICAIL)*, pp. 68–77. ACM, 2009.

[5] A. Artikis and M. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, **18**, 31–65, 2010.

[6] A. Artikis, M. Sergot, and J. Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, **171**, 776–804, 2007.

[7] A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, **10**, 2009.

[8] J. Bing. Managing copyright in a digital environment. In *The Impact of Electronic Publishing on the Academic Community*, I. Butterworth, ed., pp. 52–62. Portland Press, 1998.

[9] G. Boella, G. Pigozzi, and L. van der Torre. Normative framework for normative system change. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 169–176. ACM Press, 2009.

[10] E. Bou, M. López-Sánchez, and J. Rodriguez-Aguilar. Towards self-configuration in autonomic electronic institutions. In *Proceedings of Workshop on Coordination, Organization, Institutions and Norms in agent systems*, Vol. 4386 of *Lecture Notes in Computer Science*, pp. 220–235. Springer, 2007.

[11] E. Bou, M. López-Sánchez, and J. Rodriguez-Aguilar. Using case-based reasoning in autonomic electronic institutions. In *Proceedings of Workshop on Coordination, Organization, Institutions and Norms in agent systems*, Vol. 4870 of *Lecture Notes in Computer Science*, pp. 125–138. Springer, 2008.

[12] G. Brewka. Dynamic argument systems: a formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, **11**, 257–282, 2001.

[13] J. Broersen. Issues in designing logical models for norm change. In *Proceedings of Internation Workshop in Oprganised Adaptation on Multi-Agent Systems (OAMAS)*, G. Vouros, A. Artikis, K. Stathis, and J. Pitt, eds, Vol. 5368 of *Lecture Notes in Computer Science*, pp. 1–17. Springer, 2009.

[14] V. Bryant. *Metric Spaces*. Cambridge University Press, 1985.

[15] R. Chadha, H. Cheng, Y.-H. Cheng, J. Chiang, A. Ghetie, G. Levin, and H. Tanna. Policy-based mobile ad hoc network management. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pp. 35–44. IEEE Computer Society, 2004.

[16] A. Chopra and M. Singh. Contextualizing commitment protocols. In *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1345–1352. ACM, 2006.

[17] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: an opensource tool for symbolic model checking. In *Proceedings of International Conference on Computer-Aided Verification (CAV 2002), Copenhagen, July 2002*, Vol. 2404 of *Lecture Notes in Computer Science*. Springer, 2002. See `http://nusmv.irst.itc.it`.

[18] M. Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of Workshop on Agent Languages and Communication Policies*, pp. 21–29, 2000.

[19] A. Rocha Costa and G. Pereira Dimuro. A minimal dynamical MAS organization model. In *Multi-Agent Systems—Semantics and Dynamics of Organisational Models*. V. Dignum, ed., pp. 419–445. IGI Global, 2009.

[20] R. Craven. *Execution Mechanisms for the Action Language $\mathcal{C}+$*. PhD Thesis, University of London, September 2006.

[21] R. Craven and M. Sergot. Distant causation in C+. *Studia Logica*, **79**, 73–96, 2005.

[22] S. DeLoach. OMACS: a framework for adaptive, complex systems. In *Multi-Agent Systems: Semantics and Dynamics of Organisational Models*. V. Dignum, ed., pp. 76–104. IGI Global, 2009.

[23] S. DeLoach, W. Oyenan, and E. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, **16**, 13–56, 2008.

[24] H.-P. Dommel and J. J. Garcia-Luna-Aceves. Floor control for multimedia conferencing and collaboration. *Multimedia Systems*, **5**, 23–38, 1997.

[25] N. Dulay, E. Lupu, M. Sloman, and N. Damianou. A policy deployment model for the Ponder language. In *Proceedings of International Symposium on Integrated Network Management*, pp. 14–18. IEEE/IFIP, 2001.

[26] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, C. Castelfranchi and L. Johnson, eds, pp. 1045–1052. ACM Press, 2002.

[27] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In *Intelligent Agents VIII: Agent Theories, Architectures, and Languages*, J.-J. Meyer and M. Tambe, eds, Vol. 2333 of *Lecture Notes in Artifical Intelligence*, pp. 348–366. Springer, 2002.

[28] M. Esteva, J. Rodriguez-Aguilar, J. Arcos, C. Sierra, and P. Garcia. Institutionalising open multi-agent systems. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*, E. Durfee, ed., pp. 381–382. IEEE Press, 2000.

[29] M. Esteva, J. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specifications of electronic institutions. In *Agent Mediated Electronic Commerce*, F. Dignum and C. Sierra, eds, Vol. 1991 of *Lecture Notes in Artifical Intelligence*, pp. 126–147. Springer, 2001.

[30] M. Esteva, J. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. Verifying norm consistency in electronic institutions. In *Proceedings of the AAAI-04 Workshop on Agent Organizations: Theory and Practice*, pp. 8–14, 2004.

