

Collective Choice in Virtual Teams

Alois Ferscha and Christoph Scheiner
Department of Applied Computer Science
University of Vienna
[ferscha/scheiner]@uni.univie.ac.at

Abstract

Virtual organisations within and across enterprise structures are becoming mature as a potentially effective means for goal oriented business teamwork. Among the plentyful of support functions enabling the efficient cooperation in teams, team decision making is a very crucial one, but does not find adequate support in contemporary team software solutions. Even the most knowledgable results from collective choice theory are not reflected in the respective virtual team environments.

This paper makes a first and very systematic attempt to provide virtual teams with the power of voting procedures developed in the context of the theory of collective choice. A toolset, **TEAMVOTE**, for electronic decision making is presented, implementing a blend of the most important collective choice strategies like the Dodgson rule, the majority rule, plurality voting, Borda counting, approval voting, the Nanson rule, the Hare rule, the Coomb rule etc. based on standard Internet technology. The services of **TEAMVOTE** targeted to support goal oriented business teams in group decision matters raises the perspective of its use also to enable democratic processes on a societal scale.

Keywords: Virtual Teams, Group Decision Making, Electronic Voting, Collective Choice Theory.

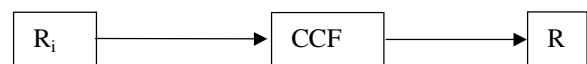
1 Introduction

Organisations like virtual teams which by nature of their configuration are geographically dislocated have raised demands for electronic decision making facilities via the networking infrastructure defining their workplace. Among the obvious advantages of electronic voting systems are the time and place independence of voters, the ease of administration with respect to cost and speed - the prevalent problems are authentication of voters and security and privacy when submitting a ballot.

Although a growing need for electronic voting system is perceived, not many efforts have been undertaken to implement flexible voting systems that can handle the basic collective choice procedures developed and discussed in collective choice theory [1]. Before discussing contemporary electronic voting systems, we first sketch basic collective choice principles within Section 1. The software architecture of **TEAMVOTE** and the Internet technologies used to implement it are outlined in Section 2, the user interface is illustrated in Section 3. Conclusion and perspectives are given in Section 4.

1.1 Collective Choice Background

After the pioneering work of Arrow [3] and Senn [1] (operationally summarized by Nurmi [2] [9]) a collective choice problem is formulated as a choice function CCF:



aggregating individual preference relations R_i to a collective preference relation R on a set of alternatives A . The individual preference relation R_i are assumed to be *reflexive*, *complete* (for any pair of alternatives $x, y \in A$ there has to be a preference among them) *transitive* – where *weak preference* (indifference) among alternatives is allowed. An alternative $x \in A$ is called *best alternative* if xRy for all other alternatives $y \in A$ (x is at least as good as all y) - if there is more than one best alternative, the set of best alternatives is called the *choice set*.

According to Arrow [3] a CCF should adhere at least 4 properties:

- **U** : unrestricted domain: Any combination of voters and alternatives must be allowed
- **P** : Pareto Criterion: If all voters (R_i) prefer an alternative x over alternative y , then the CCF should yield that x is also preferred over y in R
- **I** : Independence of irrelevant alternatives: The preference order of alternatives x and y in R should be only influenced by the individual preference (R_i) order of x and y
- **D** : Non dictatorship: No individual preference (R_i) may determine the R as a dictator

A well know result along the definition and desing of CCFs is:

Arrows Theorem [3]: There cannot exist a CCF simultaneously satisfying **U**, **P**, **I**, and **D**, if the number of voters ≥ 2 and the number of alternatives ≥ 3 .

Besides this discouraging result, yet another set of desirable CCF features have emerged in the literature, making the design of collective choice functions even more difficult. The most important ones are:

- The Condorcet winner criterion: an alternative having majority against all other alternatives should win the poll.
- The Condorcet loser criterion: an alternative dominated by all other alternatives should not win.
- Monotonicity: If an alternative is the winner of a poll, it should remain the winner also in the case if it would have been ranked higher by one or more voters.

