Formalization of a Voting Protocol for Virtual Organizations

Jeremy Pitt, Lloyd Kamara
Intelligent Systems & Networks Group
Dept. of Electrical & Electronic Engineering
Imperial College London, London, SW7 2BT, UK
{j.pitt,l.kamara}@imperial.ac.uk

Marek Sergot, Alexander Artikis
Computational Logic Group
Department of Computing
Imperial College London, London, SW7 2AZ, UK
{mjs,aartikis}@doc.ic.ac.uk

ABSTRACT

A voting protocol for decision-making in virtual organizations is presented. In an agent-based virtual organization the functions of formation, management and dissolution of the organization are passed to software processes. Each phase in this life-cycle requires decision making: an ostensibly fair way for independent agents to make decisions is to take a vote. Accordingly, this paper formalizes a protocol for voting. The emphasis is on characterising the powers, permissions, obligations and even sanctions of the voters, using a norm-governed approach to agent societies. The specification language is the Event Calculus, and its animation is informative with respect to a full implementation. It is well-known that various types of ad hoc alliance of autonomous entities require voting procedures, and a normative specification of the interactions is therefore beneficial for many aspects of self-organization and self-management.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]

General Terms

Theory

Keywords

Multi-Agent Systems, Virtual Organizations, Voting

1. INTRODUCTION

A virtual organization is a temporary network of independent companies linked together through information and communication technologies (ICT), in order to share resources and services, to access each other's market, or to form an opportunistic alliance to exploit some business opportunity. An agent-based virtual organization passes the functions of detection (of the need for), discovery (of other pre-existing), formation, run-time management and dissolution of the organization to software processes (i.e. agents). However, each phase in the virtual organization life-cycle requires decision making: an ostensibly fair way to reach a consensus between independent peers on a potentially contentious issue is to take a vote.

Accordingly, this paper formalizes a voting protocol for multi-agent virtual organizations. In particular, the emphasis is on characterising the roles, (institutional) powers, permissions, obligations and even sanctions of the voters (agents). While voting is a well-known aspect of mechanism design for multi-agent systems [7], we use a norm-governed approach to agent societies, whereby appropriate behaviour is stipulated using concepts stemming from the study of legal and social systems: i.e. powers, permissions, and obligations; and possibly other more complex normative relations. Furthermore, software tools can then be used to analyse and execute formal specifications of such systems, providing a better understanding of the system's properties and finer-grained control over actual operation.

This paper is organized as follows. Section 2 considers virtual organizations and the requirements for voting. Section 3 informally introduces a voting protocol, based on Robert's Rules of Order [18], the standard handbook for conducting business in deliberative assemblies. Section 4 presents a formal specification of this protocol via a logical axiomatisation. The specification language is the Event Calculus [14], and so the specification itself is executable. The executable specification is demonstrated in Section 5, which raises a number of issues to be addressed by a full implementation. We discuss related research in Section 6, with respect to voting as an aspect of mechanism design in multi-agent systems, voting in e-government applications, and voting in the general context of agent communication languages. We draw some conclusions in Section 7, in particular that a voting procedure is a generic requirement in various types of ad hoc alliance of autonomous entities, and so a formal specification is beneficial for many aspects of self-organization and self-management.

2. VIRTUAL ORGANIZATIONS

From a legal perspective (cf. [5]) 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' 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

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

are orthogonal to the main direction of our investigation, which is 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 [5], 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 [5] 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 [5].
- 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.

3. A VOTING PROTOCOL

In this section, we give a brief, informal description of a voting protocol, based on Robert's Rules of Order (Newly Revised) [18], henceforth RONR; and consider which agent or agents can perform the steps in the protocol.

According to RONR, an appropriate procedure (protocol) for conducting a vote in a deliberative assembly is as follows:

- the assembly sits and the chair opens a session;
- a member proposes (tables) 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 assembly.

In the context of an MVO, we will associate a 'deliberative assembly' with the MVO itself, the assembly members with the MVO partners (or rather, the software agents representing the interests of those partners), and a meeting of the partners with a session (during its life cycle, an MVO may have several sessions, at which decisions are to be made). The term agent will be used for 'member' in the subsequent analysis.

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 on the set of agents. Determining set membership is not addressed 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 an agent 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 is based on the *roles* that the agents can occupy, and includes:

voter, i.e. those agents who are empowered 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 the votes cast and the decisions reached.

