Voting in Multi-Agent Systems

Jeremy Pitt[1], Lloyd Kamara[1], Marek Sergot[2], Alexander Artikis[2]
[1]Intelligent Systems & Networks Group
Department of Electrical & Electronic Engineering
[2]Computational Logic Group, Department of Computing
Imperial College London
London SW7 2BT, UK
{j.pitt, l.kamara, m.sergot, a.artikis}@imperial.ac.uk

Abstract

Voting is an essential element of mechanism design for multi-agent systems, and by extension applications built on such systems, which includes ad hoc networks, virtual organizations, and decision support tools implementing online deliberative assemblies. Much attention has been given both to designing the process so that it is resistant to manipulation by strategic voting, and so that an automated system can follow rules of order as developed for the conduct of formal meetings. In this paper, we define, characterise, formally specify and animate a general voting protocol. In particular we show how the requirements established by the characterisation are captured by the specification, and are exhibited by the animation. The importance of these requirements is in ensuring robustness by respecting the way in which votes are cast and the outcome is declared, especially as this issue relates to the 2004 ACM Statement on E-Voting.

1 Introduction

Voting is the *sine qua non* of computer-based electronic voting (e-Voting) systems for public elections, an essential element of mechanism design for multi-agent systems, and a crucial aspect of decision making in CSCW tools implementing online deliberative assemblies.

Despite the evident potential advantages of voting for coordination (in multi-agent systems), democratic public elections, and e-Government, a number of risks remain. Some of these are endemic to the nature of voting itself: for example, Arrow's Theorem [2], which states that there is no 'ideal' election method, indicates that all methods of winner determination can be manipulated by strategic voting. Other risks pertain to the fact that these are computer-based systems, and may have been poorly designed and inadequately tested, and may have security holes which can be exploited. Yet other risks arise from the fact that in distributed systems of 'autonomous' components there is a possibility that, during the process of taking a vote, a component may, for whatever reason, misbehave (its actions deviate from the 'ideal').

Therefore, we require an electronic voting system to be robust. However, even leaving aside basic security considerations (such as encryption, secure message transport protocols, etc.), which are addressed by commercial applications (e.g. [14]), robustness has several different aspects. One aspect of robustness is that the voting system achieves the 'optimal' outcome, i.e. the outcome reflects the 'popular vote' and is resistant to manipulation by strategic voting. A second aspect of robustness is flexibility (in the sense that flexibility provides freedom to manoeuvre and robustness is the ability to accommodate such freedom without breaking). Rules in human organization have 'degrees of freedom' in terms of being broken, and automated systems should support this freedom, not compel users to comply. A third aspect of robustness is found in ensuring that whoever counts the votes and declares the results does so in accordance with the way that the votes are cast and the rules which decide who has won.

Voting, like auctions, is commonplace in multi-agent systems and much attention has been given to this particular aspect of mechanism design (the process of designing a choice mechanism that translates expressed preferences into a decision). In particular, designing the process so that it is resistant to manipulation by strategic voting has been studied, with many results (e.g. [9]). This work has contributed to achieving robustness with respect to the optimal outcome. In addition, there has been significant work in designing a decision-support system (called ZENO) for online deliberative assemblies [21, 20] using Robert's Rules of Order, the standard handbook for conducting business in deliberative assemblies [22]. In particular, a design maxim for the ZENO system could have been "anything goes unless someone objects", so this work has, inter alia, contributed to achieving robustness with respect to flexibility.

This paper is concerned with the third aspect of robustness: namely the requirement for robustness to respect the way in which votes are cast and the outcome is declared. This is especially important in light of the 2004 ACM Statement on E-Voting [1], which includes the requirements that:

> [computer-based electronic] voting systems should enable each voter . . . to verify that his or her vote has been accurately cast and to serve as an independent check on the result produced and stored by the system.

We address this issue from the perspective of the *open agent society*, where we seek to model the society in terms of powers, permissions, obligations, and sanctions [4, 6]. In particular, we aim to characterise a notion of enfranchisement with respect to a right (to vote) and an entitlement (to a 'fair' result), and capture this characterisation in a formal specification. This specification can then be animated to examine its properties, and implemented for use in applications in such a way as to (ideally) satisfy the ACM's requirements.

Accordingly, the rest of this paper is structured as follows. Section 2 informally introduces the voting protocol, and characterises a notion of enfranchisement. Section 3 presents a formal specification of the voting protocol, while Section 4 considers the use of powers, permissions, obligations, etc., to formally capture this notion of enfranchisement. Section 5 briefly considers some additional issues, Section 6 discusses the animation and implementation of the formal specification, and Section 7 discusses applications, including virtual organizations and the development of a system for online deliberative assemblies, in which the voting protocol is just one member of a suite of protocols which collectively implement rules of order. We briefly consider some related research in Section 8, and Section 9 concludes the paper, and in particular we comment on this work with respect to the ACM Statement on E-Voting.

2 A Voting Protocol

This section gives a brief, informal specification of a voting procedure, based on Robert's Rules of Order (Newly Revised) [22], henceforth RONR, making clear any assumptions or alterations needed for a computational setting. We begin by defining the procedure for taking a vote, defining some subsets of the members (agents) of the network according to what we provisionally refer to as their *roles*. We then consider what it means for an agent occupying the role of a 'voter' to be 'enfranchised'.

2.1 Voting Procedure

According to RONR, an appropriate procedure for conducting a vote in a committee meeting is as follows:

- the committee sits and the *chair* opens the meeting;
- a member proposes a *motion*;
- another member seconds the motion;
- the members debate the motion;
- the chair *calls* for those in favour to cast their vote;
- the chair calls for those against the motion to cast their vote;
- the motion is *carried*, or not, according to the standing rules of the committee.

In the context of a deliberative assembly, we will associate a committee with the assembly itself, and the committee members with the assembly members. A meeting of the members at which decisions are to be made will be termed as a session: there may be several sessions during the lifetime of an assembly. As the system is online, the access point is through an agent and the term agent will be used for 'member' in the subsequent analysis.

2.2 Voting Roles

This informal description gives the basic steps in the voting procedure. To determine who is empowered (in a sense that is made explicit later) to enact (or perform) each step, we identify a number of subsets of the set of agents. How set membership is determined we do not address precisely here; although we note it can be done in a variety of ways. For example, it could be done by default (being a representative in the assembly automatically classifies into one or other set); by qualification (being in possession of some certificate or capability ensures classification), or by assignment (i.e. some other protocol is used to allocate members to a set).

The subsets of the set of agents according to the *roles* that the agents can occupy, are:

voter, i.e. those agents who are enfranchised (have the right to vote);
proposer, those agents who are empowered to propose motions;
seconder, those agents who are empowered to second motions;
chair, those agents who are qualified to conduct the procedure; one of whom, at any time, will be designated to be the actual chair, and thereby empowered to conduct the procedure;
monitor, those members who are to be informed of the actions of others, in particular how they cast their votes (for or against), and the outcomes of votes (passed or not).

We assume a predicate *qualifies* which holds of an agent belonging to the set. We then assume (and will not address further here) a role assignment protocol which determines, for those agents that qualify, which role(s) they occupy. This determines who occupies the roles of voter, proposer, chair and monitor. It is, however, the act of proposing that determines the set of seconders (i.e., of those agents qualified to act as seconders, which of them occupy the role of seconder, whereby they are empowered to second motions).

2.3 Enfranchisement

A crucial element of voting, i.e. occupying the role of a voter, is being enfranchised: in other words, each agent who is a *voter* is enfranchised and thus has the “right to vote”. However, it appears that encapsulated in this right there is also an element of entitlement: in particular the voters are entitled to a ‘fair’ outcome which respects the way they have cast their votes and the agreed rules for determining the winner. Therefore, a formal specification of a voting protocol should properly represent such a notion of enfranchisement for encoding in an electronic voting protocol.