- Weak axiom of revealed preference: if an alternative x is the winner of a poll, it should remain the winner, if the poll is on a partial set of the alternatives that contains the alternative x .
- Path Independence: There should be no difference if the vote is made on the combined subsets *or* the whole set of alternatives.
- Consistency: There should be no difference, if the combined subsets or the whole set of voters vote.

Several collective choice procedures have emerged over the past two centuries since the early work of the Marquis de Condorcet (1743-1794), which can be categorized into *binary procedures* comparing individual preferences pair to pair, *single step procedures* establishing R in one aggregation step out of R_i and *multi step procedures* iterating and/or combining single step procedures. The most common representatives in each category are:

Binary procedures:

- Copeland: A score is computed for each alternative, based on the sum of all wins of the alternative minus the sum of all losses of that alternative. The alternative with the highest score wins.
- Dodgson: A score computed based on the necessary inversions for each alternative to become a condorcet winner. The alternative with the smallest score wins.
- Majority Rule: a complete pair to pair comparison of all alternatives is conducted.

Single step procedures:

- Plurality voting: The alternative with the highest score of first places wins the poll.
- Borda: The voter assigns a score to each alternative (ranging from 0 to the number of alternatives minus one), such that each of these scores is assigned only once. The winner is the alternative with the largest total of scores [11].
- Approval voting: Each voter approves or disapproves to alternatives (has to approve to at least one alternative). The winner is the alternative with the highest number of approvals [10].

Multi step procedures:

- Plurality Runoff: If two (or more) alternatives end up in ties according to plurality voting, the tie is broken by a new poll among the alternatives in the choice set – again using plurality voting.
- Nanson: a winner is determined by iterating the Borda rule and eliminating the lowest Borda score alternative in each iteration.
- Hare: Like Nanson, but in each iteration the alternative that has the least first places is eliminated.
- Coomb: Like Nanson, but after each iteration the alternative with the most last places is eliminated.

The relative strenghts of the respective collective choice procedures are summarized in the following table (see also Nurmi [2]):

| | Condorcet Winner | Condorcet Loser | Monotonicity | Pareto | WARP | Path Independence | Consistency | Simplicity | Easiness |
|------------------|------------------|-----------------|--------------|--------|------|-------------------|-------------|------------|----------|
| Copeland | X | X | X | X | O | O | O | O | X |
| Dodgson | X | O | X | X | O | O | O | X | O |
| Majority Rule | X | X | X | O | O | O | O | O | X |
| Plurality | O | O | X | X | X | O | X | X | X |
| Borda | O | X | X | X | O | O | X | X | O |
| Approval | O | O | X | X | X | X | X | X | X |
| Plurality Runoff | O | X | O | X | O | O | O | X | X |
| Nanson | X | X | O | X | O | O | O | X | O |
| Hare | O | X | O | X | O | O | O | X | O |
| Coomb | O | X | O | X | O | O | O | X | O |

The two additional fields "Simplicity" and "Easiness" annotate a characterisation of the respective procedure from the implementation viewpoint (Simplicity), and from a usability viewpoint for casting a vote within an electronic voting system (Easiness). Already from this table explaining the relative strengths and weaknesses of CCFs it appears advisable to cover at least this minimum blend of choice procedures within a team decision making tool – as from team decision matter to team decision matter some choice rules are more adequate than others. TEAMVOTE provides business teams with all these CCFs.

Besides the coverage of rules, a set of other requirements to electronic voting systems should be adhered [4]:

- Accuracy: It should not be possible for a ballot to be altered at any time by any means once it has been dispatched by the voter, neither should it be possible for a dispatched ballot (in our case a ballot that has been entered to TEAMVOTE) to be eliminated from the final tally, nor should it be possible for an invalid ballot to be counted.
- Invulnerability: Only eligible voters are allowed to vote and each eligible voter can only vote once.
- Privacy: Neither election authorities nor anyone else can link any ballot to the voter who casted it.
- Verifiability: Anyone can independently verify that the ballots have been counted correctly.
- Convenience: The voting system should allow voters to cast their votes quickly, in one session, and with minimal equipment(s) or special skills.
- Flexibility: The system should allow a variety of ballot formats including open ended questions.
- Mobility: There should be no restrictions on the (geographical) location from which a voter can cast a vote.