In the next section, we formalise the RONR protocol with two minor variations. Firstly, we count votes concurrently rather than sequentially, and secondly, we omit the 'debate' phase. The former may have consequences for strategic behaviour, while the latter can be expressed in the same terms [4], and so integrated with the approach used here. In neither case does the omission impact on the conceptual understanding and formalization of the voting protocol.

4. EVENT CALCULUS SPECIFICATION

The Event Calculus is a formalism from Artificial Intelligence intended to reason about action and change in non-monotonic systems [14]. There are several variants of the calculus: the version used here is that used in [2]. 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. For the latter two, we write axioms of the form:

$$\begin{aligned} &Action \text{ initiates } Fluent{=}Value \text{ at } Time \leftarrow \\ &\quad Condition_1 \wedge \ldots \wedge Condition_m \\ &Fluent{=}Value \text{ holdsat } Time \leftarrow \\ &\quad Condition_1 \wedge \ldots \wedge Condition_n \end{aligned}$$

We now give an indicative formal specification (via a logical axiomatisation) of the informal description of the voting protocol. This specification covers the fluents and actions, the status of motions, (institutional) powers, roles, permissions and obligations, and finally sanctions. The domain-independent part of the EC is coded in Prolog, and so the axiomatisation reflects that orientation.

4.1 Fluents and Actions

Table 1 lists the fluents used in the voting protocol. A session S can be open (sitting), or not. The status of a motion is pending (initially), then proposed, seconded, voting (the chair has called for votes at 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.

Table 1: Fluents in the Voting Protocol

Fluent	Range
$sitting(S)$	boolean
$status(M)$	$\{pending, proposed, seconded$ $voting(T), voted, resolved\}$
$votes(M)$	$N \times N$
$voted(Ag, M)$	$\{nil, aye, nay, abs\}$
$resolutions(S)$	list of motions
$qualifies(Ag, R)$	boolean
$role\_of(Ag, R)$	boolean
$\mathbf{pow}(Ag, Act)$	boolean
$\mathbf{per}(Ag, Act)$	boolean
$\mathbf{obl}(Ag, Act)$	boolean
$sanction(Ag)$	list of integers

Table 2 lists the actions that agents can perform in the voting protocol. The table is self-explanatory.

4.2 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. [13]). 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 eight actions listed in Table 2.

Table 2: Actions in the Voting Protocol

$open\_session(Ag, S)$	$close\_session(Ag, S)$
$propose(Ag, M)$	$second(Ag, M)$
$open\_ballot(Ag, M)$	$close\_ballot(Ag, M)$
$vote(Ag, M, aye)$	$vote(Ag, M, nay)$
$abstain(Ag, M)$	$revoke(Ag, M)$
$declare(Ag, M, carried)$	$declare(Ag, M, not\_carried)$

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

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

4.3 The Status of Motions

The status of a motion follows an obvious sequence as indicated by the values in Table 1. 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).

$$\begin{aligned} &open\_ballot(C, M) \text{ initiates } status(M) = voting(T) \text{ at } T \leftarrow \\ &\quad \mathbf{pow}(C, open\_ballot(C, M)) = true \text{ holdsat } T \\ &close\_ballot(C, M) \text{ initiates } status(M) = voted \text{ at } T \leftarrow \\ &\quad \mathbf{pow}(C, close\_ballot(C, M)) = true \text{ holdsat } T \\ &declare(C, M, carried) \text{ initiates } status(M) = resolved \text{ at } T \leftarrow \\ &\quad \mathbf{pow}(C, declare(C, M, \_)) = true \text{ holdsat } T \end{aligned}$$

Again, there are similar axioms for each of the other actions specified in Table 2. Note that declaring the result of a motion also adds the motion to the list of resolutions if, according to the standing rules of the MVO, 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 during the formation of the MVO. However, they can also be the subject of motions themselves (i.e. agents can propose to change the standing rules).

4.4 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; whereby they are empowered to propose or second motions). Occupying these roles is given by the following axioms:

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

Note that a minor clause of RONR stipulates that if an agent proposes a motion, then the 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 that qualified as seconders.

4.5 Voting 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 $\mathit{voted}(V, M)$ 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'):

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

RONR also states that an agent is empowered to change its vote before the result is announced. We will impose a slight restriction, and allow a change of vote until the ballot is closed. The *revoke* action resets the values changed by the *vote* action, as given by the following axioms (axioms for a *nay* vote are similar):

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

An *abstain* would remove the power to vote but leave the vote count unchanged. An agent can revoke its abstention.

