The Liquid Democracy Journal

on electronic participation, collective moderation, and voting systems

Issue 4

Berlin, 2015-07-28

The Liquid Democracy Journal is dedicated to the idea of Liquid Democracy, which is a democratic principle that uses transitive delegations to unite the best of direct and representative democracy.

But this journal is not just limited to Liquid Democracy; it also covers those topics coming up when implementing it: **electronic particatipation, collective moderation, and voting systems.**

The Liquid Democracy Journal
on electronic participation, collective moderation, and voting systems

Issue 4, Berlin 2015-07-28 (electronic version 2015-12-14, rev2 2017-05-10)

Published by: Interaktive Demokratie e. V., Berlin, Germany

Edited by: Jan Behrens
Axel Kistner
Andreas Nitsche
Björn Swierczek

Contact editors at: editors@liquid-democracy-journal.org

For unsolicited sent-in works neither the editors nor the publisher take any responsibility.

Subscribe at: http://www.liquid-democracy-journal.org/

Archive available at: http://www.liquid-democracy-journal.org/

ISSN-L: 2198–9532
ISSN print version: 2198–9532
ISSN electronic version: 2199–1758

Editorial

by the Editors, Berlin, July 28, 2015

It has been almost six years since we published the first version of LiquidFeedback in the autumn of 2009. Since then, we've constantly been improving the software in regards to the user experience as well as considerations regarding the democratic process implemented by the software.

LiquidFeedback is not only an implementation of Liquid Democracy (the idea of transitive, revocable delegations by topic) but also features a unique proposition development system where huge groups of people may discuss and decide in a self-organized way. It has always been the goal to abstain from the need for a moderator or request commission, and LiquidFeedback has been successfully reaching that goal in real world scenarios with up to several thousand participants.

With this Issue 4, we want to go further: we will present an approach to make the LiquidFeedback proposition development and decision making process feasible for a potentially unlimited number of participants. The article "A Finite Discourse Space for an Infinite Number of Participants" is the missing piece of the puzzle to create a process that scales – in theory – for an infinitely high number of people.

But we will also present new application fields of the LiquidFeedback process. Björn Swierczek will present not only theoretical considerations but also an implementation to connect LiquidFeedback with revision control systems.

Considering the widespread use of revision control systems in many application fields, this paves the way for democratizing work processes even in areas where such a level of democratization has not been thinkable before (e.g. infrastructural knowledge bases such as the Wikipedia). His implementation, included in this issue, has already been successfully tested in the lab, and will soon find its way into the official LiquidFeedback release.

Last but not least, Issue 4 will also contain an addendum to the mathematical proof on preferential delegation that has been presented in

the last issue. We were surprised that, in between, Google Inc. released information about an internal Liquid Democracy experiment [GooglePaper] utilizing a preferential delegation system [GoogleVideo]. As shown by our proof, certain requirements cannot be fulfilled at the same time. This also applies to those experiments, resulting in negative voting weight in case of the given approach.

We would like to note that the proof regarding negative voting weight might be generalized further. Property 2 in the original proof [PD] might be removed from the list of contradicting properties if each case considers all possible partial permutations of voters. Since such formalization and generalization wouldn't change anything in regards to the practical impact of our findings, we will not provide a formal proof in this matter. Such a proof, however, might be of theoretical interest.

Dangerous misconceptions

Liquid Democracy has recently been getting more and more attention by researchers, politicians, and even companies. The experiment done by Google Inc. is one example for this development.

While we are generally happy about the popularity of Liquid Democracy, we are also concerned about the fact that some properties of electronic decision making are constantly ignored, as so in case of the paper [GooglePaper] describing Google's experiments.*

Hardt and Lopes propose in their paper that only those ballots that use delegated voting weight need to be public.*

They refer to this conception as the "Golden rule of Liquid Democracy". [GooglePaper, p.4] But not all that glitters is gold: it can easily be shown that this allows neither for a verification of the processes by the voters nor for casting votes secretly (i.e. Google Inc. could still know what everybody voted on). Therefore, the participants of the system would not have any way to check the results of the system; they would have to blindly trust the results.* It is obvious that such an approach doesn't meet democratic standards at all and thus should neither be called "Golden rule" nor "Democracy".

As this was the first experiment of Google Inc. with Liquid Democracy, there is still the chance to correct such aberrations before creating a product for the general market. But it will be a big challenge to provide a solution which can be legitimately trusted by the users. This would require a completely transparent system which is verifiable by the users: all ballots must be published by the system and any algorithm influencing the democratic process (including any used sorting algorithm) must be public as well. For us, this seems to be contradictory to the business goals of Google Inc. at least at the current time.

Nonetheless, we do not want to criticize Google for experimenting in that domain, but we must warn people of using the term "democracy" for systems that cannot meet democratic standards. We do not want to live in a future

where people are encouraged to vote "anonymously" while – in fact – a global corporation counts and records all votes. Cryptographic protocols are unable to provide a solution to this problem. [PLF, chapter 3]

If votes are being recorded, they must be recorded publicly such that all participants may verify the process, and all participants must be aware that they are not taking part in a secret ballot but in an open ballot. For all other democratic decisions (i.e. those democratic decisions that ought to be secret), we strongly suggest the use of physical ballot boxes instead of electronic systems.

Accreditation, accreditation, accreditation (and publication of the ballot data)

We predict that the greatest challenge for a further democratization of platforms like the Wikipedia will be a proper accreditation of the participants (the only way to properly ensure one vote per real person) along with creating an acceptance for publishing ballots. Without publishing each participant's ballot in an electronic voting system, verifiability for the participants cannot be achieved. (For further details, refer to chapter 3 in our book "The Principles of LiquidFeedback" [PLF].) In no case can non-verifiable systems be called "democratic", because systems which are not verifiable by the participants are never democratic.

We hope that the dream of a more self-organized and more democratic world soon becomes true, and that secret elections remain truly secret where they are needed.

The Editors

** Edited as of May 10, 2017. For details, refer to the errata file in the ZIP archive of this issue or the corrigendum on page 4 of Issue #5 of The Liquid Democracy Journal.*

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[PD] Jan Behrens & Björn Swierczek: Preferential Delegation and the Problem of Negative Voting Weight. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 3 (2015-01-23). ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[GooglePaper] Steve Hardt & Lia C. R. Lopes: "Google Votes: A Liquid Democracy Experiment on a Corporate Social Network", Technical Disclosure Commons, June 5, 2015. http://www.tdcommons.org/dpubs_series/79

[GoogleVideo] Steve Hardt: YouTube video "Liquid Democracy with Google Votes, Part 2", March 12, 2014, GoogleTechTalks. Uploaded to YouTube on June 27, 2015. https://www.youtube.com/watch?v=F4lkCECSBFw

Errata for Issue 2

Print version

The print version of Issue 2 contained an error in Figure 6 on page 26 (article "Dividing the Pie – Visualizing Quantities and Qualities of Majorities in Pie Charts"). The unit (360°) was missing in the formula depicted in Figure 6. Instead of "α_{pie}" read "α_{pie} / 360°". This error has already been fixed in the first publication of the electronic version (revision 1, 2014-11-29) of Issue 2.

Electronic version

In the first publication of the electronic version (revision 1, 2014-11-29) of Issue 2, the HTML version of the article "Dividing the Pie – Visualizing Quantities and Qualities of Majorities in Pie Charts" contained an error in the caption of Figure 6. The second line of the caption was accidentally omitted. The full caption can be found in the PDF version and reads: "Figure 6: The formula to calculate the pie rotation in case of supermajorities (q is the required supermajority, and 7/12 is an arbitrary value > 1/2 but ≈ 1/2 to keep the 'no' block mostly left)".

In the first publication of the electronic version (revision 1, 2014-11-29) of Issue 2, a hyphen was missing in the introductory text to Figure 9 ("Once upon a time…") of the HTML version of the article "Game of Democracy". In lines 8 and 9 of the text above Figure 9 read "kingdom" instead of "king dom".

These two errors have been fixed in the electronic version, revision 2 (2015-07-27) of Issue 2.

Read in this Issue:

Democratic File Revision Control with LiquidFeedback

by Björn Swierczek, Berlin, July 28, 2015

I. Abstract

In this paper, it is shown how a project team can democratically decide on incorporating changes of files held in a repository managed by a revision control system by extending LiquidFeedback and its proposition development and decision making process.

LiquidFeedback [LF] is an open source software for proposition development and decision making published by the Public Software Group e. V., Berlin, Germany [PSG]. [PLF, p.13] Since 2010 the software LiquidFeedback is used by political parties, non-governmental organizations, regional governments, and companies for opinion formation, binding decision making, and citizen petitions.

A revision control system is the standard way to track changes of the source code of a computer software developed by multiple authors. This technique is also used for many other purposes, like tracking changes to documents, books, datasets, product data, or any other kind of information. Different revision control systems are available on the market and used widely. Often used systems are Git [Git], Mercurial [Mercurial], and Subversion [Subversion]. Other software is incorporating aspects of revision control systems to track changes while providing other functionalities, e.g. Wiki systems like MediaWiki [MediaWiki] used by Wikipedia [Wikipedia].

But these systems lack support for collective decisions on incorporating changesets. In most implementations, either a user has write privileges or not. Therefore, in larger projects, especially projects with more than a few dozen active contributors, only a small number of people regularly have final control over the files held in the repository. They are moderating the process of applying changes and finally deciding if a change set will be included or not.