TEAMVOTE satisfies all of the proposed requirements, except contradicting ones: it is almost impossible to provide ways that make a system private and verifiable at the same time. Before presenting TEAMVOTE in more detail, however, a view to related research efforts and commercial software packages with the potential for virtual team decision making support are to be referenced:

EVote: EVote is a product developed by Marilyn Davis [5], and works in conjunction with a mailing list. It allows the members of the mailing list to poll themselves. EVote is not adaptable to specific user needs and severely lacks security mechanisms. The automated processing of ballots is limited due to the syntactic freedom in designing email messages. Similar arguments hold for ScanVote, an automated paper based registration and tallying system for mail and on-site voting, TeleVote, a voting system using either voice or touch tone voting, and TouchVote, an automated on site voting system using touchscreen inputs.

Sensus [4] is a voting system implemented by Lorrie Cranor as a prototype to demonstrate a security-conscious electronic polling system. It uses a simple text based interface, but with a well developed security concept behind: it uses different encryption techniques and blind signatures. A particular flaw in the security concept is the assumption that the text messages sent from the voter to the system cannot be intercepted. (TEAMVOTE uses SSL (secure socket layer) to make sure that no such interception is possible.) EVOX [14] appears quite similar to Sensus with respect to the exploited security and encryption techniques.

Ventana's GroupSystems "Vote" and "Survey" [7] are decision making tools as part of the GroupSystems Software package. It lets a user set up a poll, gather the ballots and compute a result. It lacks accessibility as users are required to install a special client software to use it and the evaluation methods are not as covering and configurable as in TEAMVOTE. An ambitious approach to accessibility has been undertaken in the Web4Groups Project [15] providing a webinterface to a groupware system also including a plurality voting module.

1.2 Team Decisions in TEAMVOTE

As was seen from the above survey on electronic voting systems, both research prototypes and commercial products are still rather immature with respect to CCF coverage and usability features. As a consequence, TEAMVOTE besides a systematic adoption of results from collective choice theory, offers a variety of innovative features not found in comparable systems.

The main point of critique of existing electronic voting systems is the lack of different CCFs. Most systems offer just one CCF, assuming a general applicability in different choice matters, which, considering the discussion on the relative suitabilities of different CCFs can lead to severe misuse in team decisions. TEAMVOTE leaves the creator of a vote with the choice of selecting an appropriate CCF. Secondly, the set-up process of the poll is almost as important as the poll itself, as the creator of the poll needs to specify all the parameters. For this, TEAMVOTE provides highly customisable set-up procedures which can also be accessed through the Internet. Thirdly, to promote the agenda of a poll, it is important to deliver background information on the different alternatives involved in the poll: eligible voters should not have to search for this information themselves. To attach alternative related information to the ballot itself, TEAMVOTE offers the

option to add hyperlinks to the respective alternatives. This information is thus easily accessed by clicking on a button that spawns e.g. a new browser window displaying the requested page.

Another important aspect for voting in an organisation and especially voting in an opinion poll on the web is the ability to integrate all client user interfaces seamlessly into HTML pages. TEAMVOTE's user interfaces are all self resizing applets that can handle all voting activities via standard Web browsers. More than that, web designers can reuse these applets in any existing web site by simply plugging them into respective web pages. A dedicated Votecreator applet can be integrated in a page describing the different properties of the evaluation methods, etc.

With its automated e-mail services TEAMVOTE accelerates the announcement, voting and reminder process: e-mails notify voters on open polls, remind lazy voters and distribute poll results to eligible recipients. The few necessary but essential administrative functions like deleting a poll, reminding the voters to vote or to trigger the evaluation process are all executed at the server application in a secure (it allows only a person with access to the server machine to perform these actions) and user friendly way. Security in particular is handled by the transmission over a secure socket layer (SSL), which takes care of encryption, decryption and validation processes needed to secure the electronic poll.

TEAMVOTE also enables easy access to the results of a poll. Result notification happens via e-mail notification messages, as well as via the results applet viewable by any browser, so that poll results can be brought to a broad audience (over the Internet) as well.