[31] C. B. Excelente-Toledo and N.R. Jennings. The dynamic selection of coordination mechanisms. *Autonomous Agents and Multi-Agent Systems*, **9**, 55–85, 2004.

[32] N. Fornara and M. Colombetti. *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Formal specification of artificial institutions using the event calculus. IGI Global, 2009.

[33] N. Fornara, F. Viganò, and M. Colombetti. Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, **14**, 121–142, 2007.

[34] N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti. Artificial institutions: a model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, **16**, 89–105, 2008.

[35] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid—enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, **15**, 200–222, 2001.

[36] A. García-Camino, P. Noriega, and J. Rodríguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 667–673. ACM Press, 2005.

[37] M. Gebser, T. Grote, and T. Schaub. Coala: a compiler from action languages to ASP. In *European Conference in Logics in Artificial Intelligence (JELIA)*, pp. 360–364, 2010.

[38] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, **153**, 49–104, 2004.

[39] E. Giunchiglia, J. Lee, V. Lifschitz, and H. Turner. Causal laws and multi-valued fluents. In *Proceedings of Workshop on Nonmonotonic Reasoning, Action and Change (NRAC)*, 2001.

[40] S. Godic and T. Moses. Oasis extensible access control markup language (xacml), vesrion 2.0. `http://www.oasis-open.org/commitees/xacml/index.shtml`, March 2005.

[41] G. Governatori, V. Padmanabhab, and A. Rotolo. Rule-based agents in temporalised defeasible logic. In *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Vol. 4099 of *Lecture Notes in Computer Science*, pp. 31–40. Springer, 2006.

[42] G. Governatori, M. Palmirani, R. Riveret, A. Rotolo, and G. Sartor. Norm modifications in defeasible logic. In *Proceedings of Conference on Legal Knowledge and Information Systems (JURIX)*, pp. 13–22. IOS Press, 2005.

[43] G. Governatori and A. Rotolo. Changing legal systems: abrogation and annulment. Part I: revision of defeasible theories. In *Proceedings of Conference on Deontic Logic in Computer Science (DEON)*, R. van der Meyden and L. van der Torre, eds, Vol. 5076 of *Lecture Notes in Computer Science*, pp. 3–18. Springer, 2008.

[44] G. Governatori and A. Rotolo. Changing legal systems: abrogation and annulment. Part II: temporalised defeasible logic. In *Proceedings of Workshop on Normative Multiagent Systems (NORMAS)*, G. Boella, G. Pigozzi, M. Singh, and H. Verhagen, eds, pp. 112–127, 2008.

[45] G. Governatori, A. Rotolo, R. Riveret, M. Palmirani, and G. Sartor. Variants of temporal defeasible logics for modelling norm modifications. In *Proccedings of International Conference on Artificial Intelligence & Law (ICAIL)*. ACM, 2007.

[46] M. Hardwick and R. Bolton. The industrial virtual enterprise. *Communications of the ACM*, **40**, 59–60, 1997.

[47] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, **47**, 79–106, 1991.

[48] M. Hoogendoorn. Adaptation of organizational models for multi-agent systems based on max flow networks. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1321–1326, 2007.

[49] M. Hoogendoorn, C. Jonker, M. Schut, and J. Treur. Modeling centralized organization of organizational change. *Computational & Mathematical Organization Theory*, **13**, 147–184, 2007.

[50] J. F. Hübner, J. S. Sichman, and O. Boissier. Using the MOISE+ for a cooperative framework of MAS reorganization. In *Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA)*, Vol. 3171 of *Lecture Notes in Artifical Intelligence*, pp. 506–515. Springer, 2004.

[51] L. L. Jilani, J. Desharnais, and A. Mili. Defining and applying measures of distance between specifications. *IEEE Transactions on Software Engineering*, **27**, 673–703, 2001.

[52] A. Jones and M. Sergot. On the characterisation of law and computer systems: the normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pp. 275–307. J. Wiley and Sons, 1993.

[53] A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, **4**, 429–445, 1996.

[54] R Kota, N. Gibbins, and N. Jennings. Self-organising agent organizations. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pp. 797–804. IFAAMAS, ACM, 2009.

[55] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, **4**, 67–96, 1986.

[56] J. Lee, V. Lifschitz, and H. Turner. A representation of the zoo world in the language of the Causal Calculator. In *Proceedings of Symposium on Formalizations of Commonsense Knowledge*, 2001.

[57] R. Loui. Process and policy: Resource-bounded non-demonstrative argument. *Technical report*, Washington University, Department of Computer Science, 1992.

[58] D. Makinson. On the formal representation of rights relations. *Journal of Philosophical Logic*, **15**, 403–425, 1986.

[59] D. Makinson and L. van der Torre. Input-output logics. *Journal of Philosophical Logic*, **29**, 383–408, 2000.