To overcome these limitations and to get rid of the need of privileged moderators, it is shown that it is possible to extend and use LiquidFeedback in such a way that the process of

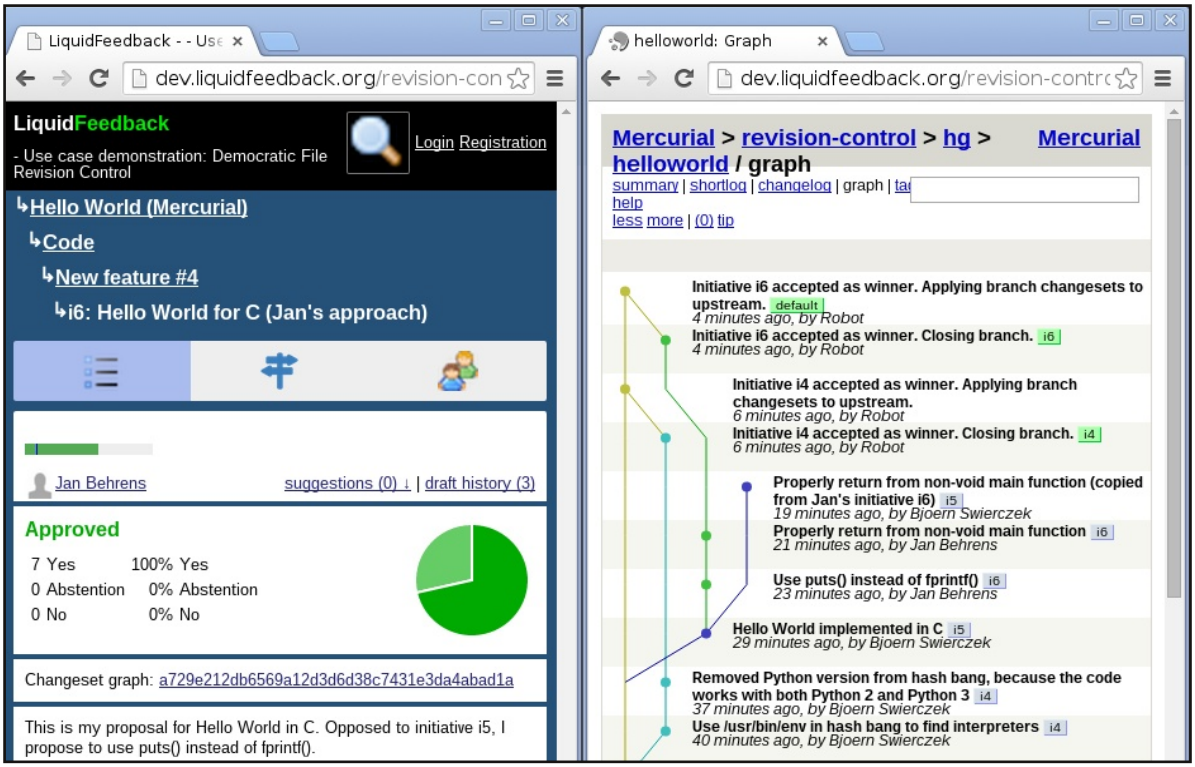


Figure 1: Screenshots of LiquidFeedback (mobile view) and HgWeb serving a Mercurial repository (graph view)

changing the files managed by a revision control system can be organized collectively by the members of the project team using LiquidFeedback's proposition development and decision making process. Furthermore, it will be shown that these concepts can also be adopted by many other systems which are utilizing revision control systems.

Through implementing a proof of concept, it is demonstrated that it is technically possible to organize democratic decision making on incorporating changes to files held in a repository. Therefore, the decision if a project makes use of privileged moderators or implements a democratic decision process is not a technical question anymore, but an organizational or political one. This opens new application fields for democratic processes in the context of revision control and at the same time new application fields for revision control systems in the context of democratic processes. It is also possible to create completely new application fields by using the synergistic effects created by utilizing LiquidFeedback with revision control systems.

As this paper addresses different scientific fields, the basic functionality of LiquidFeedback and revision control systems are explained in the following sections II and III.

II. Introduction to LiquidFeedback's Proposition Development and Decision Making Process

LiquidFeedback is an open source software, which can be installed on an internet server and used through a web browser. LiquidFeedback offers a unique proposition development and decision making process whose structure and main features are described in the following:

II.1. Organizational unit (short form: unit)

The units are the highest hierachical structure of LiquidFeedback, intended to represent organizational units, like national, regional and local chapters. Units are organized as tree to organize subsidiary chapters below their superior chapters. Voting privileges are given to users per unit without inheritance to subsidiary or superior units. [PLF, p.158]

II.2. Subject area (short form: area)

Every unit has one or more subject areas, holding the possible issues together in groups of similar topics, e.g. finances, public relations, different fields of politics, etc. A subject area belongs to a unit. [PLF, section 4.8] [PLF, p.165]

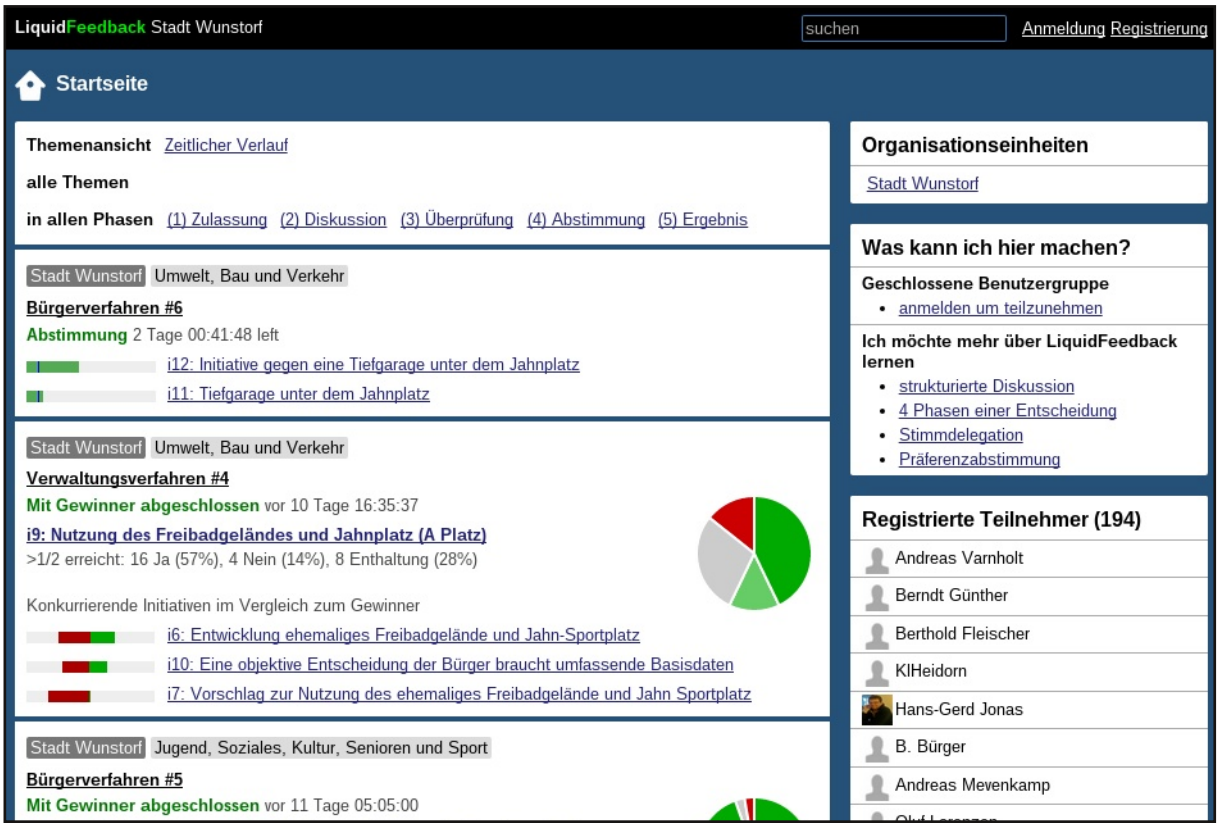


Figure 2: Screenshot showing LiquidFeedback used for civic participation by the City of Wunstorf [Wunstorf]

II.3. Issue

An issue is a group of competing initiatives going together through the four phases of a decision in LiquidFeedback. An issue is automatically created when a new initiative is started and not placed into an existing issue. Each issue is identified by a unique number, which is automatically assigned when it is created. Per issue, not more than one initiative can be accepted as winner in the end. An issue belongs to a subject area. [PLF, section 4.4] [PLF, p.147] [PLF, section 4.8]

II.4. Initiative

The initiative is the main form to express a will in LiquidFeedback. It can consist of a proposal and/or reasons for it and/or reasons against other competing initiatives. Initiatives can be supported by users, changed until the verification phase begins, and finally be voted upon in the voting phase. An initiative belongs to an issue. [PLF, subsection 4.1.1] [PLF, section 4.4]

II.5. Draft

A draft is a version of an initiative. Every time an initiator changes the content of an initiative, a new draft is created. Old drafts are saved for future reference. [PLF, subsection 4.1.1]

II.6. Initiator

The initiator is the user who created an initiative. Only an initiator can change the content of an initiative during discussion. The initiator can invite other users as initiator, which gain the same rights as the original initiator after accepting the invitation. This includes the right to grant or revoke initiator privileges to/from another initiator of the same initiative. [PLF, subsection 4.1.1] [PLF, p.154]

II.7. Supporter

A supporter is a user supporting an initiative, helping it to fulfill the quora measured at the end of the admission and verification phases. Users which have rated a suggestion as “must” but “not fulfilled”, or “must not” but “fulfilled” are counted as potential supporters, supporting the initiative only under the requirements expressed in the rated suggestions. [PLF, (sub)sections 4.1.1, 4.1.2, 4.3, 4.6]

II.8. Suggestion

Suggestions are placed by users to propose improvements for initiatives. This can range from a simple typo or grammar correction to complex changes of the initiative. All supporters of an initiative can rate suggestions to let the initiator(s) know about the collective opinion and how they could improve the initiative. Suggestions can be rated as “must”, “should”, “should not”, “must not” and whether they are “implemented” or “not implemented”.