We characterise this notion of enfranchisement as being composed of two elements:

- having the right to vote; and
- having an entitlement associated with that right.

In this instance, having the right to vote is decomposed into three aspects:

- having the power (being empowered) to vote;
- denying anyone else the power to object to 'appropriate' exercise of this power; and
- subjecting inappropriate removal of this power to sanction;

while the associated entitlements are also decomposed into three aspects:

- being entitled to access the 'voting machinery';
- being entitled to have the vote counted correctly; and
- being entitled to a 'fair' outcome (i.e. the result is declared according to the way the votes were cast with respect to the standing rules of the committee).

Thus enfranchisement is, in the context of this work, composed of two elements: a right and an entitlement. The right is characterised by the power to perform the action, and the fact that nobody is allowed to prevent the agent from acting in this way, either by objecting (see later) or by removing the power without repercussion. However, there also appears to be a second element of enfranchisement, that of entitlement. In our analysis, being enfranchised is (partially) analogous to being entitled to a resource: when that is the case, a request to access the resource must count as such, there is an obligation on the access controller to grant access to the resource, and the resource should be properly usable (cf. [3]). From this perspective, having a vote should be like having (being granted) access to an abstract voting machine, which correctly records the vote, and produces the 'correct' result (correct according to whatever rules have been agreed). Therefore there appears to be three aspects to this entitlement. The first is being entitled to access the 'voting machinery': this is really concerned with the physical dimension of voting, and need not concern us further here. The second aspect of entitlement is implicit in the 2004 ACM Statement on E-Voting, and the requirement that "the vote has been accurately cast". The third aspect of entitlement is characterised by the fact that 'being entitled to' [something] means in part having the (institutional) power (i.e. performing the action counts as a valid action in the context) *and* a corresponding obligation on someone else (cf. [11]). That 'someone else' will also have to occupy a role as identified in the protocol: by analogy this will be the resource controller, and so in the case of the voting protocol it will be whoever occupies the role of *chair*.

It is these concepts that we seek to capture through the formalization presented in the next two sections.

3 Formal Specification

In this section, we give an indicative formal specification (via a logical axiomatisation in the Event Calculus [17]) of the informal description of the voting protocol. The specification in this section covers the fluents and actions, the (institutional) powers, changing the status of motions, roles and role assignment, and various aspects of voting. The following section considers the use of power, permission and obligation to capture the concept of enfranchisement considered above. We begin, though, with a brief overview of the analytic basis of the formal specification, namely the Event Calculus and the concept of Institutional Power [16].

3.1 Analytic Base

The Event Calculus (EC) is a formalism from Artificial Intelligence intended to reason about action and change in non-monotonic systems [17]. There are several variants of the calculus: the version used here is that used in [3]. An Event Calculus specification consists of a domain-independent part and an application specific part. The domain independent part includes axioms for determining what *holds* at a given time (or in a given state). What holds are fluents, which can be either *true* or *false* if the fluent is boolean, or some value from a specific range in the case of many-valued fluents. The application-specific part specifies axioms for determining: what holds at the initial time, the values of particular fluents that are said to be *initiated* by a specific action at any time, and the state constraints which determine what holds (or does not hold) at any time. Note that the animation (Section 6) is coded in Prolog, and so the axiomatisation reflects that orientation.

Every action of the voting protocol (see Table 1 blow) is a communicative one, that is, it is being communicated from one agent to another. We have chosen to assume that every protocol action is physically possible; for example, every agent is physically capable of performing any communicative act.

However, in order to express the *effects* of performing the protocol actions we need to specify the institutional powers of the participating agents. The term institutional (or 'institutionalised') power refers to the characteristic feature of organisations/institutions — legal, formal, or informal — 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 [24], 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.

According to the account given by Jones and Sergot [16], institutional power can be seen as a special case of a more general phenomenon whereby an action, or a state of affairs, A — because of the rules and conventions of an institution — counts, in that institution, as an action or state of affairs B. For example, in a committee meeting of some institution, the chair's performance of the speech act 'the motion m is carried' counts as, as far as that institution is concerned, a way of establishing that motion m is carried. One then says that the chair has the power, or is empowered by the institution (and within the remit of that committee), to establish that motion m is carried. The same action performed by an agent without this power has no effect on the status of m (it may have other effects, however, such as the removal of the agent from the meeting room). Depending on the content of motion m, other agents may acquire further roles or powers within the institution.

3.2 Actions

The set of possible actions that may be performed by agents can be identified from the informal description of the protocol and are listed in Table 1. For the committee to sit there have to be actions to open and close a session; we need actions to propose and second motions; we need actions to open and close a ballot on a motion; and above all we need actions to vote for/against a motion (or abstain), and to declare a motion carried or not. It should be intuitive from the identified roles and an informal understanding of the actions that only agents occupying certain roles may meaningfully perform certain actions. This has significance for establishing the status of motions, which are institutional facts in the domain (cf. [23, 16]).

Table 1: Actions in the Voting Protocol

Action	**Textual Description**
$open\_session(Ag, S)$	agent Ag opens session S
$close\_session(Ag, S)$	agent Ag closes session S
$propose(Ag, M)$	agent Ag proposes motion M
$second(Ag, M)$	agent Ag seconds motion M
$open\_ballot(Ag, M)$	agent Ag opens the voting on motion M
$close\_ballot(Ag, M)$	agent Ag closes the voting on motion M
$vote(Ag, M, aye)$	agent Ag votes *for* motion M
$vote(Ag, M, nay)$	agent Ag votes *against* motion M
$revoke(Ag, M)$	agent Ag revokes its vote on motion M
$abstain(Ag, M)$	agent Ag abstains on motion M
$declare(Ag, M, carried)$	agent Ag declares that motion M has carried
$declare(Ag, M, not\_carried)$	agent Ag declares that motion M has not carried

3.3 State Information

At any point in the voting protocol, various elements of state information need to be maintained. In particular, these concern the status of motions and information about votes and voters.

The status of a motion follows a nearly linear sequence as indicated by the actions in Table 1. Initially, the status of all (putative) motions is null. After an agent proposes a motion its status is *proposed*; after it is seconded, *seconded*; after the chair opens a ballot on the motion, it is *voting*; after a certain time, during which votes are cast for or against the motion, when the chair closes the ballot its status is *voted*; and after the votes are counted, then the motion is declared either *carried* or *not_carried*, according to the standing rules of the committee. Note that the standing rules can be anything 'reasonable' (majority of members, majority of those voting, etc.), but the rules themselves will have been negotiated during the formation of the committee. However, they can also be the subject of motions themselves (i.e. agents can propose to change the standing rules).

In addition to the status of a motion, we need to represent the following items of information: the votes cast for/against the motion, which way each voter voted on a motion, the roles of each agent, the normative positions of each agent, and any sanctions held against each agent.

This information will be recorded as fluents (predicates whose values can change over time). The fluents required for the voting protocol are thus summarised in Table 2.