2 General Description of Implementation

TEAMVOTE is a set of tools supporting team decisions in distributed environments. Users who want to take part in a poll or who want to set up a poll themselves have to register themselves to TEAMVOTE. The system allows to set up polls, where the creator has to determine different parameters of the poll. These parameters are: the name of the poll, the set of alternatives (possible with links to background information about the alternatives), the list of eligible voters and the desired CCF. After setup, eligible voters are notified by e-mail that a poll has been opened and that ballots can be dispatched.

TEAMVOTE is a client/server application where the client software can be displayed by any Java enabled Internet browser – and is, as such, platform independent. The server-module as the central program serves primarily as the database for the data of users and of the different polls currently open on that system. In addition, it provides processes for the application of different CCFs in the different polls. The server side of the TEAMVOTE itself is divided into two logical parts: the User_Registration module and the Vote module. The User_Registration is optional and handles all transactions related to the registration of users into the system. The Vote module handles all transactions dealing with polls (creating polls, dispatching ballots and computing the poll result).

2.1 Base Technologies Used in Implementation

As TEAMVOTE was designed for use over the Internet and as a client server architecture, the choice of the implementation environment was Java [8] [13], particularly the following enabling technologies:

Sockets: The Java language offers all the tools to connect applications via the Internet with its network sockets which can be used to transmit data using the TCP/IP protocol.

Streams: The use of streams, especially objectstreams make the exchange of data using the before mentioned sockets easy. Streams are the main I/O tool for Java and can be linked to any device (file, keyboard, network socket). Objectstreams are streams that transport whole objects.

Threads: As TEAMVOTE should be able to handle more than one request at a time, the usage of concurrently running processes is a necessity. Java offers a comfortable way to implement this using Java threads – a dedicated service thread is spawned by the server program for each connected client at runtime.

Synchronisation: Having threads executing concurrently is the source of synchronisation problems, if they are sharing the same data and manipulating the same data at the same time. By making every method that accesses shared data a “synchronized” method, Java itself guarantees mutually exclusive data access. This synchronisation is using a monitor concept, where every application accessing the data has to get hold of the monitor first, which allows just one accessing method at a time.

Applets: The Client side of TEAMVOTE was supposed to be platform-independent with unrestricted access from the net. Applets are small applications that are accessed on an http server, transmitted over the net, and automatically loaded and executed as a part of a web page. Once an applet is installed on the requesting client, it serves as a dynamic input form to TEAMVOTE.

AWT: All TEAMVOTE user interfaces are implemented using standard AWT components, guaranteeing a high degree of flexibility and customisation.

Security: All the relevant poll and user data is kept on the protected TEAMVOTE server. The transfer of user and ballot data is implemented using SSL (secure socket layer), protecting the data from corruption and attacks, but also verifying authenticity of the sender.

Consequently, TEAMVOTE can be run on any system with the latest version of JDK installed, together with an http server able to deliver applets to provide the users with the client GUI. For storing user and poll related data, a directory on the server machine must be provided. Clients who want to view the applets need Java-enabled browsers supporting the event model of the AWT GUI.

2.2 The TEAMVOTE Architecture

The main part of TEAMVOTE (see Fig. 1) is the server sided **Votelistener** which is the central control- and co-ordination component of the system. The second server sided application, the **Userreglistener** is used to maintain the database of potential or eligible voters. The

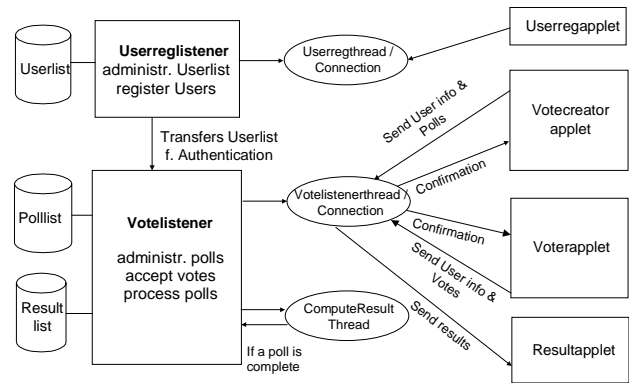


Fig.1 TEAMVOTE Architecture

Userreglistener is in charge of integrating new users in the **Userlist**. A user is an instance of a user object containing user name, password, e-mail address, a unique userid, etc., organised in a Java vector called **Userlist**. A vector is used for its flexibility as a dynamic data structure, able to hold a quasi-unlimited amount of objects, not requiring to specify the size of the vector at instantiation.