4.6 Permission and Obligation

We note, en passant, that there is no fixed relationship between powers and permissions. In some cases, an agent may be permitted to perform an action simply if it is empowered. This is the case for the ordinary member's *propose*, *second* and indeed *vote* actions. However, it is not the case for the chair's *open_ballot* and *close_ballot* actions. For example, with closing a ballot, we need to trade off 'correctness' (ensuring that every agent gets to vote, for example) against flexibility (we do not want the protocol to block, waiting for some agent to vote).

As an example, then, we could specify that the chair has the power to close a ballot at any time after the ballot has opened. However, we might also specify that the chair does not have permission to exercise this power until a certain condition has been met. We might also specify that under other conditions the chair might even be *obliged* to close the ballot. As an illustrative example, the following axioms give permission when more than half the agents have voted (assuming, say, there are 3 agents to vote), and an obligation to close the ballot when all the agents have voted:

$$
\begin{aligned}
&\mathbf{per}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \ \mathsf{holdsat} \ T \ \leftarrow \\
&\quad \mathbf{pow}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\
&\quad \mathit{votes}(M) = (F, A) \ \mathsf{holdsat} \ T \ \wedge \\
&\quad (F + A) \geq 2 \\
&\mathbf{obl}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \ \mathsf{holdsat} \ T \ \leftarrow \\
&\quad \mathbf{pow}(C, \mathit{close\_ballot}(C, M)) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\
&\quad \mathit{votes}(M) = (F, A) \ \mathsf{holdsat} \ T \ \wedge \\
&\quad (F + A) = 3
\end{aligned}
$$

Another obligation on the chair is to declare the result of the vote; furthermore the declaration must be correct with respect to the votes cast and the standing rules. The following axiom specifies a simple majority vote:

$$
\begin{aligned}
&\mathbf{obl}(C, \mathit{declare}(C, M, \mathit{carried})) = \mathit{true} \ \mathsf{holdsat} \ T \leftarrow \\
&\quad \mathbf{pow}(C, \mathit{declare}(C, M, \_)) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\
&\quad \mathit{votes}(M) = (F, A) \ \mathsf{holdsat} \ T \ \wedge \\
&\quad F > A
\end{aligned}
$$

This obligation is crucial because it makes the power of voting truly meaningful: one can question the extent to which an agent is truly empowered to vote if the chair is under no obligation to declare the result according to the way the votes are cast. A detailed investigation of this issue is beyond the scope of this paper, but it is addressed in [15].

4.7 Sanctions

Permissions and obligations are the mechanism for identifying 'undesirable' behaviour. Sanctions are the mechanism for addressing such behaviour. Relevant examples here include declaring a motion carried when it should not be, closing a ballot without permission, trying to vote twice, an agent seconding its own proposal, and so on.

Sanctions are heavily domain-dependent, and the actual form of representation is as complex as that of representing motions. In this formulation of the voting protocol, we will 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. We then record, for each agent, a list of such codes (initially empty). A code is added to the list as a consequence of actions performed by the agent that are considered transgressions of acceptable behaviour.

For example, the axiom below associates 'sanction code 102' with the act of declaring a motion *not_carried* contrary to an obligation (to declare it *carried*:

$$\begin{aligned} &\mathit{declare}(C, M, \mathit{not\_carried}) \ \mathsf{initiates} \\ &\qquad \mathit{sanction}(C) = [(102, M)|S] \ \mathsf{at} \ T \ \leftarrow \\ &\quad \mathbf{obl}(C, \mathit{declare}(C, M, \mathit{carried})) = \mathit{true} \ \mathsf{holdsat} \ T \ \wedge \\ &\quad \mathit{sanction}(C) = S \ \mathsf{holdsat} \ T \end{aligned}$$

4.8 Additional Issues

We conclude this section with some brief remarks on three other issues: casting votes, proxy votes, and objections.

According to RONR, in some situations, the chair gets a vote (the casting vote) only when its vote would change the result. In a majority vote with a tie, if $F = A$, the motion fails; however, the chair can (is empowered to) cast a vote for the motion which would then carry. Similarly, if $F = A + 1$, the chair can cast a vote against the motion which then does not carry. Alternatively, the chair can abstain. (The motivation for this is to ensure that the chair remains impartial.) Therefore the specification must ensure that whichever agent occupies the role of *chair* is not necessarily empowered to vote if it also occupies the role of *voter*, and also, an additional axiom is required to empower the chair to vote under the conditions indicated above.

It is also possible to allow proxy votes if there is a procedure (which we do not specify here) for establishing the power to represent. A specification of representation relations can however be given in terms of institutional power. Representation then enables one agent (the representative) to act in the name of another (the principal). In this case, we can then empower one agent to vote on behalf of another.