Table 2: Fluents in the Voting Protocol

Fluent	**Range**	**Textual Description**
$status(M)$	$\{$ *pending*	initially status of any motion is pending
	proposed	motion M was proposed
	seconded	motion M has been seconded
	$voting(T)$	voting on motion M is taking place from time T
	voted	voting on motion M has been closed
	resolved $\}$	decision reached on motion M (carried or not carried)
$votes(M)$	$N \times N$	2-tuple counting the votes for/against motion M
$voted(Ag, M)$	$\{nil,$	agent Ag has not voted (*nil*)
	$aye, nay, abs\}$	or has voted for/against/abstained on motion M
$qualifies(Ag, R)$	boolean	agent Ag is qualified for the role of R
$role\_of(Ag, R)$	boolean	agent Ag occupies the role of R
$sitting(S)$	boolean	the assembly S is in session (sitting) or not
$\mathbf{pow}(Ag, Act)$	boolean	agent Ag is empowered to perform action Act
$\mathbf{per}(Ag, Act)$	boolean	agent Ag is permitted to perform Act
$\mathbf{obl}(Ag, Act)$	boolean	agent Ag is obliged to perform Act
$sanction(Ag)$	N^*	zero or more codes identifying sanctions on agent Ag

This table shows, *inter alia*, that a session S can be open (sitting). The status of a motion is pending (initially), then proposed (by some agent A at some time T), seconded (by some other agent B at a later time T'), voting (the chair agent C has called for votes at some even later time T''), voted (voting has closed), or resolved (a decision has been reached). The *votes* fluent is used to count the votes for/against a motion; the *voted* fluent records how each agent Ag voted on motion M (has not voted, for, against, abstained). If a vote is carried, then it is added to the list of resolutions from this session. R ranges over the values *proposer*, *seconder*, *voter*, *monitor* and *chair*: for each agent and each value, an agent *qualifies* for or occupies the *role_of* that value (or does not, since these are boolean-valued fluents). There are three more boolean valued fluents for the normative positions of an agent (its powers, permissions, and obligations), and the sanctions are a list of integers which will be used as codes for 'inappropriate' behaviour.

3.4 Institutional Powers

The value of a fluent changes as a consequence of an empowered agent performing a designated act. It will be those agents that occupy particular roles that are empowered to perform such acts (cf. [16]). Generally, agents only 'possess' these powers when particular fluents have a certain value, that value being the status of a motion or a session. In other words, status plus role determines power. The following axioms illustrate this general principle for the *open_ballot* and *vote* actions, and there are similar axioms for the other (relevant) actions listed in Table 1.

$$\begin{aligned}
&\mathbf{pow}(C, \mathit{open\_ballot}(C, M)) = \mathit{true} \;\; \mathsf{holdsat} \;\; T \leftarrow \\
&\quad \mathit{status}(M) = \mathit{seconded} \;\; \mathsf{holdsat} \;\; T \;\; \wedge \\
&\quad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true} \;\; \mathsf{holdsat} \;\; T \\
&\mathbf{pow}(V, \mathit{vote}(V, M, \_)) = \mathit{true} \;\; \mathsf{holdsat} \;\; T \leftarrow \\
&\quad \mathit{status}(M) = \mathit{voting}(T') \;\; \mathsf{holdsat} \;\; T \;\; \wedge \\
&\quad \mathit{role\_of}(V, \mathit{voter}) = \mathit{true} \;\; \mathsf{holdsat} \;\; T \;\; \wedge \\
&\quad \mathit{voted}(V, M, \_) = \mathit{nil} \;\; \mathsf{holdsat} \;\; T
\end{aligned}$$

The only minor irregularity in these axioms is the check that an agent V has not voted on motion M ($\mathit{voted}(V, M) = \mathit{nil}$ $\mathsf{holdsat}$ T). This is because we want to prevent multiple votes but also allow an agent to change its mind (see later).

3.5 The Status of Motions

The status of a motion follows an obvious sequence as indicated by the values in Table 2. We have seen how when a motion has a particular status and an agent occupies a specific role, then the agent is empowered to perform an action. Generally, the exercise of the power by performing the action is sufficient to change the status of a motion: this effectively 'switches off' the existing power and 'switches on' other powers. The following illustrative EC axioms specify how the performance of an action by an empowered agent changes the status of a motion (and so changes powers).

$$\begin{aligned}
&\mathit{open\_ballot}(C, M) \;\; \mathsf{initiates} \;\; \mathit{status}(M) = \mathit{voting}(T) \;\; \mathsf{at} \;\; T \leftarrow \\
&\qquad \mathbf{pow}(C, \mathit{open\_ballot}(C, M)) = \mathit{true} \;\; \mathsf{holdsat} \;\; T \\
&\mathit{close\_ballot}(C, M) \;\; \mathsf{initiates} \;\; \mathit{status}(M) = \mathit{voted} \;\; \mathsf{at} \;\; T \leftarrow \\
&\qquad \mathbf{pow}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \;\; \mathsf{holdsat} \;\; T
\end{aligned}$$

$$\begin{aligned} &\mathit{declare}(C, M, \mathit{carried})\ \mathsf{initiates}\ \mathit{status}(M) = \mathit{resolved}\ \mathsf{at}\ T \leftarrow \\ &\qquad \mathbf{pow}(C, \mathit{declare}(C, M, \_)) = \mathit{true}\ \ \mathsf{holdsat}\ \ T \end{aligned}$$

Again, there are similar axioms for each of the other actions specified in Table 1. Note that declaring the result of a motion also adds the motion to the list of resolutions if, according to the standing rules, the vote has carried. Then, a new motion can be proposed: i.e., motions are dealt with sequentially rather than concurrently.

Note also that the standing rules can be anything 'reasonable' (majority of members, majority of those voting, etc.), but the rules themselves will have been negotiated and agreed as part of the institution. However, they can also be the subject of motions themselves (i.e. agents can propose to change the standing rules).

3.6 Roles & Role Assignment

We assume a predicate *qualifies* which holds of an agent belonging to the sets identified previously. A role assignment protocol (not addressed here) then determines, for those agents that qualify, who occupies the roles of voter, chair and monitor. It is, however, the acts of opening a session and proposing that (respectively) determine the set of proposers and seconders (i.e., of those agents qualified to act as proposer and/or seconder, which of them occupy the role; and by occupying the role they are empowered to propose or second motions). Occupying these roles is given by the following axioms:

$$\begin{aligned} &\mathit{open\_session}(C, M)\ \ \mathsf{initiates}\ \ \mathit{role\_of}(A, \mathit{proposer}) = \mathit{true}\ \ \mathsf{at}\ \ T \leftarrow \\ &\qquad \mathbf{pow}(C, \mathit{open\_session}(C, M)) = \mathit{true}\ \ \mathsf{holdsat}\ \ T\ \ \wedge \\ &\qquad \mathit{qualifies}(A, \mathit{proposer}) = \mathit{true}\ \ \mathsf{holdsat}\ \ T \\ &\mathit{propose}(A, M)\ \ \mathsf{initiates}\ \ \mathit{role\_of}(B, \mathit{seconder}) = \mathit{true}\ \ \mathsf{at}\ \ T \leftarrow \\ &\qquad \mathbf{pow}(A, \mathit{propose}(A, M)) = \mathit{true}\ \ \mathsf{holdsat}\ \ T\ \ \wedge \\ &\qquad \mathit{qualifies}(B, \mathit{seconder}) = \mathit{true}\ \ \mathsf{holdsat}\ \ T\ \ \wedge \\ &\qquad A \neq B \end{aligned}$$

Note that a minor clause of RONR stipulates that if an agent proposes a motion, then that same agent cannot also be the seconder for that motion. The check $A \neq B$ allows agents to occupy the role but also prevents the same agent both proposing and seconding the same motion.

Note that axioms are also needed to take agents 'out of role'; for example, the action *second* by an empowered agent initiates $\mathit{role\_of}(B, \mathit{seconder}) = \mathit{false}$ for all agents (including this one) that qualified as seconders.