At startup, the **Userlist** vector is loaded from a file that is specified in a configuration file. For convenience the whole **Userlist** vector is saved using a Java objectstream, which allows simple saving and loading of objects to or from files. As the **Userreglistener** itself can only handle one request by a client at a time, the listener starts a thread called **Userregthread** for each incoming request. This thread handles a request until all transactions are complete and the thread is killed. The thread communicates with the clients through standard Java sockets using a port specified in the configuration file, and streams (objectstreams for input and printstreams for output) which consists mainly of message-strings, notifying the client that the transaction was complete or that an error occurred. The main purpose of **Userreglistener** is to add new users. It receives a new user object from the client, checks for existence and adds the user to the **Userlist**. Synchronisation of different registration threads is handled through the Java synchronisation mechanisms. The secondary purpose of the **Userreglistner** is to provide a complete list of eligible voters to the **Votecreator** client. The **Votecreator** needs this list for creating new polls, i.e. to select users are eligible to vote.

The **Votelistener** uses many of the concepts used in the **Userreglistener**. The main data structure of the **Votelistener** is **Polllist**, containing all active polls in the system. This list is also saved to disk for backup and is loaded when the system starts up. **Polllist** again is a Java vector holding individual instances of poll objects. A poll object is a data structure covering a unique poll identifier, the title of a poll, creator id, CCF id, the number of alternatives, a step counter for multi-step CCFs, the number of voters and vectors of all alternatives, voters and incoming ballots. The **Votelistener** also spawns a thread for each incoming request and uses the same mechanisms for synchronization as the **Userreglistener**. The communication is also based on sockets and streams using a

combination of objectstreams for the transfer of more complex objects and printstreams for the exchange of simple strings. The Votelistener has five main functions which are handled by the spawned threads: i) verification of a user's name and password, ii) entering a new poll object into the poll list vector, iii) sending a list of polls to a voter, iv) entering a single vote of a user into the appropriate poll object and adding it to the incoming votes vector, and v) sending a poll result list to a user upon request. The input validation is always handled by the client program so as to avoid sending an object with invalid content. Incoming ballots are objects that contain the id of the user, the id of the poll, an CCF and a string that describes the ballot of a voter. This string has three basic appearances corresponding to the three different input methods provided by the voter applet: i) Preference Relation: the string contains the id of the alternatives with the symbols ~ (indifferent) and > (strong preference) e.g.: 2>5~1>3>4 standing for alternative 2 strongly preferred over alternative 5 and 1, which are considered at the same preference level, but strongly preferred over alternative 3, etc. ii) to support approval voting: for each alternative the symbol "0" indicates disapproval, "1" indicates approval, e.g.: 1-0;2-1;3; iii) Borda score: for each alternative an integer indicates the score, e.g.: 1-4;2-1;3-0;4-2;5-3

Every time a poll is created or evaluated an e-mail thread is started sending out e-mails to all voter, notifying where and when to vote or telling them the result of the poll. This e-mail thread can also be called manually from the server application to remind users to vote. The e-mail thread uses a socket connection to communicate with a mail server and sends out standard text mails. Whenever an incoming vote is added to a poll object, it is checked if the poll is complete and the evaluation process can be started. This evaluation process is also a thread that does not influence the main Votelistener component (the thread contains the implementation of different CCFs mentioned before). If the poll used a single step CCF, a result object is created and the poll object is destroyed, otherwise a preliminary result is created and the poll object is generated according to a CCF of choice. The result created is a text stream describing the result of the poll. It can be viewed with the result browser client program.

The two main server based programs can be configured with a configuration file or parametrized upon call at the command. The configuration file is based on the properties concept of Java, easing a configuration valid for the scope of the whole application. The options of configuration concern the internet host (the server host of TEAMVOTE), Filewlist, the file where Polllist resides, Fileulist, the file holding Userlist, Filerlist, the file holding the results of a poll, Userport, the port Userregistrationserver is listening on, and Voteport, the port Votelistener is listening on.