If a suggestion will be implemented or not is the decision of the initiator(s) only. But if a widely demanded suggestion is not implemented in the initative, any user can start an alternative competing initiative implementing this suggestion (similiar to “forking” a software project). [PLF, subsection 4.1.2]

II.9. Policy (rules of procedure)

A policy (also referred to as "rules of procedure") is a set of configuration settings, including:

- how long the four phases of a decision should last,
- which quora are applied at the end of the admission and verification phases, and
- which majorities are needed in the voting phase to become accepted as winner.

The initiator of an initiative, which is not placed in an existing issue, can choose the policy to use for the newly created issue. As there is no computable way to check if the correct policy is chosen, it is up to the users not to support issues which are misusing policies. [PLF, subsection 4.7]

II.10. Predictable timing of four phases

An issue in LiquidFeedback is going through four phases: [PLF, section 4.6]

- admission phase,
- discussion phase,
- verification phase, and
- voting phase.

The duration of the four phases depends on the settings of the chosen policy. Therefore the timing can be predicted. [PLF, section 4.5]

II.11. Using quora to moderate the process

To moderate the overall process and to filter out issues and initiatives which do not have enough support, a quorum needs to be passed after the admission phase and before the voting phase. How many supporters are needed to let an issue or initiative pass is configured in the chosen policy. [PLF, section 4.3] [PLF, section 4.7]

II.12. Transitive delegated voting (Liquid Democracy)

The basic idea of Liquid Democracy is a democratic system in which issues are decided by direct referendum, but votes can be dynamically delegated by topic as not every participant has time and personal knowledge about every issue. Implementing this idea allows also to dynamically find experts for specific subject areas and issues in a democratic and traceable way. Other terms referring to the same idea are "Delegated Voting" and "Proxy Voting". [PLF, chapter 2]

II.13. Minority Protection with the Harmonic Weighting algorithm for a fair share of display space

Even though any democratic decision that has at least one dissentient vote leads to a overruled minority, it is still possible to protect minorities in democratic processes. The most important measures of protecting minorities are unalienable, constitutional rights, which are granted in most democracies. But this cannot be ensured algorithmically and is therefore out of scope of computer software. [PLF, section 4.10]

Another form of minority protection is to give minorities a fair chance to promote their positions for discussion in the democratic process. [PLF, section 4.10]

LiquidFeedback implements this form of minority protection: minorities are given the right to promote their positions. Technically there is no limit in the number of issues to be discussed in an online system since many issues can be handled simultaneously (unlike "offline" conventions, where usually only one issue can be discussed at a time). But there is another limit of online systems: the display is limited and can only present a small amount of information at the same time and users can only absorb a limited amount of information per time. To ensure a fair share of this limited display space, LiquidFeedback implements the Harmonic Weighting algorithm, which proportionally shares the available display space between all initiatives in such a way that minorities can put their issues and initiatives into the debate while noisy minorities (e.g. so-called internet trolls) are not able to harm other minorities by allocating an unfair share of display space by placing a large amount of initiatives. [PLF, section 4.10] [Evolution]

II.14. Preferential voting avoiding tactical behavior

LiquidFeedback utilizes a modern preferential voting system for the final decision in the voting phase of an issue which is based on the Schulze Method (sometimes referred to as Cloneproof Schwartz Sequential Dropping). The Schulze Method fulfills certain criteria which are desired for democratic processes, e.g. [Schulze] [PLF, section 4.14]

- Independence of Clones,
- Monotonicity,
- Schwartz Criterion, and
- Independence of Smith-dominated Alternatives (ISDA or Smith-IIA).

While fulfilling several further properties, the Schulze Method is implemented in LiquidFeedback with a robust tie breaking system, solving situations the Schulze Method cannot solve alone. [Schulze] [TieBreaker] The most notable property of the Schulze Method is to lessen incentives for tactical voting behavior. [PLF, section 4.14]

II.15. Further process and implementation details

The LiquidFeedback proposition development and decision making process and it's implementation in LiquidFeedback Core [Core] and LiquidFeedback Frontend [Frontend] utilize further concepts to provide a scalable way of collective proposition development and decision making and is actively advanced regarding theory and practice by the Public Software Group e. V. [PSG] and Interaktive Demokratie e. V. [IAD], both in Berlin, Germany. [LF] [PLF] [LDJournal] [liquidfeedback.org]

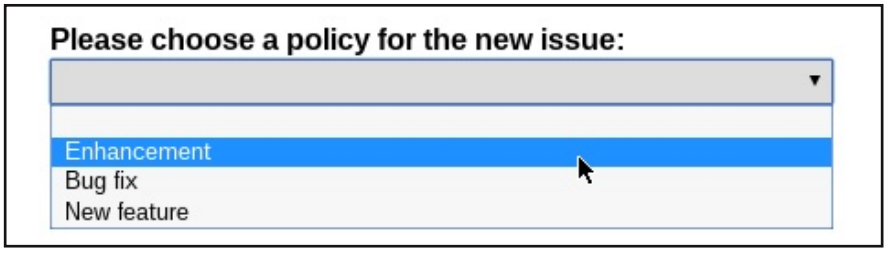


Figure 3: Screenshot of choosing a policy in LiquidFeedback's frontend

III. Introduction to Revision Control Systems

A revision control system is a computer software to manage different versions of data files. Especially it is used to track changes on text files like source code, configuration files, documentation, articles, books, but also to track changes on product data, i.e. construction data for cars, airplanes and other products.

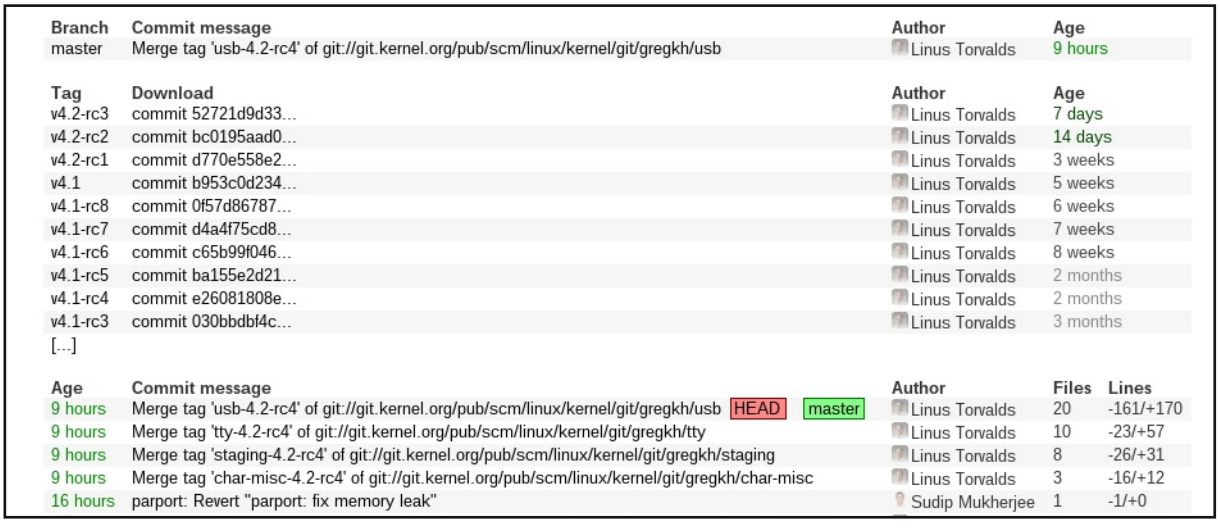
With a revision control system, so-called "repositories" can be created, which hold different versions of files and track their changes. A bundle of changes to files in the repository is often called changeset. An important feature of revision control systems beneath the tracking of changes is the ability to go back to any past revision of a file or any previous changeset, as long as it is stored in the repository.

Nowadays revision control systems use the internet to allow groups of creators working together from different places of the world while tracking each changeset back to its originator. This enables groups of creators to collectively work on source code, articles, or other data while minimizing communication overhead.

Most revision control systems are offering the possibility to create "branches", i.e. giving a name to a series of changesets which are not (yet) part of the official main branch. The official or main branch of a project is often referred to as trunk or master branch.

To incorporate changesets made in branches into the trunk or master branch, most revision control systems have a merge command, which allows merging changes.

Aspects of revision control systems are also incorporated in other computer software for tracking changes, e.g. in Wiki systems like MediaWiki used by the Wikipedia. [MediaWiki] [Wikipedia]



Branch	Commit message	Author	Age
master	Merge tag 'usb-4.2-rc4' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/usb	Linus Torvalds	9 hours

Tag	Download	Author	Age
v4.2-rc3	commit 52721d9d33...	Linus Torvalds	7 days
v4.2-rc2	commit bc0195aad0...	Linus Torvalds	14 days
v4.2-rc1	commit d770e558e2...	Linus Torvalds	3 weeks
v4.1	commit b953c0d234...	Linus Torvalds	5 weeks
v4.1-rc8	commit 0f57d86787...	Linus Torvalds	6 weeks
v4.1-rc7	commit d4a4f75cd8...	Linus Torvalds	7 weeks
v4.1-rc6	commit c65b99f046...	Linus Torvalds	8 weeks
v4.1-rc5	commit ba155e2d21...	Linus Torvalds	2 months
v4.1-rc4	commit e26081808e...	Linus Torvalds	2 months
v4.1-rc3	commit 030bbdbf4c...	Linus Torvalds	3 months

[...]