3.7 Voting

3.7.1 Casting and Counting Votes

Once a motion has been proposed, seconded, and the ballot opened by the session chair, votes have to be cast and counted. The following axioms specify two aspects of voting. The first is how an *open_ballot* action on motion M initialises the vote count to zero and sets $voted(V, M)$ to *nil* for each agent V occupying the role of voter. The second is how a *vote* action performed by an empowered agent increments the count of votes 'for' and records the way its vote was cast (similar axioms are required for a vote 'against'):

$$
\begin{array}{l}
\mathit{open\_ballot}(C, M) \ \textsf{initiates} \ \mathit{votes}(M) = (0, 0) \ \textsf{at} \ T \leftarrow \\
\quad \mathbf{pow}(C, \mathit{open\_ballot}(C, M)) = \mathit{true} \ \textsf{holdsat} \ T \\
\mathit{open\_ballot}(C, M) \ \textsf{initiates} \ \mathit{voted}(V, M) = \mathit{nil} \ \textsf{at} \ T \leftarrow \\
\quad \mathbf{pow}(C, \mathit{open\_ballot}(C, M)) = \mathit{true} \ \textsf{holdsat} \ T \ \wedge \\
\quad \mathit{role\_of}(V, \mathit{voter}) = \mathit{true} \ \textsf{holdsat} \ T \\
\mathit{vote}(V, M, \mathit{aye}) \ \textsf{initiates} \ \mathit{votes}(M) = (F1, A) \ \textsf{at} \ T \leftarrow \\
\quad \mathbf{pow}(V, \mathit{vote}(V, M)) = \mathit{true} \ \textsf{holdsat} \ T \ \wedge \\
\quad \mathit{votes}(M) = (F, A) \ \textsf{holdsat} \ T \ \wedge \\
\quad F1 = F + 1 \\
\mathit{vote}(V, M, \mathit{aye}) \ \textsf{initiates} \ \mathit{voted}(V, M) = \mathit{aye} \ \textsf{at} \ T \leftarrow \\
\quad \mathbf{pow}(V, \mathit{vote}(V, M, \_)) = \mathit{true} \ \textsf{holdsat} \ T
\end{array}
$$

Note that an *abstain* would remove the power to vote but leave the vote count unchanged.

3.7.2 Revoking Votes

One feature of an election may be the idea of 'one man, one vote', i.e. each agent in the system only gets one vote. It is for this reason that if the fluent *voted* for some agent V did not have the value *abs*, then it meant that the agent had already cast its vote. Then, it no longer had the power to vote on this election, and any further vote actions it performed would not *count as* votes, so the fluent *votes* (for this M) would not be updated.

However, it is also a feature of voting in RONR that voters are allowed to change their minds, provided they do so before the result is announced. We will impose a slight restriction, and allow a change of vote until the ballot is closed. To accommodate this feature, we can introduce a *revoke* action. An agent occupying the role of voter would be empowered to revoke its vote (or its abstention) if, just as before, the motion status and role determined the power, but also if the value of the *voted* fluent was not *nil*. If it then exercised this power it would initiate two changes to the fluents. Firstly, the *voted* fluent would be 'reset' (to *nil*), and secondly, the *votes* fluent would need to be decremented according to how the original vote had been cast.

Some of the axioms for specifying this are as follows (axioms for changing a *nay* vote are similar):

$$\begin{aligned}
&\mathit{revoke}(V,M)\ \textsf{initiates}\ \mathit{votes}(M)=(F,A)\ \textsf{at}\ T \leftarrow\\
&\qquad \mathit{voted}(V,M)=\mathit{aye}\ \textsf{holdsat}\ T\ \ \wedge\\
&\qquad \mathit{status}(M)=\mathit{voting}\ \textsf{holdsat}\ T\ \ \wedge\\
&\qquad \mathit{votes}(M)=(F1,A)\ \textsf{holdsat}\ T\ \ \wedge\\
&\qquad F=F1-1\\
&\mathit{revoke}(V,M)\ \textsf{initiates}\ \mathit{voted}(V,M)=\mathit{nil}\ \textsf{at}\ T \leftarrow\\
&\qquad \mathit{voted}(V,M)=\mathit{aye}\ \textsf{holdsat}\ T\ \ \wedge\\
&\qquad \mathit{status}(M)=\mathit{voting}\ \textsf{holdsat}\ T
\end{aligned}$$

Note that an agent can also revoke its abstention.

3.7.3 Proxy Votes

We conclude this section with a brief remark that in a deviation from 'one man one vote' it is also possible to allow multiple votes if we allow proxy voting. To do this, we need a procedure (which we do not specify here) for establishing the power to represent. However, this procedure requires no extra expressive power beyond that which we already have: a specification of representation relations can be given in terms of institutional power. Representation then enables one agent (the representative) to act in the name of another (the principal). In our case, we can then get one agent to vote on behalf of another. Such representation is an important feature of the systems we are developing (see later), where presence 'at' the meetig or assembly may be transient.

4 Enfranchisement: Revisited

In this section, we consider how the formal specification of the previous section, in conjunction with other normative positions relating to permission and obligation, meets the requirements characterising our notion of enfranchisement identified in Section 2.3.

We begin with the 'right' aspect of enfranchisement. Section 2.3 partially characterised the right to vote by the power to vote. We have already specified how an agent has the power to vote if it occupies the role of voter (so it has previously qualified for this role), if the motion has the appropriate status (i.e. the ballot has been opened), and in 'one man one vote' elections, the agent has not already voted (or has voted but has revoked the vote).

The second element of the characterisation was that no other agent was empowered to object to the valid exercise of the power. By 'valid exercise' of power, we really mean permission. So, for this, we simply include an axiom of the form 'power implies permission'.

Note that there is no fixed relationship between power and permission: an agent may have the power but not the permission, and equally may have the permission to perform an action but not the power (so perfoming the action does not 'count as'). In our case, for this aspect of the right to vote, we include the axiom:

$$\begin{aligned}&\mathbf{per}(V, \mathit{vote}(V, M, \_)) = \mathit{true} \;\;\textsf{holdsat}\;\; T \;\;\leftarrow\\ &\quad\mathbf{pow}(V, \mathit{vote}(V, M, \_)) = \mathit{true} \;\;\textsf{holdsat}\;\; T\end{aligned}$$

and stipulate that there can be no valid objection to a permitted action.

The third aspect of the characterisation *is* an example of exercising power without permission. This was concerned with inappropriate removal of the power. There are in fact (at least) two ways of formulating the power to vote. One way is to always grant the power but only occasionally the permission. A second way, as used here, is to only occasionally grant the power but always give permission whenever the power is granted. However, this second way requires that there is some mechanism for 'switching on' and 'switching off' this power. We have already seen one: if an agent casts a vote, that removes its power (to vote again), although it can recover its power by revoking the vote. However, there is another one. This is when the chair agent closes the ballot: note that this too removes the power to vote because the action changes the status of the motion.

Therefore, for the action of closing the ballot, we do not want power to imply permission. It would certainly be undesirable if the chair could close a ballot at any time, especially as soon as its preferred result was in force, but not everyone had voted. On the other hand, for flexibility, we do not necessarily want to preclude the chair from closing a ballot ahead of time (for example, if a simple majority had already voted one way).

We can allow for both eventualities by specifying that the chair has the power to close a ballot, but does not have permission to exercise this power unless certain conditions are satisfied. Then, if the chair exercises its power without permission, the ballot will still be closed, but the chair agent will now be subject to a sanction (see the next section). For the sake of example, it could be that a certain minimum time should have elapsed since the opening of the ballot (say 10 time units). In addition, after a further time has elapsed (say 20 time units) the chair might even be *obliged* to close the ballot. Performing some actions when under obligation to perform another (e.g. closing a session while under obligation to close a ballot) can also be subject to a sanction.

This example could be specified by the following two axioms.

$$\begin{aligned}&\mathbf{per}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \;\;\textsf{holdsat}\;\; T \;\;\leftarrow\\ &\quad\mathit{role\_of}(C, \mathit{chair}) = \mathit{true} \;\;\textsf{holdsat}\;\; T \;\;\wedge\\ &\quad\mathit{status}(M) = \mathit{voting}(T') \;\;\textsf{holdsat}\;\; T \;\;\wedge\\ &\quad T > T' + 10\\ &\mathbf{obl}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \;\;\textsf{holdsat}\;\; T \;\;\leftarrow\\ &\quad\mathit{role\_of}(C, \mathit{chair}) = \mathit{true} \;\;\textsf{holdsat}\;\; T \;\;\wedge\end{aligned}$$

$$\begin{aligned}&\quad \mathit{status}(M) = \mathit{voting}(T') \ \textsf{holdsat}\ T \ \wedge \\ &\quad T > T' + 20\end{aligned}$$

The second element of enfranchisement was the element of entitlement. This in turn had three aspects. The first aspect was an entitlement to access the voting machinery: as already mentioned, this is a physical condition and not our concern here. The second aspect was an entitlement to have the vote counted correctly. This is effectively captured by the *votes* and *voted* fluents as described above. (Of course, in implementation as opposed to specification, there is an obligation on whoever is responsible for operating the 'voting machine' to record these fluents correctly; but again this is outside the scope of our concerns here.) Finally, the third aspect was the entitlement to a 'fair' outcome.