3 The TEAMVOTE User Interface

All clients of the TEAMVOTE are standard Java applets, which can be displayed by any browser interpreting Java. The graphical user interface components are all standard awt (abstract window toolkit) components like textfields,

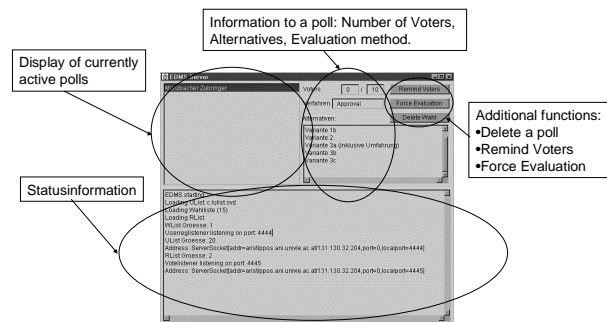


Fig.2 TEAMVOTE Server Interface

choice menus, radio buttons, buttons, etc. The interaction is implemented based on standard Java events. Most of the applets semantically validate the input asking for appropriate corrections in case of invalid user interaction. As an example, the Votecreator checks if there are at least two alternatives and at least three voters before it sends the poll object to the server. The server user interface uses the same GUI components as Votecreator, the only difference being that the server is a Java application, not an applet. The graphical user interfaces componts are linked to the main components of the TEAMVOTE: registration, votecreator, voter, resultbrowser and the TEAMVOTE server program.

Registration: New users submit their name and password for authentication via the registration applet. The provision of an e-mail address is also required since much information about upcoming polls or about results (at least the respective notifications) are distributed via e-mail.

Server Interface: The server interface (Figure 2) is concerned with the output of information on the current state of the server. Information about incoming ballots, serverports or evaluation threads is displayed in the large text-output window on the lower half of the applet. The window in the top left corner displays the titles of all the polls in the system that are not yet ready for evaluation. Clicking on one of the polls displays additional information to that poll in the windows in the top middle section. Here the number of voters who have already submitted their ballots and the number of total voters, the CCF used and the set of alternatives are displayed. The top right corner aggregates administrative services. The button "Remind Voters" generates e-mail messages to lazy voters and also reminds them of the location of the voter applet. The second button "Force Evaluation" starts the evaluation process for the selected poll, no matter how many outstanding votes are left. The last button "Delete Poll" deletes the poll from the system.

Poll Administration: The Votecreator (Figure 3) is a tool to administrate polls. The creator of a poll has to authenticate himself (name, password) in order to create the poll. A name has to be given to the poll before the creator can select from the set of CCFs (3 binary methods, 3 single step methods, 4 multi step methods) via radio

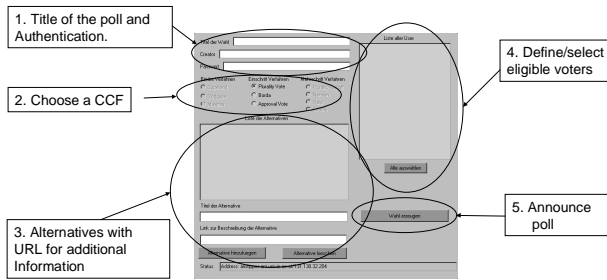


Fig.3 TEAMVOTE Poll Creator

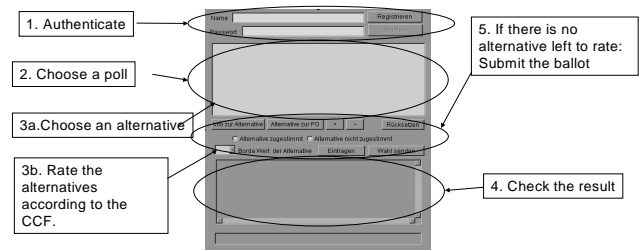


Fig.4 TEAMVOTE Ballot Applet

buttons. Eligible voters can be selected by clicking in the user list. The set of alternatives is provided by putting the name in the first field and an http-link (where to find related information to that alternative) in the second field of a list displayed in the large window. Alternatives can be deleted by the same simple means of direct manipulation of the list. The poll is finally created on the press of a button – related status information is displayed.