[60] C. Martin and K. S. Barber. Adaptive decision-making frameworks for dynamic multi-agent organizational change. *Autonomous Agents and Multi-Agent Systems*, **13**, 391–428, 2006.

[61] N. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, **17**, 183–195, 1991.

[62] N. Minsky. On conditions for self-healing in distributed software systems. In *International Workshop on Active Middleware Services*. IEEE Computer Society, 2003.

[63] N. Minsky. *Law-Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction and a Reference Manual)*, 2005. Retrieved 24 October 2008, from `http://www.moses.rutgers.edu/documentation/manual.pdf`.

[64] N. Minsky. Decentralised regulation of distributed systems: Beyond access control. Submitted for publication. Retrieved 24 October 2008, from `http://www.cs.rutgers.edu/~minsky/papers/IC.pdf`, 2008.

[65] N. Minsky and T. Murata. On manageability and robustness of open multi-agent systems. In *Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications*, Vol. 2940 of *Lecture Notes in Computer Science*, pp. 189–206. Springer, 2004.

[66] N. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **9**, 273–305, 2000.

[67] J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in multi-agent systems. *Computer Journal*, **49**, 156–170, 2006.

[68] H. Prakken and T. Gordon. Rules of order for electronic group decision making – a formalization methodology. In *Collaboration between Human and Artificial Societies*, J. Padget, ed., Vol. 1624 of *Lecture Notes in Computer Science*, pp. 246–263. Springer, 1999.

[69] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, **7**, 25–75, 1997.

[70] J. Rodriguez-Aguilar and C. Sierra. Enabling open agent institutions. In *Socially Intelligent Agents: Creating relationships with computers and robots*, K. Dautenhahn, A. Bond, L. Canamero, and B. Edmonds, eds, pp. 259–266. Kluwer Academic Publishers, 2002.

[71] R. Rubino and G. Sartor. Source norms and self-regulated institutions. In *Computable Models of the Law, Languages, Dialogues, Games, Ontologies*, P. Casanovas, G. Sartor, N. Casellas, and R. Rubino, eds, Vol. 4884 of *Lecture Notes in Computer Science*, Springer, 2008.

[72] J. Searle. What is a speech act? In *Philosophy of Language*, 3rd edn. A. Martinich, ed., pp. 130–140. Oxford University Press, 1996.

[73] C. Serban and N. Minsky. In vivo evolution of policies that govern a distributed system. In *International Symposium on Policies for Distributed Systems and Networks*, pp. 134–141. IEEE, 2009.

[74] M. Sergot. $(\mathcal{C}+)^{++}$: An action language for modelling norms and institutions. *Technical Report 2004/8*, Department of Computing, Imperial College London, 2004.

[75] M. Sergot. Modelling unreliable and untrustworthy agent behaviour. In *Proceedings of Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS)*, B. Dunin-Keplicz, A. Jankowski, A. Skowron, and M. Szczuka, eds, Advances in Soft Computing, pp. 161–178. Springer-Verlag, 2004.

[76] Y. Shoham and M. Tennenholtz. On the emergence of social conventions: modeling, analysis and simulations. *Artificial Intelligence*, **94**, 139–166, 1997.

[77] M. Singh. Agent communication languages: rethinking the principles. *IEEE Computer*, **31**, 40–47, 1998.

[78] M. Singh. An ontology for commitments in multiagent systems: towards a unification of normative concepts. *Artificial Intelligence and Law*, **7**, 97–113, 1999.

[79] M. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, F. Dignum and M. Greaves, eds, Vol. 1916 of *Lecture Notes in Computer Science*, pp. 31–45. Springer, 2000.

[80] M. Sirbu. Credits and debits on the Internet. *IEEE Spectrum*, **34**, 23–29, 1997.

[81] G. Tesauro, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White. A multi-agent systems approach to autonomic computing. In *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 464–471. ACM, 2004.

[82] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, **10**, 287–299, 2007.

[83] M. Venkatraman and M. Singh. Verifying compliance with commitment protocols. *Journal of Autonomous Agents and Multi-Agent Systems*, **2**, 217–236, 1999.

[84] F. Viganò and M. Colombetti. Symbolic model checking of institutions. In *Proceedings of Conference on Electronic commerce*, pp. 35–44. ACM, 2007.

[85] G. Vouros, A. Artikis, K. Stathis, and J. Pitt, eds. *Organized Adaption in Multi-Agent Systems*, Vol. 5368 of *Lecture Notes in Artifical Intelligence*. Springer, 2008.

[86] G. Vreeswijk. Representation of formal dispute with a standing order. *Artificial Intelligence and Law*, **8**, 205–231, 2000.

[87] W. Zhang, C. Serban, and N. Minsky. Establishing global properties of multi-agent systems via local laws. In *Environments for Multiagent Systems III*, D. Weyns, ed., Vol. 4389 of *Lecture Notes in Artifical Intelligence*. pp. 170–183. Springer, 2007.

Received 23 July 2010