In an analysis of an argumentation protocol [5], between a proponent and an opponent, there was a determiner who declared the result at the end. The determiner was also obliged to declare the result for the proponent, if the opponent conceded, and vice versa.

The situation is much the same here. Therefore we can also formulate axioms which oblige the chair to declare the motion carried, if certain conditions have been satisfied, and equally, to declare the motion not carried if one or other of these conditions has not been satisfied. Note these 'certain conditions' are completely application dependent, and a codification of the standing rules of the committee. For example, typical conditions might be that the vote is quorate (however 'quorate' is defined by the standing rules), that there is a two-thirds majority in favour of those that have voted, and so on. Supposing that we wanted to simply implement majority voting, then the obligation on the chair would be expressed by the following axioms:

$$\begin{aligned}&\mathbf{obl}(C, \mathit{declare}(C, M, \mathit{carried})) = \mathit{true}\ \textsf{holdsat}\ T \leftarrow \\ &\qquad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true}\ \textsf{holdsat}\ T \ \wedge \\ &\qquad \mathit{status}(M) = \mathit{voted}\ \textsf{holdsat}\ T \ \wedge \\ &\qquad \mathit{votes}(M) = (F, A)\ \textsf{holdsat}\ T \ \wedge \\ &\qquad F > A \\ &\mathbf{obl}(C, \mathit{declare}(C, M, \mathit{not\_carried})) = \mathit{true}\ \textsf{holdsat}\ T \leftarrow \\ &\qquad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true}\ \textsf{holdsat}\ T \ \wedge \\ &\qquad \mathit{status}(M) = \mathit{voted}\ \textsf{holdsat}\ T \ \wedge \\ &\qquad \mathit{votes}(M) = (F, A)\ \textsf{holdsat}\ T \ \wedge \\ &\qquad F < A\end{aligned}$$

Notice that the obligation to declare the correct result also implicitly demands that the votes are counted according to the way they are cast.

Thus we see that certain agents are enfranchised (have the right, and are entitled to vote). This means that when a vote on a motion is called, these agents are empowered to vote. Taking all the votes together, the motion is carried if all the conditions are satisfied, and not carried if any condition is not satisfied. However, the chair is obliged to declare the result correctly with respect to these conditions (which are the codification of the standing rules of the committee). This obligation on the chair is the counter-point to the power which constitutes the entitlement to a 'fair' outcome.

5 Additional Issues

In this section, we briefly look further at two residual issues. First, we look further at the issue of sanctions, picking up on actions of the chair that were not permitted or contrary to obligation. Secondly, we consider the use of an objection mechanism, as another way of dealing with 'invalid' behaviour.

5.1 Sanctions

Sanctions and enforcement policies are a means of dealing with 'undesirable' behaviour, or behaviour that does not accord with the standing rules of the committee. We have already seen some indications of this: for example, declaring a motion as *carried* when the votes cast and standing rules dictate that it should not have carried; and closing a ballot without permission (to close the ballot).

Sanctions are heavily domain-dependent, and the actual form of representation is as complex as that of representing motions. In our formulation of the voting protocol, we associate a 3-figure 'sanction code' (in the same way that 3-figure error codes are used in Internet protocols) with each type of 'undesirable behaviour', and record, for each agent, a list of such codes which will be added to as a consequence of actions performed by the agent that are considered transgressions of acceptable behaviour. Note that what these describe are not sanctions *per se*, as there is no penalty applied: what we have instead are codes which identify violations so that the appropriate sanction (penalty) can be applied.

The following set of axioms illustrate the specification of sanctions for three of the 'undesirable' actions identified above (the others are similar). Note that with the sanction code, the motion related to the sanction is also recorded, and as the code numbers may indicate, this is not an exhaustive specification of all the sanctions.

$$\begin{aligned}
&\mathit{declare}(C, M, \mathit{carried}) \ \mathsf{initiates} \\
&\qquad\qquad \mathit{sanction}(C) = [(101, M)|S] \ \mathsf{at} \ T \ \leftarrow \\
&\quad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\
&\quad \mathbf{obl}(C, \mathit{declare}(C, M, \mathit{not\_carried})) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\
&\quad \mathit{sanction}(C) = S \ \mathsf{holdsat} \ T \\
&\mathit{close\_ballot}(C, M) \ \mathsf{initiates} \\
&\qquad\qquad \mathit{sanction}(C) = [(103, M)|S] \ \mathsf{at} \ T \ \leftarrow \\
&\quad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge
\end{aligned}$$

$$\begin{aligned}
&\qquad \mathbf{per}(C, \mathit{close\_ballot}(C,M)) = \mathit{false}\ \textsf{holdsat}\ T \ \wedge \\
&\qquad \mathit{sanction}(C) = S\ \textsf{holdsat}\ T \\
&\mathit{open\_ballot}(C,M)\ \textsf{initiates} \\
&\qquad\qquad\qquad \mathit{sanction}(C) = [(105,M)|S]\ \textsf{at}\ T \ \leftarrow \\
&\qquad \mathit{role\_of}(C, \mathit{chair}) = \mathit{true}\ \textsf{holdsat}\ T \ \wedge \\
&\qquad \mathbf{per}(C, \mathit{open\_ballot}(C,M)) = \mathit{false}\ \textsf{holdsat}\ T \ \wedge \\
&\qquad \mathit{sanction}(C) = S\ \textsf{holdsat}\ T
\end{aligned}$$

Note it is the second of these axioms, closing the ballot without permission, which incurs a sanction for 'violating' an agent's right to vote (although this not the only way that this can come about).

5.2 Objection

One important point to note about sanctions, is that agents which have powers, and exercise them even without permission, bring about a change of state regardless of whether the action was correct or not. Together with the sanction mechanism, this provides a substantial element of flexibility, which, as indicated earlier, was a desirable aspect of robustness. For example, given a proposal, we can give the chair the power to open a ballot without waiting for a seconder. The sanction (code 105) is duly recorded, but the vote can proceed and the decision taken regardless. Having identified a 'sanctionable action', the appropriate sanction can be applied at a later time.

What is also required, is a general objection mechanism, which retracts the effects of an 'objectionable' action, that is, an action 'not according to the rules' (of RONR). Depending on the protocol specification, an action can be objectionable (that is, its effects will be retracted if an agent objects), sanctionable (that is, the agent that performed it will be penalised), or both. An objection mechanism was used by Artikis et al [5] in their specification of an argumentation protocol, and a similar formulation can be applied here.

6 Animation and Implementation

In this section, we briefly consider the executable specification (animation) of the voting protocol as formalised in the previous section, and the issues this raises for use in a 'fully-fledged' implementation.

6.1 Animation

A simple pre-processor converts axioms written in the style of the previous section (with some minor syntactic variant) into Prolog code. It can then be queried in the same way that other protocols specified in this way have been (e.g. [4]). Testing of the voting protocol has mainly focused on systematic runs of exemplary narratives and manual inspection that the agents' powers, permissions, obligations and sanctions accord with the specification. However, formally proving specific properties has been discussed and demonstrated in [5, 3].