Submitting Ballots: The Voter-applet is the user interface for eligible voters to cast their vote (Figure 4). The user who wants to submit a ballot has to authenticate himself again by using the name and password input fields. All the polls a user is involved in are displayed in the large window in the upper half of the applet. After selecting a poll, the lower part of the applet (which has been invisible up until this action) is being displayed and the different alternatives are presented in the main window. Different input methods corresponding to different CCFs are being served: i) approval voting: the user decides for each alternative if he approves or disapproves it marking each alternative using the radio buttons - he has to approve at least one alternative, ii) Borda score: the user assigns each alternative a score ranging from 0 to the number of alternatives minus one, by choosing a score from a dropdown list – only one distinct value can be assigned to one alternative, iii) preference relation: the user specifies a full preference string relating the alternatives using the symbols “>” and “~”. For further information on related alternatives, the user can follow the links associated with each alternative, viewing this information via an automatically spawned web browser. Once the voter is decided with respect to the alternatives, the ballot is submitted via a "send" button.

Viewing Results: The TEAMVOTE result browser is a tool for viewing poll results according to the used CCF. The respective displays are rank list oriented in case of score based CCFs, and table oriented in case of full preference relation based CCFs .

4 Conclusions

Research in collective choice has revealed that there is no single group decision procedure favourable in all thinkable group decision matters. Every single method is flawed in one way or the other, either violating democratic principles, yielding paradoxical decisions or just being prone to strategic behaviour. After outlining the relative

strengths and weaknesses of intuitive group decision making policies, a systematic attempt has been made to implement the whole spectrum of collective choice procedures (ten CCFs) in an integrated, distributed environment, TEAMVOTE. The primary audience of TEAMVOTE are organisations, that need decision making support as a part of their meeting software or as a stand alone tool to administer their choice processes like in investment decisions, elections or survey polls. The implementation platform of TEAMVOTE being standard Internet technologies (HTML, Java) raises the perspective of its use also for marketing opinion polls, governmental elections, or to enable societal democracy processes on a broad scale. A challenging, yet very useful extension of TEAMVOTE will be the dynamic poll evaluation and "on-the-fly" presentation of poll results, so as to allow users in dedicated decision scenarios to follow the evolution of the choice process, and to add new alternatives as they appear throughout negotiations.

We wish to acknowledge particularly the latter inspiration to extend the functionality of TEAMVOTE to one of the anonymous reviewers. Overall, we wish to thank all the involved reviewers for their exceptional support.

5 References

- [1] Senn A. K., *Collective Choice and Social Welfare*, Holden-Day Inc., San Francisco, 1970.
- [2] Nurmi H., *Voting procedures: A summary analysis*, British Journal of Political Science, 13(2): 181-209, 1983.
- [3] Arrow K., *Social Choice and Individual Values*, Wiley, New York, 1951.
- [4] Cranor L. R. and Cytron R. K., *Sensus: A Security-Conscious Electronic Polling System for the Internet*. Proceedings of the Hawai'i International Conference on System Sciences, IEEE CS Press, 1997.
- [5] Davis M., Evote <http://www.ozemail.com.au/~jjjacq/evote>
- [6] Automated Election Administration <http://www.trueballot>.
- [7] Ventana Corp., GroupSystems <http://www.ventana.com>
- [8] Naughton P. and Schildt H., *The complete reference Java 1.1*, Second Edition. 1998.
- [9] Nurmi H., *Comparing Voting Systems*, D. Reidel Publishing Company, Dordrecht, 1987.
- [10] Brahm's St. J. and Fishburn P. C., *Approval Voting*, Birkhaus, Boston, 1983.
- [11] Saari D. G., *Geometry of Voting*, volume 3 of Studies in Economic Theory. Springer-Verlag, New York, 1994.
- [12] Java JCE 1.2 beta documentation <http://www.javasoft.com:81/security/JCE1.2/earlyaccess>
- [13] Campione M. and Walrath K., *The Java Tutorial*, Object Oriented Progr. for the Internet. Addison-Wesley, 1996.
- [14] <http://theory.lcs.mit.edu/~cis/voting/voting.html>
- [15] Web4Groups, <http://www.web4groups.at>.