Age	Commit message	Author	Files	Lines
9 hours	Merge tag 'usb-4.2-rc4' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/usb HEAD master	Linus Torvalds	20	-161/+170
9 hours	Merge tag 'tty-4.2-rc4' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/tty	Linus Torvalds	10	-23/+57
9 hours	Merge tag 'staging-4.2-rc4' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging	Linus Torvalds	8	-26/+31
9 hours	Merge tag 'char-misc-4.2-rc4' of git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/char-misc	Linus Torvalds	3	-16/+12
16 hours	parport: Revert "parport: fix memory leak"	Sudip Mukherjee	1	-1/+0

Figure 4: Screenshot of the Git repository of Linus Torvalds' sources for the Linux Kernel [Torvalds]

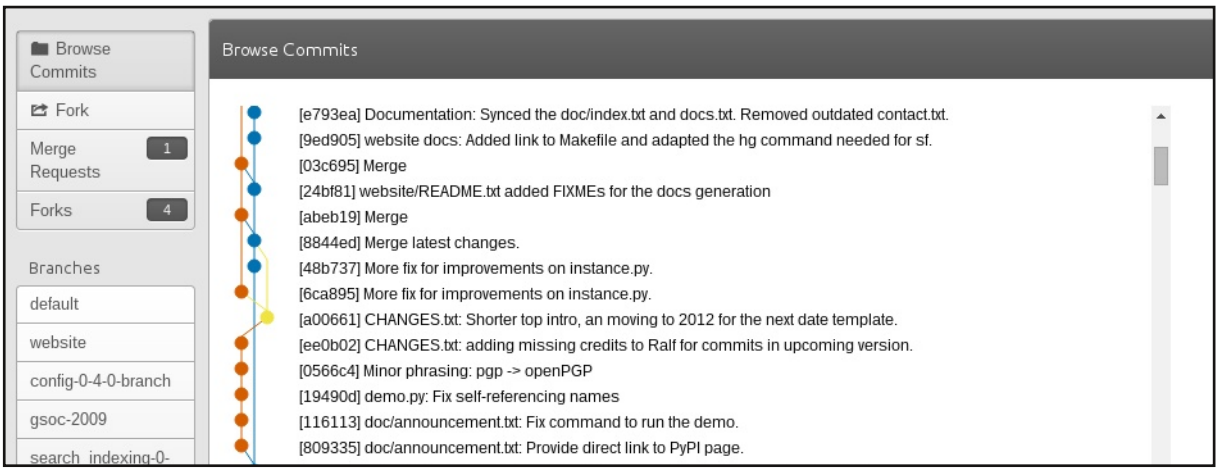


Figure 5: Screenshot of the Mercurial repository of the Roundup Issue Tracker as seen on SourceForge [Roundup]

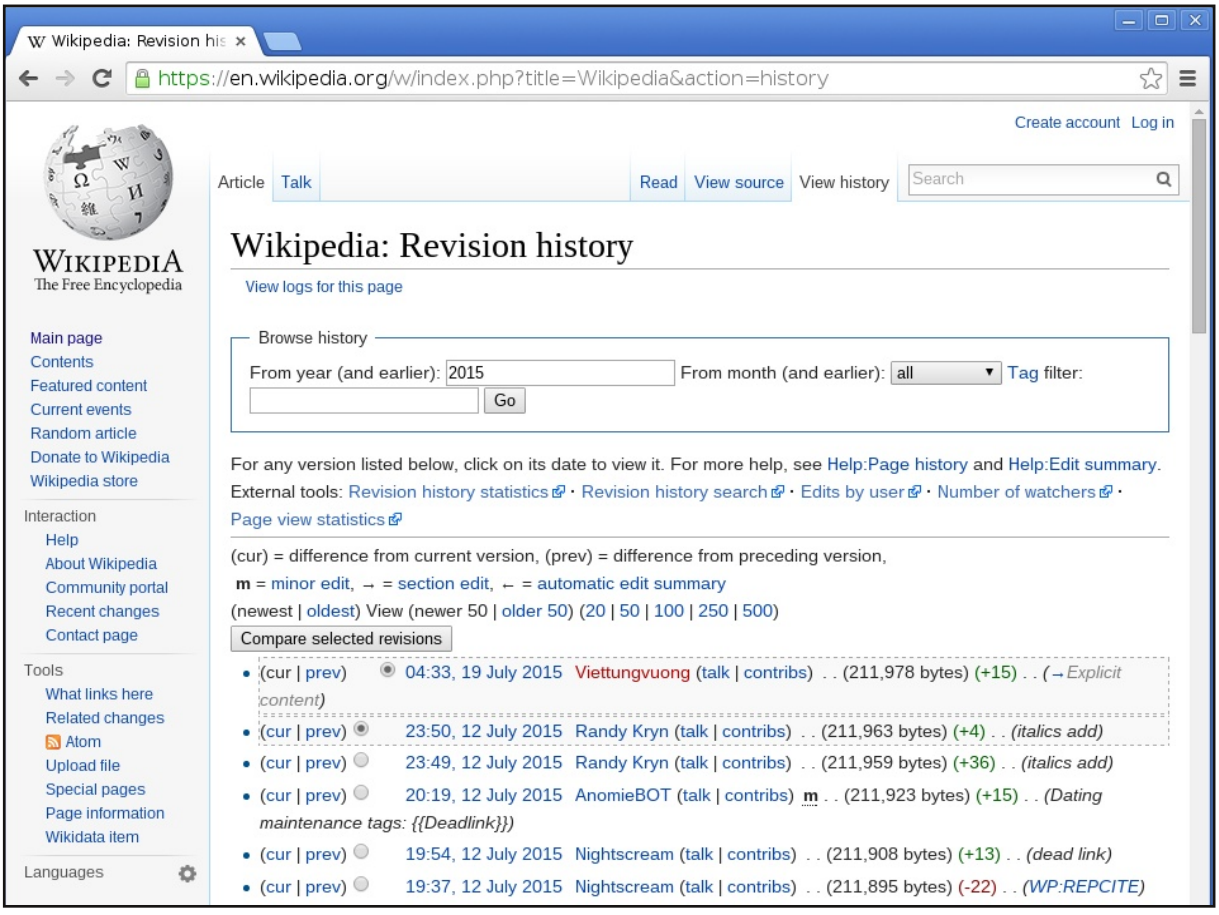


Figure 6: Screenshot of the revision history of the Wikipedia article on Wikipedia itself [Wikipedia2]

IV. Extending LiquidFeedback for use with a revision control system

In this section, it is shown how LiquidFeedback can be extended to collectively decide on merging changesets to the trunk or master branch of a repository.

IV.1. Terms

The terms used by LiquidFeedback are created for generic democratic decisions. For using the LiquidFeedback process in the context of revision control systems, I propose a mapping of the terms as follows:

Unit -> Repository	*Initiative -> Branch*
Area -> Module	*Draft -> Changeset*
Issue -> Issue	*Suggestion -> Suggestion*

IV.2. The basic idea

The basic idea is that changes to the files held in the trunk of a repository require a formal decision by the project team (or another empowered group of persons) using the LiquidFeedback proposition development and decision making process.

To achieve this functionality, it is proposed to extend LiquidFeedback in such a way that each initiative represents a branch in the repository and each draft represents a changeset of this branch. Even while I suggest to map "initiatives" to "branches" for this specific purpose, in fact they will still be LiquidFeedback initiatives, which need to go successfully through all four phases of the LiquidFeedback proposition development and decision making process to become accepted.

During these phases, the members of the project team (or other empowered persons) can use all regular functionalities of LiquidFeedback to debate and decide about the branch. At the same time, all regular functionalities of the revision control system can be used with one exception: branches with names possibly referencing an LiquidFeedback initiative (i.e. having a name in a certain format, e.g. "i123" to reference the initiative with the ID 123) can only be committed to the repository if an initiative with the referenced ID exists and is still in admission or discussion phase. Furthermore the user committing the branch needs to be initiator of that initiative. This also allows to merge changesets created in other branches to one's own branch, i.e. one's own initiative.

To enhance a branch, suggestions can be made unless the verification phase has already begun. The suggestion may include additional changesets to be added by the initiator(s) if they like to incorporate them into their branch.

Any member of the project team (or other empowered persons) can also create a competing branch by creating an alternative LiquidFeedback initiative in the same issue unless the voting phase has already begun.

After the voting phase of an issue ends, it is proposed to merge that branch to the project's trunk which has been declared winner of the issue, if any.

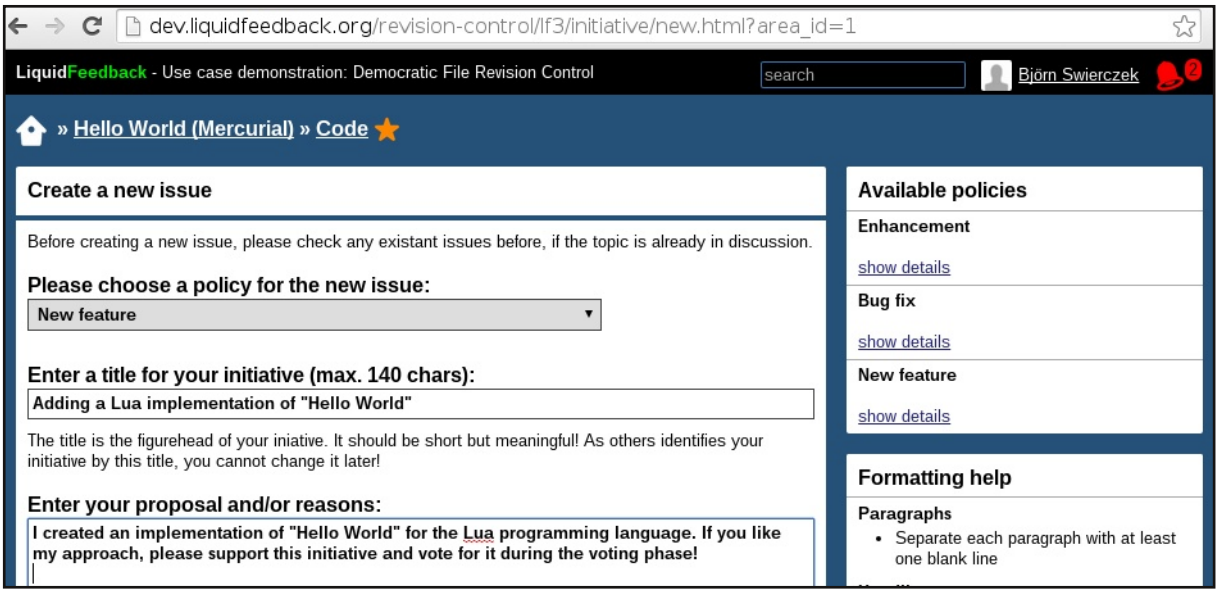


Figure 7: Screenshot showing creation of a new initiative in LiquidFeedback

```
[user@client ~]$ hg clone http://dev.liquidfeedback.org/revision-control/hg/helloworld
[...]
requesting all changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
[user@client ~]$ cd helloworld/
[user@client ~/helloworld]$ hg branch i1
marked working directory as branch i1
(branches are permanent and global, did you want a bookmark?)
[user@client ~/helloworld]$ vi helloworld.lua
[user@client ~/helloworld]$ hg add helloworld.lua
[user@client ~/helloworld]$ hg commit -m 'First version of helloworld.lua'
[user@client ~/helloworld]$ hg push --new-branch
pushing to http://dev.liquidfeedback.org/revision-control/hg/helloworld
[...]
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 1 changes to 1 files
remote: [lf4rcs] inspecting changesets
remote: [lf4rcs] checking branch i1
remote: [lf4rcs] adding node 31eac3c1bc07ce76c498d89512d6ed2c0981c7f4 to initiative i1
remote: [lf4rcs] changes cleared. continue committing.
```