For example, consider a 'meeting' of a 'committee' containing four agents, cAgent, pAgent, sAgent, and vAgent. cAgent is assigned to the role of *chair*; pAgent and sAgent qualify to propose and second; and all four are assigned to the role of *voter*. An exemplary narrative is illustrated in Table 3 (this is LaTeX generated by a Tcl post-processor, from the logfile output by the Prolog coding of the voting protocol). The changes in the roles, powers, permissions, obligations and sanctions have all been described by the axioms in the previous section.

We make three observations about the animation. Firstly, the results of the animation can be inspected to determine that the specification works as intended. It can also be used as a basis for verifying specific properties of the system, as mentioned above, although we do not pursue this issue here. Secondly, while voting has some distinctive features of its own, there are similarities with specifications of other protocols. This in itself is corroboration that the norm-governed specification of protocols provides both patterns that can be re-used and descriptive adequacy when new issues arise. We contrast this to experience with the FIPA ACL specifications [10]. Thirdly, the animation raises a number of issues when this specification is encoded in a system implementation which correctly implements the protocol. These issues are discussed in the next section.

6.2 Implementation Issues

In this section, we briefly consider some of the implementation issues raised by the animation. These are:

- *Message transport*: the message transport needs to support both multi-cast messages (e.g. for proposals, seconds, and comments) and point-to-point messages (for votes) within the same protocol. Similarly, there are different levels of security required: proposals, comments etc. are *meant* to be open and read by all, votes may be private.
- *Agent types*: it must be determined if the target system consists of purely software components, only human 'agents' (and the software is a decision-support system [21]), or mixed. In each case, the extent to which the system should be regimented [15] (the agents can only do what they are permitted to do) has to be balanced against the flexibility to perform actions 'out of order'.
- *Self-modification*: the scope of motions is potentially very broad, and so not only can motions be about decisions to be made concerning the committee, there may also be motions about *the process* by which those decisions are reached. For example, there may be a motion to change the simple majority to a two-thirds majority of those that voted. To achieve this in the animation requires replacing the line $F > A$ with the line $(F/(F + A)) \geq .667$. To effect this in a real implementation requires interpretable code or some form of dynamic compilation.

Table 3: Sample Run of the Voting Protocol

agent	roles	powers	permissions	obligations	sanctions
cAgent	*chair voter*	*open_session*	*open_session*		
pAgent	*voter*				
sAgent	*voter*				
vAgent	*voter*				
happens($open\_session(cAgent, sesh)$)					
cAgent	*chair voter*	*close_session*	*close_session*		
pAgent	*voter proposer*	*propose*	*propose*		
sAgent	*voter proposer*	*propose*	*propose*		
vAgent	*voter*				
happens($propose(pAgent, m1)$)					
cAgent	*chair voter*	*close_session*			
pAgent	*voter proposer*				
sAgent	*voter proposer seconder*	*second*	*second*		
vAgent	*voter*				
happens($second(sAgent, m1)$)					
cAgent	*chair voter*	*open_ballot close_session*	*open_ballot*	*open_ballot*	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens($open\_ballot(cAgent, m1)$)					
cAgent	*chair voter*	*close_ballot close_session*			
pAgent	*voter proposer*	*vote*	*vote*		
sAgent	*voter proposer*	*vote*	*vote*		
vAgent	*voter*	*vote*	*vote*		
happens($vote(pAgent, m1, aye)$)					
cAgent	*chair voter*	*close_ballot close_session*			
pAgent	*voter proposer*				
sAgent	*voter proposer*	*vote*	*vote*		
vAgent	*voter*	*vote*	*vote*		
happens($vote(sAgent, m1, nay)$)					
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*		
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*	*vote*	*vote*		
happens($vote(vAgent, m1, nay)$)					
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*	*close_ballot*	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens($revoke(sAgent, m1)$)					
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*		
pAgent	*voter proposer*				
sAgent	*voter proposer*	*vote*	*vote*		
vAgent	*voter*				
happens($vote(sAgent, m1, aye)$)					
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*	*close_ballot*	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens($close\_ballot(cAgent, m1)$)					
cAgent	*chair voter*	*declare close_session*	$declare(carried)$	$declare(carried)$	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens($declare(cAgent, m1, not\_carried)$)					
cAgent	*chair voter*	*close_session*	*close_session*		102
pAgent	*voter proposer*	*propose*	*propose*		
sAgent	*voter proposer*	*propose*	*propose*		
vAgent	*voter*				
happens($close\_session(cAgent, sesh)$)					
cAgent	*chair voter*	*open_session*	*open_session*		102
pAgent	*voter*				
sAgent	*voter*				
vAgent	*voter*				

- *Sanctions*: the animation shows that it is possible to detect sanctions, and the code is helpful in indicating the nature of the violation. Applying and enforcing a *penalty* for the sanction is an entirely different matter, and this needs to be implemented relative to a legal contract agreed between the members of the committee.
- *Monitors*: the role of monitors in the voting protocol is to be informed of agents' votes and to verify that declared results do actually concur with the way that votes were cast.

It is this final issue – the correct declaration of the result according to the standing rules – which is of utmost importance in any 'real' system implementation which enacts the protocol. Recall the specification included a fluent *votes* for each motion, whose range was a 2-tuple. One element was incremented each time a vote was cast for, or against, the motion. However, the animation effectively takes an 'external' view of a 'perfect' system: in reality, the data structure for counting votes has to be stored somewhere, furthermore, it has to be accessed correctly, and it must support the correct decision being made (i.e. a guarantee that the chair has declared the result correctly according to the rules).

7 Applications

In this section, we demonstrate the general utility of the voting protocol by briefly considering its role in two applications. These are online deliberative assemblies and virtual organizations.

7.1 Online Deliberative Assemblies

We are developing a system for supporting online deliberative assemblies based on a concrete setting. This setting is illustrated in Figure 1.

The physical components of this system include:

- a shared whiteboard and projector: the shared whiteboard is a conventional technology used for allowing two or more people to view and write on a shared drawing surface. This equipment is set up in a fixed physical space, e.g. a meeting or conference room;
- the resource controller: a computer running the shared whiteboard server application, also containing a thread which implements the agent/controller that executes commands, grants access, etc. (cf. [3]);
- mobile devices: these are portable devices (e.g. phones, PDAs, laptops, etc.) that are carried into the physical space, connected to the server by some wireless communciations (e.g. Bluetooth), and run client agents (whose role/entitlement could be voters, proposers, seconders etc.);