Finally, when all actions are performed 'in order' and the result declared correctly, there is no problem. However, for the sake of flexibility, an MVO may tolerate deviation from the required behaviour. Examples are putting a motion to a vote without waiting for a seconder; or closing a ballot before everyone has voted if there is overwhelming support for the motion. We can then distinguish different levels of 'seriousness' for behaviour which deviates from the ideal. We can still use the sanction code to identify the 'inappropriate' behaviour but the penalties can reflect the perceived seriousness of the violation. Penalties associated with sanctions can come in many forms (cf. [4, 2]).

However, what is also required is an objection action, which retracts the effect of an action that was 'not according to the rules'. A treatment of such an action has been specified in [4], as part of an argumentation protocol, and could also be used here.

5. ANIMATION & IMPLEMENTATION

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

5.1 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. [3]). 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 accords with the specification. However, formally proving specific properties has been discussed and demonstrated in [4, 2].

For example, consider a 'meeting' of the MVO 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 simlarities 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 [8]. 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.

5.2 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 [17]), or mixed. In each case, the extent to which the system should be regimented [12] (the agents can only do what they are permitted to do) has to be balanced against the flexibility to perform actions 'out of order'.

Table 3: Sample Run of the Voting Protocol

agent	roles	powers	permissions	obligations	sanctions
cAgent	*chair voter*	*open_session*	*open_session*		
pAgent, sAgent, vAgent: role of *voter* only [manual edit to fit page size]					
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, sAgent, vAgent: role of *voter* only [manual edit to fit page size]					

- *Self-modification*: the scope of motions is potentially very broad, and so not only can motions be about decisions to be made concerning the MVO, 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 2/3 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.
- *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 partners in the MVO.
- *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).

This is important in light of the 2004 ACM Statement on E-Voting [1], which includes the stipulation 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.

6. 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 [3], an argumentation protocol [4] and a resource control protocol [2]. Prakken [16] 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 [7] develop a voting system for choosing candidate 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 online elections (e.g. Votenet [11]), 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 [17, 16] 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 [8] 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 [20], joint intentions [6, 8], conversation policies [10], and normative systems (as here and in [3, 4, 2]). 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.

7. SUMMARY & CONCLUSIONS

The development of virtual organizations has benefited considerably from agent technology, from implementation (e.g. [19]) to legal perspectives [5], and voting is a well-known aspect of mechanism design for multi-agent systems [7]. Here, we have considered a virtual organization based on agent technology as a kind of open agent society. One approach to specifying such a society is to use the concepts of norm-governed systems [3]. This work was concerned, then, with taking a norm-governed systems approach to multi-agent virtual organizations (MVO). As a potentially transient system, an MVO has a life cycle: we observed that there were many occasions in this life cycle were a decision was required. As a non-hierarchical system of independent, autonomous and self-interested peers, an ostensibly fair way to reach these decisions is to take a vote. We therefore developed a voting protocol for norm-governed virtual organizations. This protocol was specified in the Event Calculus

and has been animated in Prolog, which has been useful in highlighting key issues in implementation. We are currently developing a run-time implementation of this formalization taking these considerations into account.

There also remains, however, 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 actions languages such as $\mathcal{C}^+$ [9] 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 [17], and can also be used for admission, session and resource control in ad hoc networks. It is generally applicable to for open system 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 alternative mechanisms for self-management and self-organization.

Finally, 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.

8. ACKNOWLEDGMENTS

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). We gratefully acknowledge the useful comments of the anonymous reviewers.

9. REFERENCES

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

[2] A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A protocol for resource sharing in norm-governed ad hoc networks. In *Proc. DALT'04 Workshop*. Springer Verlag, 2004 (to appear).

[3] 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.

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

[5] 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.

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

[7] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proc. 18th IJCAI'03, Acapulco, Mexico, 2003*, pages 781–788. 2003.

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

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

[10] 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.

[11] Voting Systems Inc. Votenet. http://www.votenet.com.

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

[13] 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.

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

[15] J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in online deliberative assemblies. In A. Gardner and G. Sartor, eds., *Proc. ICAIL'05*. 2005 (to appear).

[16] 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.

[17] 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*, volume 1924 of *LNAI*, pages 246–263. Springer-Verlag, 1999.

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

[19] W. Vasconcelos, D. Robertson, C. Sierra, M. Esteva, J. Sabater, and M. Wooldridge. Rapid prototyping of large multi-agent systems through logic programming. *Annals of Mathematics and Artificial Intelligence*, 41:135–169, 2004.

[20] 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.