Figure 8: Protocol of a terminal session adding a repository changeset to a LiquidFeedback initiative

The policies (rules of procedure) framework of LiquidFeedback can be used to allow different quora, timings and majorities for different types of branches, i.e. for bug fixes, enhancements, documentation, etc. to reflect the different needs of different types of decisions.

IV.3. Concurrency of decisions

A special problem seems to be the fact that other changesets may have been already applied to the trunk or master branch between proposing a changeset the first time and the decision after the four phases of LiquidFeedback.

Using different timings, it is also possible that a changeset is applied before another changeset although it has been started after the other changeset's LiquidFeedback initiative already reached the verification or voting phase. Therefore, it is not possible to reflect changes made by the "faster" changeset anymore. This can lead to a merge conflict, i.e. two changesets changing the same part of a file. Thus a proposed changeset can break at any time, so it would not be possible to apply the changeset at the current time.

But more important would be the answer to the question: "Would the changeset still apply without merge conflicts after its decision and after other changesets are applied to the trunk or master branch?"

IV.3.a. Predict future changesets

We cannot predict which changesets will be accepted in the end. But at any time we could make useful assumptions based on the approval rates (supporter counts) made during the first three phases of LiquidFeedback. It could be assumed that any branch which has been accepted for discussion and has an approval rate of more than 50% (and being the initiative with the highest supporter count in its issue) will be accepted as winner later. Based on this assumption, we can try to apply the associated changesets in the order of the predictable end of the voting phase and check if they cleanly apply. If any changeset would not be (hypothetically) applicable anymore, the initiators or interested users could be informed appropriately.

This prediction system could be extended to a more sophisticated system, which traces different paths in parallel.

IV.3.b. Exclusions and requirements

Another mechanism to solve the problem of concurrent decisions could work as follows: LiquidFeedback could be extended to allow the initiator(s) to add two lists of references to other branches, which are expected to be decided previously. An entry in one list indicates that the current branch should *not* be applied if at least one of the referenced branches is accepted (exclusion) while all branches referenced in the other list need to be accepted (requirement).

As soon as an excluded branch is accepted or a required branch is declined, the branch should become automatically revoked unless the initiative is still in admission or discussion phase

and the initiator(s) could still make appropriate changes healing the branch. For the purpose of automatic revocation, a new finished state for LiquidFeedback issues could be introduced: "Canceled because of unfulfilled precondition".

This system could be extended by a more sophisticated way to describe the preconditions, i.e. nested sets of exclusions and requirements combined with logical "AND"/"OR" operations.

IV.3.c. Let the users decide

A very easy way to solve problems related to concurrency is to simply ignore merge conflicts algorithmically but let the users handle them. For example, after the changesets associated with a LiquidFeedback initiative cannot be merged due to conflicting changesets, any user could start another initiative and post a changeset which fixes this conflict. As soon as this initiative is accepted as winner, this fixing changeset will be incorporated together with the changesets of the initiative which couldn't be merged previously. For this purpose, a special policy with an appropriately short discussion and decision time could be configured in LiquidFeedback.

IV.3.d. Merging 2nd winner if 1st winner fails

In situations where changesets associated with a LiquidFeedback initiative which has been declared winner cannot be merged due to other conflicting changesets which already have been merged, one could come up with the idea to try merging the 2nd winner (or the 3rd if the 2nd fails too, and so on), based on the motto "you had your chance". But this would not be wise, as it could create an unwanted feedback to the behavior of users. A user who is obsessed about an initiative could try to create short lasting initiatives (e.g. with a bug fix or documentation policy), or modify already running initiatives, to intentionally break a promising initiative. This attempt could be easily hidden, e.g. as typo fix or an enhancement of documentation. If such an attempt is successful, a situation could arise where an initiative being the first winner is not applicable anymore, but the second one (the one the user is obsessed about) is. As this would be unfair, only first winners should be merged.

IV.4. Accreditation of users

Any democratic decision making system needs a proper accreditation to ensure that every eligible person (e.g. developers of a software project) get exactly one account (and therefore one vote) and nobody else. This condition is also valid for LiquidFeedback and therefore also for using LiquidFeedback with revision control systems. [PLF, subsection 6.1.1]

IV.5. Verifiability by the users

Democratic decisions need to be verifiable. It is impossible to make electronic votings secret and at the same time verifiable by the participants. Therefore, the only way to implement electronic decisions verifiable by the participants are open recorded votes. [PLF, chapter 3]

V. Technical Implementation

V.1. Environment

To use LiquidFeedback and one or more revision control systems on an internet server, it is necessary to setup the following open source software packages according to the setup instructions provided by the distribution and package maintainers:

- Linux or BSD (operating system) [Debian] [ArchLinux] [FreeBSD],
- HTTP web server supporting the CGI interface [Apache] [Lighttpd],
- PostgreSQL (database server), version 9.1 (or higher) [PostgreSQL],
- LiquidFeedback Core, version 3.0.5 [Core],
- Lua (programming language), version 5.2 or 5.3 [Lua],
- Moonbridge Network Application Server for Lua, version 1.0.1 [Moonbrige],
- WebMCP (web application framework), version 2.0.2 [WebMCP],
- Markdown2 [Markdown2],
- LiquidFeedback Frontend, version 3.0.9 [Frontend],
- one or more revision control systems, e.g.
  - Git (distributed version control system) [Git], including
    - Gitweb [GitWeb] and
    - git-http-backend [git-http-backend],
  - Mercurial (distributed source control management system) [Mercurial], including
    - HgWeb [HgWeb],
  - other revision control systems.

V.2. Configuring the HTTP web server to authenticate users against LiquidFeedback

The web server needs to be set up in such a way that users need to log in with a user name and password (via HTTP access control). To allow authentication with the username and password used in LiquidFeedback (single sign-on, SSO), a htaccess file as used by many HTTP web servers to store user authentication data can be created using the following shell command, which should be run regularly to reflect changes of the LiquidFeedback user database in the web server's authentication file (<name of LiquidFeedback database> needs to be replaced by the name of the LiquidFeedback database in PostgreSQL):

```
echo "SELECT login || ':' || password …
FROM member WHERE NOT locked;" | psql …
<name of LiquidFeedback database> -A -t > …
/etc/lighttpd/htpasswd.new && mv …
/etc/lighttpd/htpasswd.new …
/etc/lighttpd/htpasswd
```

V.3. Configuration of LiquidFeedback

V.3.a. Setup for controlling revision control systems

For LiquidFeedback, I assume the following setup:

- The LiquidFeedback web service provided by LiquidFeedback Frontend is available at: http://liquidfeedback/lf3

The ellipsis symbol … indicates that the current line of source code is wrapped for the print layout and continues at the next line.

- The accreditation of the team members (or other empowered persons) as LiquidFeedback members is finished and done properly. [PLF, subsection 6.1.1]

V.3.b. Setup for a project

For any project, a unit needs to be created in the administration interface of LiquidFeedback Frontend. The revision control system and the repository used by this project need to be configured in the external reference field of the unit created. The format of the configuration is

```
<repository type> <path on file system> …
<web url of repository>
```

V.4. LiquidFeedback Extension for Revision Control Systems (lf4rcs)

To allow LiquidFeedback (and therefore its users by democratic decision) to actually control one or multiple revision control systems, I implemented an extension for LiquidFeedback. This extension implements a modular framework which can be configured to be used with revision control systems. The extension consists of three parts:

- controlling and tracking commits to a repository,
- handling finished issues and winner initiatives, and
- formatting of web links to changesets.