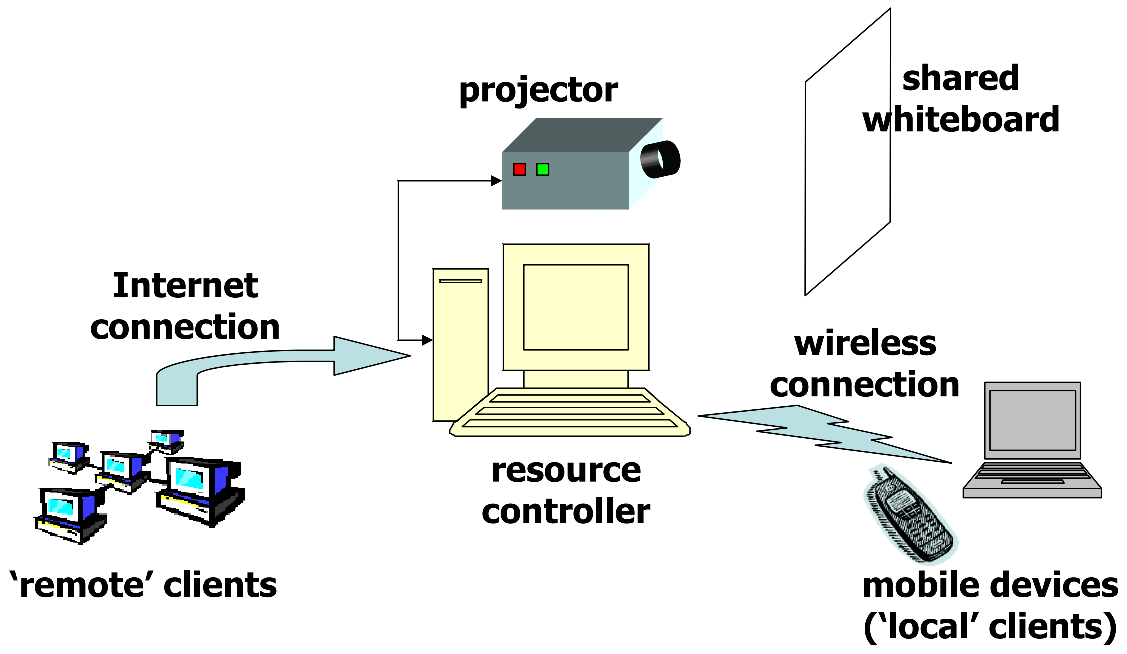

Figure 1: Platform for Online Deliberative Assemblies

- remote clients: as above, but situated in remote locations and connected by a fixed infrastructure (i.e. the Internet)
- networking: the system supports formation of ad hoc networks at the physical level (integrating mobile devices) and at the application level (local and remote clients can come and go at will).

The system has been inspired by the ZENO system, an automated system providing advice for human mediators of discussion groups and decision support functions for group decision-making applications [21]. There is much in ZENO, in particular in terms of its formalization of RONR, that can directly benefit this development. One additional element here is the exploitation of ad hoc networks, while another new element is the integration of a number of protocols. [21] outline a 'main loop' of their system: Figure 2 is an adaptation of their diagram. This diagram also shows that the implementation of this loop can be conducted through three protocols:

- *floor control protocol*: [3] specifies a floor control protocol which ensures that only a designated user or users can have access to certain objects (shared resources). This protocol is necessary to allow agents to acquire the floor in order to propose motions;
- *argumentation protocol*: [5] specifies an argumentation protocol which provides a procedure for discussion and dispute resolution. This protocol is necessary to allow

agents to debate the motion; the determiner can optionally be used to declare a 'winner' but in any case we expect that following the debate the motion will be put to a vote in any case;

- *voting protocol*: as formulated here. Note that a vote is technically possible for every issue that requires a decision, but there is an issue of granularity here. While it might make sense, for example, to vote on admission control, it would not make so much sense to take a vote every time someone requested the floor.

Note that the specification of each of these three protocols has the same conceptual and theoretical basis: therefore we expect the voting protocol to interoperate with the argumentation protocol since the same formalism has been used to specify both tasks.

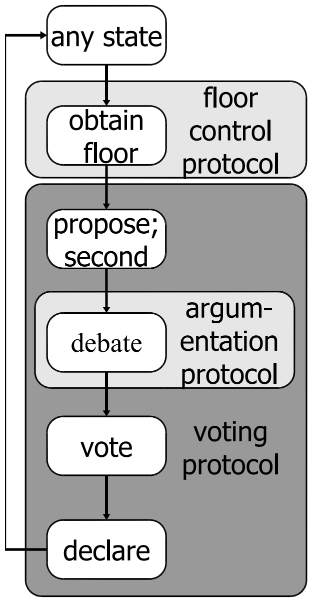

Figure 2: Protocols in Deliberative Assemblies

7.2 Virtual Organizations

From a legal perspective (cf. [7]) a virtual organization can be defined as a transient, inter-organizational, cross-border ICT-enabled collaboration between legally independent entities, usually with a specific economic goal. Where the 'ICT-enabled' (Information and Communication Technologies) element of the definition involves agents, and important aspects of the virtual organization's functionality are realised through agent interactions, we refer to the organization as a multi-agent virtual organization (MVO).

In this case, the economic motivations and characteristics are orthogonal to the main concerns of our investigation, which are the requirements and mechanisms for formation, run-time management, and dissolution of the virtual organization. Similarly, detecting the need or opportunity for an MVO and search techniques for finding a useful pre-existing MVO, which can be informed by 'intelligent' distributed algorithms, i.e. agents, are also outside the scope of this paper. However, from [7], we can identify several occasions in the life-cycle of an MVO where some kind of decision has to be made, and several features of an MVO which impact on how it is to be made. This includes:

- The MVO does not aim at achieving a legal status separate from its partners. No hierarchical structure is set up and the partners participate on a peer-to-peer basis (logically as well as physically in terms of the actual network).
- Autonomous agents are able to decide what actions each individually chooses to perform. They may also have an opinion on or preference for the actions each other should perform, or the MVO's collective goal.
- The formation of the MVO and the subsequent mapping of tasks to the partial business processes of individual partners is crucial. It is suggested in [7] that this can be performed by auctions, but auctions are just one 'tool' in the mechanism design 'tool-box'.
- The protection of personal data may be implemented by appropriate technical measures, and is the responsibility of an appointed controller, who can be chosen by agreement [7].
- The protection of Intellectual Property Rights (IPR) also needs to be considered by the MVO partners, and measures need to be developed in order to distinguish between freely utilisable and protected material. Equally, measures are required to determine the disposal of IPR, and in an MVO this may be the common ownership of multiple partners.
- In an open, non-hierachical structure like an MVO, actions may have unexpected and undesirable outcomes, and agents may take actions to further self-interest rather than the common good. In such cases, disputes may arise between partners, and a mechanism for dispute resolution is required.

Therefore, we have a non-hierachical system composed of independent and autonomous peers, who have to deal with formation of the MVO (which may include applications to join), action selection, appointment to 'roles', disposal of IPR, dispute resolution, and dissolution of the MVO. All of these cases require a decision to be made: and to reiterate, an ostensibly fair way to reach a consensus between independent peers on a potentially contentious issue is to take a vote. On this basis, a voting protocol is a prerequisite for coordination in an MVO.

8 Related Research

The formalization of protocols and rules of procedure in terms of powers, permissions, and so on has been applied to the contract-net protocol [4], an argumentation protocol [5] and a resource control protocol [3]. Prakken [20] gives a comprehensive specification of Robert's Rules of Order in first-order logic, but is not as detailed or as specific with respect to the normative aspects. In particular, the power to vote and the inextricably linked obligation on the chair to declare the result correctly is missing.

Related work on voting in multi-agent systems sees voting as an element of mechanism design in co-ordination, similar to auctions, team formation and negotiation. Therefore the emphasis of the study is the voting *strategy* to prevent manipulation, rather than the voting *procedure* to ensure correct results, which is the emphasis here. So, for example, Conitzer and Sandholm [9] develop a voting system for choosing candidates based on preference, whose computational complexity encourages voters to express their true preferences candidly, rather than engage in tactical voting. Our work focuses on the external specification of the decision-making process between all agents, not the internal deliberations of each agent. This is concerned with the dynamic system of normative positions that is modified by agents' interactions, and the dynamic system of norms that governs their behaviour, rather than specifying and proving properties of algorithms that optimise the outcome.

Electronic voting (e-voting) has been the subject of considerable interest from the e-government perspective. There are commercial systems available which offer to manage on-line elections (e.g. Votenet [14]), but these are not based on a representation of the norms governing the system. In addition, there has been significant work in designing a decision-support system (called ZENO) for online deliberative assemblies [21, 20] based on RONR. However, the principal concern here is maintaining public confidence in the electoral process, in particular that the winning candidate did actually receive a mandate from the popular vote. The fact that this requirement is integrally captured by the specification should be a factor in engineering 'correct' software. Therefore the approach developed here can be seen as complementary to conventional security techniques, but for electronic voting in public elections even this may not be enough, given the current limitations of security in host machines, in the Internet itself, or in the face of systematic malfeasance.