The following source code needs to be loaded by the configuration of LiquidFeedback Frontend:

See Source code 1 (on following pages)

It is necessary to create /srv/commithook.lua with the following content:

See Source code 2 (below)

… and to make it executable by the operating system:

```
chmod +x /srv/commithook.lua
```

```
#!/usr/local/bin/lua

assert(loadfile("/srv/webmcp/framework/bin/mcp.lua"))(
  "/srv/webmcp/framework",
  "/srv/liquid_feedback_frontend",
  "main",
  "revision-control"
)

local arg1, arg2, arg3 = ...

lf4rcs.update_references(arg1, arg2, arg3)
```

Source code 2: Lua code of a generic commit hook handler for use by revision control systems

```
_G["lf4rcs"] = {}

lf4rcs.config = {}
lf4rcs.log_prefix = "[lf4rcs] "

function lf4rcs.exec(...)
  local output, err_message, exit_code = extos.pfilter(nil, ...)
  local command_parts = {...}
  for i, part in ipairs(command_parts) do
    if string.match(part, " ") then
      command_parts[i] = '"' .. part .. '"'
    end
  end
  local command = table.concat(command_parts, " ")
  return command, output, err_message, exit_code
end

function lf4rcs.get_config(unit)
  if not unit.external_reference then
    error("Unit is not configured for lf4rcs")
  end
  local repository, path, url = string.match(
    unit.external_reference, "([^ ]+) ([^ ]+) (.*)"
  )
  return repository, path, url
end

function lf4rcs.commit(issue)
  local repository, path, url = lf4rcs.get_config(issue.area.unit)
  if not (lf4rcs.config[repository] and lf4rcs.config[repository].commit) then
    error("Unsupported repository type")
  end
  local initiatives = Initiative:new_selector()
    :add_where{ "issue_id = ?", issue.id }
    :exec()
  for i, initiative in ipairs(initiatives) do
    local function exec(...)
      local command, output, err_message, exit_code = lf4rcs.exec(...)
      local log = "Executed: " .. command .. "\n"
      if output then
        log = log .. output .. "\n"
      end
      if err_message and #err_message > 0 then
        log = log .. "ERROR: " .. err_message .. "\n"
      end
      if exit_code and exit_code ~= 0 then
        log = log .. "Exit code: " .. tostring(exit_code) .. "\n"
      end
      issue.admin_notice = (issue.admin_notice or "") .. log
      issue:save()
    end
    local close_message, merge_message
    if initiative.winner then
      close_message = "Initiative i" .. initiative.id
```

Source Code 1: Lua code of the LiquidFeedback extension lf4rcs (part 1 of 4)

```
                      .. " accepted as winner. Closing branch."
      merge_message = "Initiative i" .. initiative.id
                      .. " accepted as winner. Applying branch changesets to upstream."
    else
      close_message = "Initiative i" .. initiative.id .. " rejected. Closing branch."
    end
    local target_node_id = initiative.current_draft.external_reference
    if target_node_id then
      local branch = "i" .. initiative.id
      lf4rcs.config[repository].commit(
        path, exec, branch, target_node_id, close_message, merge_message
      )
    end
  end
end

function lf4rcs.render_draft_reference(draft, wrapper)
  local repository, path, url = lf4rcs.get_config(draft.initiative.issue.area.unit)
  if not (lf4rcs.config[repository] and lf4rcs.config[repository].render_draft_reference)
then
    error("Unsupported repository type")
  end
  if draft.external_reference then
    wrapper(function()
      lf4rcs.config[repository].render_draft_reference(url, draft)
    end)
  end
end

function lf4rcs.render_initiative_reference(initiative, wrapper)
  if initiative.current_draft.external_reference then
    config.render_external_reference.draft(initiative.current_draft, wrapper)
  end
end

function lf4rcs.update_references(repository, path, unit_id)
  local function log(message)
    print(lf4rcs.log_prefix .. message)
  end
  if not lf4rcs.config[repository]
      or not lf4rcs.config[repository].get_remote_user
      or not lf4rcs.config[repository].get_branches
  then
    log("Unsupported repository type")
    os.exit(1)
  end
  log("inspecting changesets")
  local remote_user = lf4rcs.config[repository].get_remote_user()
  local function abort(message)
    log("TEST FAILED: " .. message)
    log("ABORTING and ROLLBACK due to failed test.")
    db:query("ROLLBACK")
    os.exit(1)
  end
```

Source Code 1: Lua code of the LiquidFeedback extension lf4rcs (part 2 of 4)

```
db:query("BEGIN")
local member = Member:new_selector()
  :add_where{ "login = ?", remote_user }
  :optional_object_mode()
  :exec()
if not member then
  abort(
    "internal error, member '"
    .. remote_user .. "' not found in database"
  )
end
local function exec(...)
  local command, output, err_message, exit_code = lf4rcs.exec(...)
  if not output then
    log("Could not execute: " .. command)
    abort(err_message)
  end
  if exit_code ~= 0 then
    log("Could not execute: " .. command)
    abort("Exit code: " .. tostring(exit_code))
  end
  return output
end
if lf4rcs.config[repository].extra_checks then
  local success, err_message = lf4rcs.config[repository].extra_checks(path, exec)
  if not success then
    abort(err_message)
  end
end
local branches, err = lf4rcs.config[repository].get_branches(path, exec)
if not branches then abort(err) end
for branch, head_node_ids in pairs(branches) do
  log('checking branch ' .. branch)
  if branch ~= lf4rcs.config[repository].working_branch_name then
    local initiative_id = string.match(branch, "^i([0-9]+)$")
    if not initiative_id
        or initiative_id ~= tostring(tonumber(initiative_id))
    then
      abort("this branch name is not allowed")
    end
    initiative_id = tonumber(initiative_id)
    if #head_node_ids > 1 then
      abort("number of heads found for branch is greater than 1: " .. #head_node_ids)
    end
    local initiative = Initiative:by_id(initiative_id)
    if not initiative then
      abort("initiative i" .. initiative_id .. " not found" )
    end
    if initiative.issue.area.unit_id ~= tonumber(unit_id) then
      abort("initiative belongs to another unit (unit ID " ..
                          initiative.issue.area.unit_id .. ")")
    end
    if
      initiative.issue.state ~= "admission" and initiative.issue.state ~= "discussion"
```

Source Code 1: Lua code of the LiquidFeedback extension lf4rcs (part 3 of 4)

```
      then
        abort("issue is already frozen or closed (" .. initiative.issue.state .. ")")
      end
      if initiative.revoked then
        abort("initiative has been revoked")
      end
      local initiator = Initiator:by_pk(initiative.id, member.id)
      if not initiator then
        abort("member is not initiator of initiative i" .. initiative_id)
      end
      if not initiator.accepted then
        abort(
          "member has not accepted invitation to become initiator of initiative i"
          .. initiative_id
        )
      end
      local node_id = head_node_ids[1] or false
      if node_id then
        log("adding node " .. node_id .. " to initiative i" .. initiative_id)
      else
        log("removing node reference from initiative i" .. initiative_id)
      end
      Draft:update_content(member.id, initiative_id, nil, nil, node_id)
    end
  end
  log("changes cleared. continue committing.")
  db:query("COMMIT")
  os.exit(0)
end

function lf4rcs.notification_handler(event)
  if event.event == "issue_state_changed" and (
      event.state ~= "admission" and
      event.state ~= "discussion" and
      event.state ~= "verification" and
      event.state ~= "voting"
  ) then
    lf4rcs.commit(event.issue)
  end
end

function lf4rcs.init()
  local super_handler = config.notification_handler_func
  config.notification_handler_func = function(event)
    if super_handler then super_handler(event) end
    lf4rcs.notification_handler(event)
  end
  config.render_external_reference = {
    draft = lf4rcs.render_draft_reference,
    initiative = lf4rcs.render_initiative_reference
  }
end
```

Source Code 1: Lua code of the LiquidFeedback extension lf4rcs (part 4 of 4)

V.5. Configuration for controlling Git

V.5.a. Setup for revision control by decisions made in LiquidFeedback

The following configuration is assumed for the project specific setup in the following subsection b:

- The root of the repositories is /srv/http/git
- The repository directories are owned by the web server operating system user
- The repository directories can be read and written by web server operating system user
- The directory /srv/git can be read and written by web server operating system user
- git-http-backend is serving repositories at the base address http://liquidfeedback/git/
- Gitweb is serving repositories at the base address http://liquidfeedback/gitweb/

The following source code needs to be loaded by the configuration of LiquidFeedback Frontend after lf4rcs has been loaded:

See Source Code 3.1

V.5.b. Setup for a project

For any project, a bare git repository needs to be created:

```
git init --bare …
/srv/http/git/helloworld.git
```

A call of the lf4rcs commit hook needs to be added to the hooks of the git repository (assuming the corresponding LiquidFeedback unit ID is 1):

See Source Code 3.2

Finally a clone of the repository has to be created:

```
cd /srv/git/
git clone /srv/http/git/helloworld.git
```

The configuration stored as external reference of the corresponding LiquidFeedback unit needs to be set to:

```
git /srv/git/helloworld …
http://liquidfeedback/gitweb/helloworld.git
```

```
[www@server ~]$ git init --bare /srv/http/git/helloworld.git
Initialized empty Git repository in /srv/http/git/helloworld.git/
[www@server ~]$ cat > /srv/http/git/helloworld.git/hooks/pre-receive << EOF
> #!/usr/bin/bash
> /srv/commithook.lua git /srv/http/git/helloworld.git 1
> EOF
[www@server ~]$ chmod +x /srv/http/git/helloworld.git/hooks/pre-receive
[www@server ~]$ cd /srv/git
[www@server /srv/git]$ git clone /srv/http/git/helloworld.git
Cloning into 'helloworld'...
warning: You appear to have cloned an empty repository.
done.
[www@server /srv/git]$
```

Figure 9: Protocol of a terminal session preparing a Git repository

```
lf4rcs.config.git = {

  render_draft_reference = function(url, draft)
    if not draft.external_reference then return end
    ui.tag{ content = _"Changeset:" }
    slot.put(" ")
    ui.link{
      text = draft.external_reference,
      external = url .. ";a=commit;h=" .. draft.external_reference
    }
  end,

  get_remote_user = function()
    return os.getenv("REMOTE_USER")
  end,

  get_branches = function(path, exec)
    local branches = {}
    for line in io.lines() do
     local oldrev, newrev, branch = string.match(line, "([^ ]+) ([^ ]+) refs/heads/(.+)")
      if not branch then
        return nil, "unexpected format from git hook environment"
      end
      branches[branch] = { newrev }
    end
    return branches
  end,

  commit = function(path, exec, branch, target_node_id, close_message, merge_message)
    if merge_message then
      exec("git", "-C", path, "checkout", "master")
      exec("git", "-C", path, "merge", target_node_id, "-m", merge_message)
      exec("git", "-C", path, "push", "origin", "master")
    end
  end

}
```

Source code 3.1: Configuration of lf4rcs for Git

```
cat > /srv/http/git/helloworld.git/hooks/pre-receive << EOF
#!/usr/bin/bash
/srv/commithook.lua git /srv/http/git/helloworld.git 1
EOF
chmod +x /srv/http/git/helloworld.git/hooks/pre-receive
```

Source code 3.2: Shell code to generate a commit hook handler for a Git repository

```
lf4rcs.config.hg = {
  working_branch_name = "work",
  render_draft_reference = function(url, draft)
    if not draft.external_reference then return end
    ui.tag{ content = _"Changeset graph:" }
    slot.put(" ")
    ui.link{
      text = draft.external_reference,
      external = url .. "/graph/" .. draft.external_reference
    }
  end,
  get_remote_user = function() return os.getenv("REMOTE_USER") end,
  get_branches = function(path, exec)
    local first_node_id = os.getenv("HG_NODE")
    if not first_node_id then return nil, "internal error, no first node ID available" end
    local hg_log = exec(
      "hg", "log", "-R", path, "-r", first_node_id .. ":", "--template", "{branches}\n"
    )
    local branches = {}
    for branch in hg_log:gmatch("(.-)\n") do
      if branch == "" then branch = "default" end
      if not branches[branch] then
        branches[branch] = {}
        local head_lines = exec(
          "hg", "heads", "-R", path, "--template", "{node}\n", branch
        )
        for node_id in string.gmatch(head_lines, "[^\n]+") do
          table.insert(branches[branch], node_id)
        end
      end
    end
    return branches
  end,
  extra_checks = function(path, exec)
    local result = exec("hg", "heads", "-t", "-c")
    for branch in string.gmatch(result, "[^\n]+") do
      if branch == lf4rcs.config.hg.working_branch_name then
        return nil, "open head found for branch " .. lf4rcs.config.hg.working_branch_name
      end
    end
    return true
  end,
  commit = function(path, exec, branch, target_node_id, close_message, merge_message)
    exec("hg", "up", "-R", path, "-C", "-r", target_node_id)
    exec("hg", "commit", "-R", path, "--close-branch", "-m", close_message)
    if merge_message then
      exec("hg", "up", "-R", path, "-C", "-r", "default")
      exec("hg", "merge", "-R", path, "-r", "tip")
      exec("hg", "commit", "-R", path, "-m", merge_message)
    end
  end
}
```

Source code 4.1: Configration of lf4rcs for Mercurial

V.6. Configuration for controlling Mercurial

V.6.a. Setup for revision control by decisions made in LiquidFeedback

The following configuration is assumed for the project specific setup of the following subsection b:

- The root of the repositories is /srv/hg
- The repository directories are owned by the web server operating system user
- The repository directories can be read and written by web server operating system user
- HgWeb is serving repositories at the base address http://liquidfeedback/hgweb/

The following source code needs to be loaded by the configuration of LiquidFeedback Frontend after lf4rcs has been loaded:

See Source code 4.1

V.6.b. Setup for a project

For any project, a mercurial repository needs to be created:

```
hg init /srv/hg/helloworld
```

A call of the lf4rcs commit hook needs to be added to the hooks of the Mercurial repository (assuming the corresponding LiquidFeedback unit ID is 2):

See Source code 4.2

The configuration stored as external reference of the corresponding LiquidFeedback unit needs to be set to:

```
hg /srv/hg/helloworld …
http://liquidfeedback/hgweb/helloworld
```

V.7. Configuration for controlling other revision control systems

An implementation for any other revision control system can be carried out similarly to the previously described setup using Git and/or Mercurial, as long as the revision control system supports:

- named branches,
- a merge facility,
- and a hook which allows external control over commits to a repository.

```
cat >> /srv/hg/helloworld/.hg/hgrc << EOF
[hooks]
pretxnchangegroup = /srv/commithook.lua hg /srv/hg/helloworld 2
EOF
```

Source code 4.2: Shell code to generate a commit hook handler for a Mercurial repository

V.8. Using the system

It is assumed, that the users of the repository have already checked out a local copy of the repository.

V.8.a. Creating an initiative in LiquidFeedback

To commit changesets to a repository controlled by LiquidFeedback as described before, a user must first create an initiative in the unit corresponding to the repository:

- open the appropriate area in the LiquidFeedback unit corresponding to the repository for which the changeset is intended,
- start a new initiative,
- enter a meaningful title for the changes,
- enter a draft describing the changes and reasoning them,
- actually create the initiative by publishing it.

V.8.b. Committing changesets

To associate changesets to the initiative created in LiquidFeedback, a user must mark them appropriately with a branch name in the following format, where <ID of initiative> is to be replaced with the numeric ID of the initiative:

```
i<ID of initiative>
```

As long as this initiative is in admission or discussion phase and the user is still initiator of the initiative, the user is allowed to push further changesets for this branch.

V.8.c. Committing changesets using Git

The user needs to mark the leading head of the changesets to be associated with the LiquidFeedback initiative as Git branch with the corresponding name, e.g. i123 for the initiative with the ID 123.

```
git branch i123
git checkout i123
```

The user can switch to different branches (associated with different LiquidFeedback initiatives):

```
git checkout i78
```

The user may push changes to the server repository, as long as all initiatives affected by the push request are still in admission or discussion phase:

```
git push origin
```

V.8.d. Committing changesets using Mercurial

The user needs to mark all changesets to be associated with the LiquidFeedback initiative as Mercurial branch with the corresponding name, e.g. i123 for the initiative with the ID 123.

```
hg branch i123
```

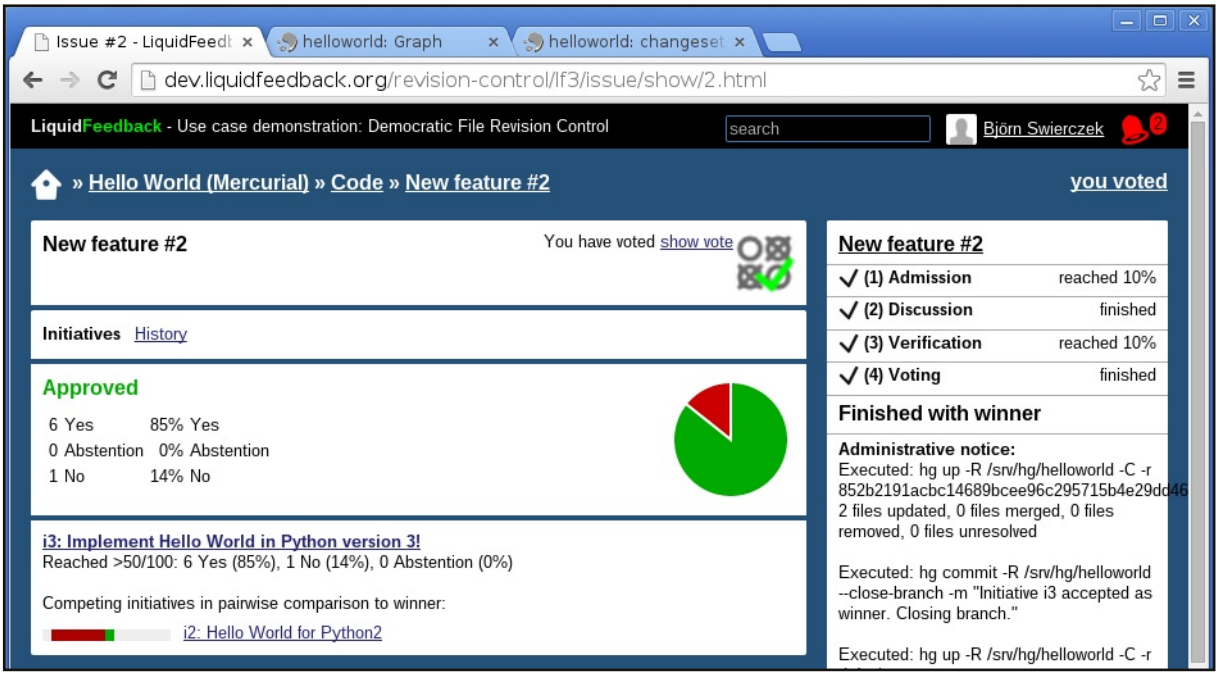


Figure 10: Screenshot of initiative "i3" winning over initiative "i2" in LiquidFeedback's frontend

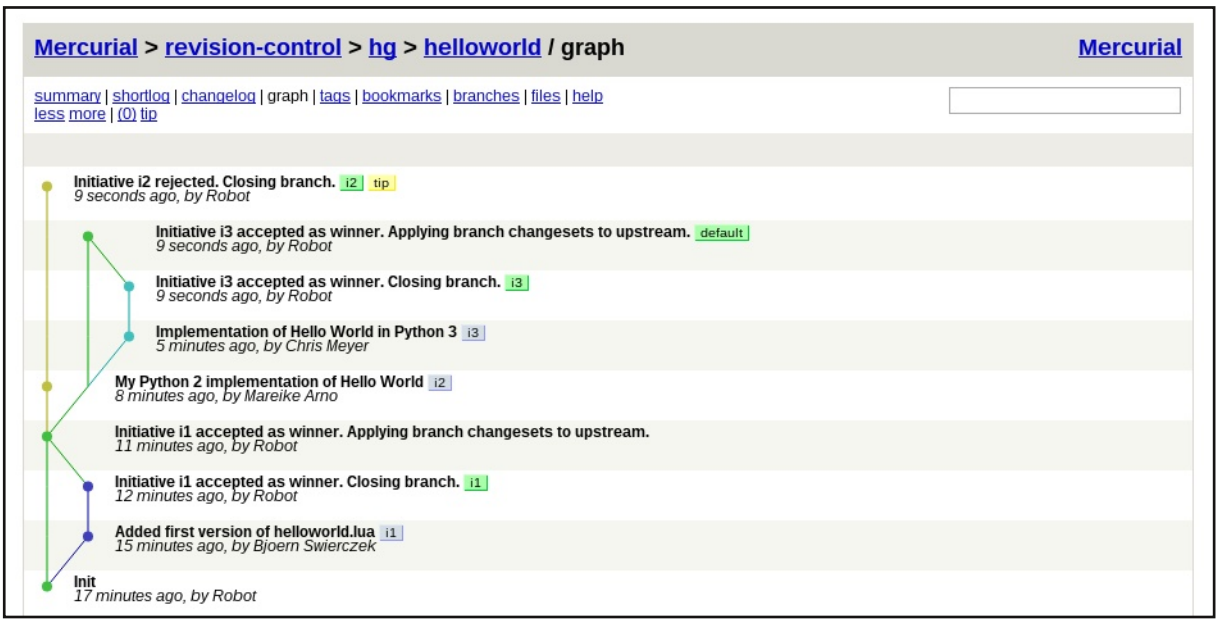


Figure 11: Screenshot of HgWeb, where branch "i3" is merged with the "default" (master) branch