This work also has some bearing on the issue of communication in multi-agent systems, where no voting protocol has received the same attention as, for example, the contract-net protocol. FIPA specified the Borda Count Protocol [10] for voting, but only defined a structured exchange of messages in AUML, and not the meaning of the message themselves in terms of 'feasibility preconditions' and 'rational effects'. Therefore a thorough analysis and specification of a voting protocol seems to be required: the question then is how best to deliver it. There are at least four main approaches (with some overlap) to Agent Communication Languages, primarily based on the concepts they deal with: for example social commitments [25], joint intentions [8, 10], conversation policies [13], and normative systems (as here and in [4, 5, 3]). Considering the latter works, it is the case that negotiation, argumentation and resource control protocols can be specified in other ways. However, it is a challenge to see how voting can be characterised in terms of joint intentions, social commitments, or policies: at the very least, the additional concept of institutional power is also required.

9 Summary & Conclusions

This paper has been concerned with the issue of taking votes in multi-agent systems. The main aspect of our concern has not been in ensuring that the process is resistant to manipulation by strategic voting, nor that the process is as flexible as possible, but in ensuring the validity of the outcome. To this end, we identified a procedure for taking votes at meetings based on RONR, and then analysed what it mean to be 'enfranchised' in such a procedure. Our analysis (one of many) was based on the intuition that there appeared to be both an element of right *and* an element of entitlement that constituted the concept of 'enfranchisement'. We then showed how these conceptual requirements could be encapsulated in a formal specification of the voting protocol expressed in the Event Calculus. We also concentrated on the executable specification of this formalization of the voting protocol, and considered how this work could be applied for example in online-deliberative assemblies and virtual organizations. We note, however, that there remain several other aspects to formalise and implement, for example casting votes for the chair, recounts, candidate elections, and so on.

Clearly, though, this is a preliminary conceptualisation of right, entitlement and enfranchisement. A detailed investigation of these concepts in the context of voting, including an axiomatisation in EC (or some other formalism), is a direction for further research. There also remains substantial further work to complete both the specification and implementation. This work ranges from the incremental (such as studying variations of the protocol to handle, for example, concurrent motions, secret ballots, and candidate elections), to the radical, whereby different action languages such as $\mathcal{C}^+$ [12] or enhanced versions of the Event Calculus are used for the specification. The radical approach is required to understand better the alternative conceptual formalizations of rights (of voters) and duties (of vote counters) in voting protocols, and also to provide a computational platform more suited to representing and reasoning about these alternatives.

The voting protocol itself may turn out to be a valuable generic resource, as it appears to be central to many types of transient, ad hoc alliance in dynamic systems. It appears to be common across applications, but also across 'layers' within applications. For example, we have seen how voting occurred in formation, role assignment, and general management of a virtual organization. It has been used as a decision-support system in CSCW tools [21], and can also be used for admission, session and resource control in ad hoc networks. It is generally applicable to open systems which requires run-time modification or completion of a policy (partially) specified at design time. A formal, well-understood protocol for voting therefore offers a viable and in some cases more suitable alternative to other mechanisms for self-management and self-organization.

The analysis presented here has demonstrated the importance of normative concepts like powers, permissions, and obligations in socially-organized interaction. It has highlighted that the process by which a decision is reached must also preserve the validity of the outcome, i.e. the decision is correctly reached (this is a different matter to reaching the correct decision). In one sense, having a vote should be like having access to an abstract voting machine, which verifies the voter and correctly records the vote. This has effectively been encapsulated in the formalization presented here. It would be interesting to see how similar properties are given using social commitments or joint intentions.

The final challenge consists of correctly realising the specification in software. If we consider again the ACM Statement on E-Voting, and in particular the requirements that votes should be "accurately cast" and provide a "check on the result". We note that these are intrinsic properties of our specification: therefore if our implementation (verifiably) meets the specification then it is satisfying these requirements.

Acknowledgements

This paper is an updated and extended version of [18, 19]. We would like to thank the reviewers and participants of the AAMAS '05 and ICAIL '05 conferences and the reviewers of this paper who gave us useful feedback.

This research is being carried out with the support of the NoGoSoN project and is funded by the UK EPSRC Grant No. (GR/S69252/01).

References

[1] ACM. ACM Statement on E-voting. `http://www.acm.org/usacm/weblog/index.php?p=73`, 2004.

[2] K. Arrow. *Social Choice and Individual Values (Second Edition)*. Yale University Press, 1970.

[3] A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A protocol for resource sharing in norm-governed ad hoc networks. In *Proceedings of the Declarative Agent Languages and Technologies (DALT) Workshop*. Springer Verlag, 2004 (to appear).

[4] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings AAMAS'02*, pages 1053–1062. ACM Press, 2002.

[5] A. Artikis, M. Sergot, and J. Pitt. An executable specification of an argumentation protocol. In *Proceedings of International Conference on Artificial Intelligence and Law (ICAIL)*, pages 1–11. ACM Press, 2003.

[6] A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the causal calculator. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Proceedings AOSE'03*, volume LNCS2585, page to appear. Springer-Verlag, 2003.

[7] C. Cevenini. Legal considerations on the use of software agents in virtual enterprises. In J. Bing and G. Sartor, editors, *The Law of Electronic Agents*, volume CompLex 4/03, pages 133–146. Oslo: Unipubskriftserier, 2003.

[8] P. Cohen and H. Levesque. Communicative actions for artificial agents. In V. Lesser, editor, *Proceedings ICMAS95*. AAAI Press, 1995.

[9] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, 2003.*, pages 781–788. 2003.

[10] FIPA. FIPA'97 specification part 2: Agent communication language. FIPA (Foundation for Intelligent Physical Agents), http://www.fipa.org, 1997.

[11] B. S. Firozabadi and M. Sergot. Contractual access control. In B. Christianson, B. Crispo, J. Malcolm, and M. Roe, editors, *Proceedings 10th International Workshop on Security Protocols*, volume 2845 of *LNCS*, pages 96–102. Springer-Verlag, 2004.

[12] E. Giunchiglia, J. Lee, N. McCain, V. Lifschitz, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2003.

[13] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume LNAI1916, pages 118–131. Springer-Verlag, 2001.

[14] V. S. Inc. Votenet. http://www.votenet.com.

[15] A. Jones and M. Sergot. On the characterization of law and computer systems: The normative systems perspective. In J.-J. Meyer and R. Wieringa, editors, *Deontic Logic in Computer Science*. John Wiley and Sons, 1993.

[16] A. Jones and M. Sergot. A formal characterisation of institutionalized power. *Journal of the Interest Group in Pure and Applied Logics*, 4(3):429–455, 1996.

[17] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.

[18] J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Formalization of a voting protocol for virtual organizations. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. Singh, and M. Wooldridge, editors, *Proceedings 4th AAMAS'05*, pages 373–380. ACM, 2005.

[19] J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in online deliberative assemblies. In A. Gardner and G. Sartor, editors, *Proceedings 10th ICAIL*, pages 195–204. ACM, 2005.

[20] H. Prakken. Formalizing Robert's Rules of Order: An experiment in automating mediation of group decision making. GMD report 12 (www.bi.fraunhofer.de/publications/report/0012/), 1998.

[21] H. Prakken and T. Gordon. Rules of order for electronic group decision making. a formalization methodology. In J. Padget, editor, *Collaboration between Human and Artificial Societies: Coordination and Agent-Based Distributed Computing*, volume 1924 of *LNAI*, pages 246–263. Springer-Verlag, 1999.

[22] H. Robert and Others. *Robert's Rules of Order Newly Revised 10th edition*. Cambridge, Mass.: Perseus Publishing, 2000.

[23] J. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[24] J. Searle. What is a speech act? In A. Martinich, editor, *Philosophy of Language*, pages 130–140. Oxford University Press, third edition, 1996.

[25] M. Venkatraman and M. Singh. Verifying compliance with commitment protocols: Enabling open web-based multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.