Figure 12: Screenshot of initiative "i6" in discussion with competing initiative "i5" in LiquidFeedback's frontend

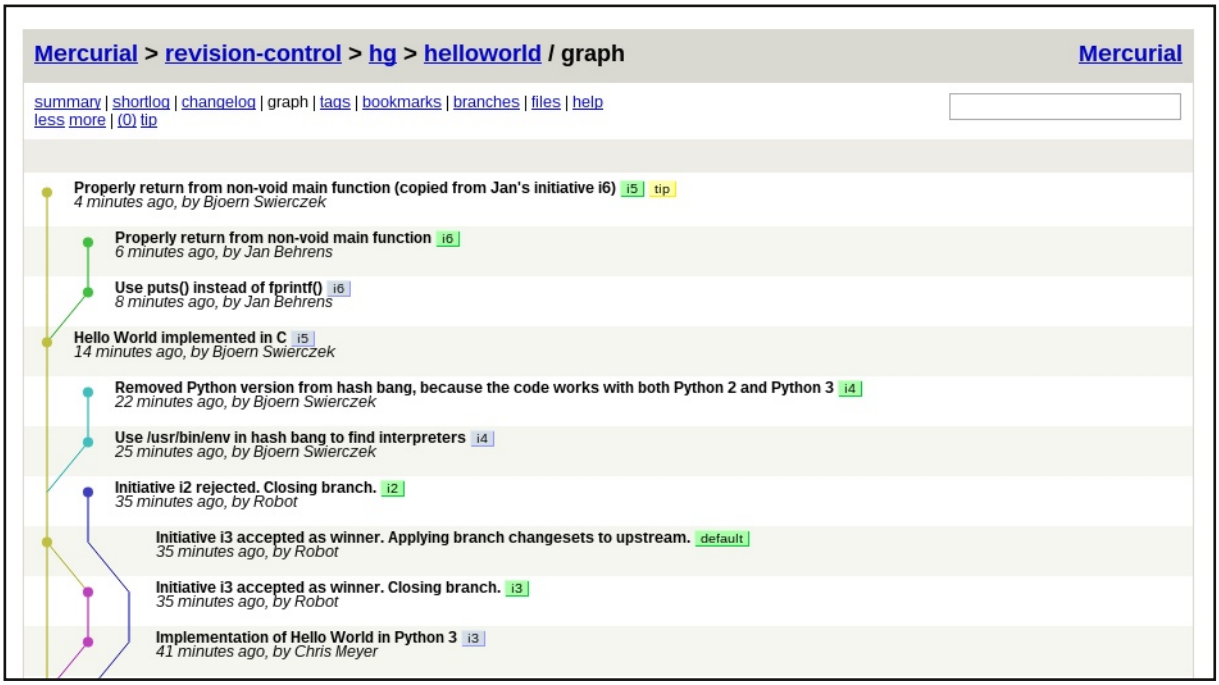


Figure 13: Screenshot of HgWeb, where "i6" has been forked from "i5"

Figure 14: Screenshot of HgWeb, where branch "i6" has been merged with the "default" (master) branch (along with the concurrent branch "i4")

The user can switch to different branches (associated with different LiquidFeedback initiatives):

```
hg up i78
```

The user may push changes to the server repository, as long as all initiatives affected by the push request are still in admission or discussion phase:

```
hg push --new-branch
```

V.8.e. Committing changesets using other revision control systems

For other revision control systems, the features corresponding to the features described for Git and Mercurial above should be used to accomplish these tasks.

V.8.f. Providing links to changesets associated with an initiative

For any LiquidFeedback initiative which has associated changesets, links to the corresponding views of the revision control system's frontend will be provided in LiquidFeedback to allow direct access by the user.

V.8.g. Commit changesets associated with winning initiatives

As soon as an initiative has been declared as winner by LiquidFeedback, the corresponding changesets will be applied automatically to the trunk or master branch of the repository if it is applicable without merge conflicts.

V.9. A cherry on top: adding Wiki functionality

It is possible to build a Wiki functionality on top of the setup described in the previous subsections. To demonstrate this, a repository may hold a file per Wiki page containing the content of this page (formatted with Markdown2). The following source code can then be run after each merge of a branch with the trunk or master branch of the repository:

See Source code 5.1

Furthermore, a HTML template file needs to be placed in the root of the repository. In this file, the place to put the rendered content of each page has to be marked with the string "$content$" in a separate line:

See Source code 5.2

The source code of the Wiki engine must be executed as follows, assuming it is placed in the directory /srv/ (<repository path> has to be replaced by the path to the repository containing the Markdown2 formatted files while <target path> needs to be replaced by a path to store the created HTML pages, which could be served by an HTTP web server):

```
/srv/wikify.lua <repository path> …
<target path>
```

Unusual for a Wiki, editing of pages is carried out by editing files and committing the changesets to a repository. But this approach is only a proof of concept and could be extended

to create an integrated (democratic) Wiki user interface. It would also be possible to use other parser engines to support different types of formatting languages. Even the integration of a comfortable WYSIWYG text editor is thinkable.

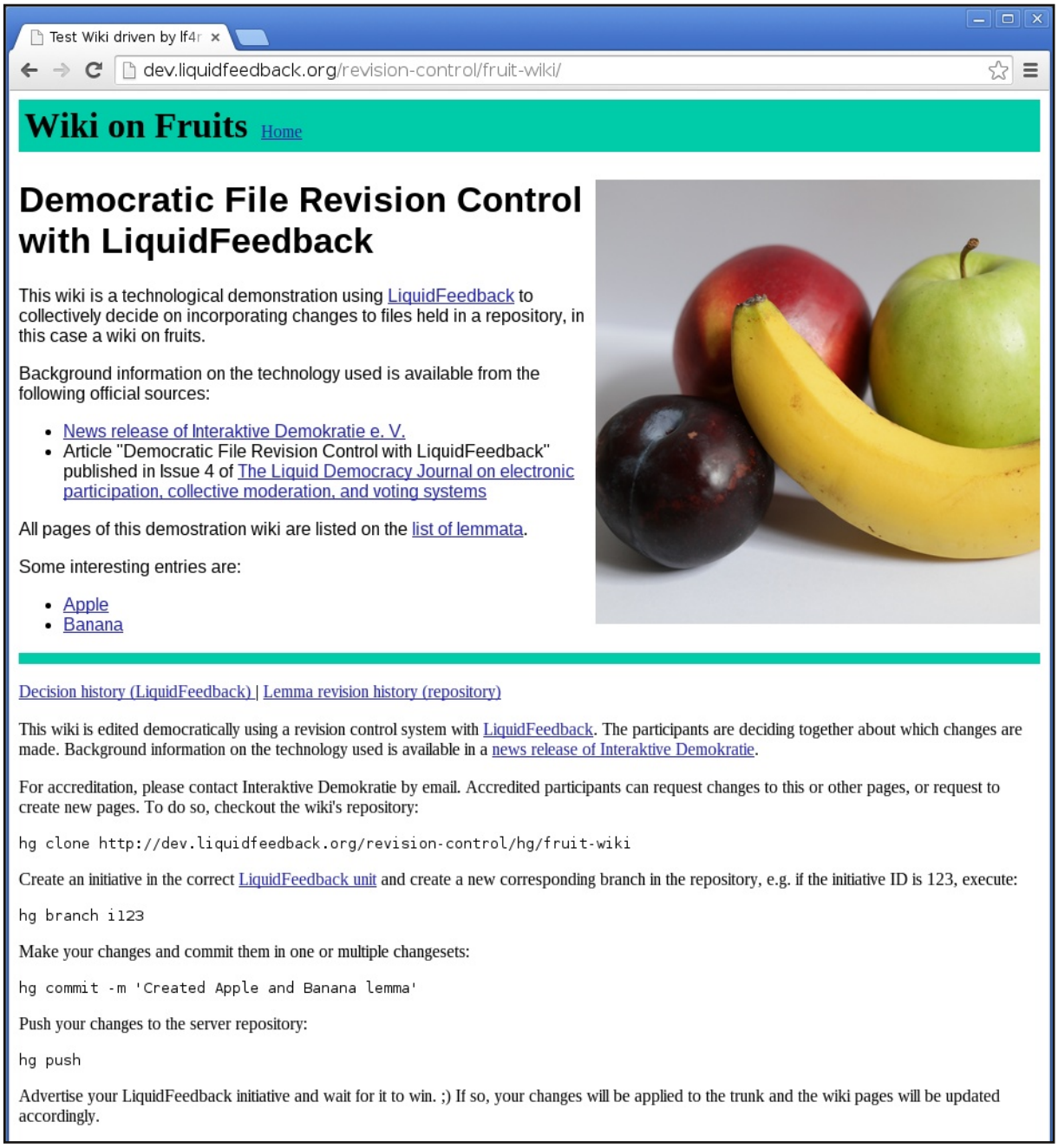


Figure 15: Screenshot of an example built with the described wiki functionality

```
#!/usr/bin/lua

local wikiparser = "/usr/bin/markdown2 -s escape"
local source_dir, target_dir = ...

local pages = {}
os.execute("rm -f " .. target_dir .. "/*.html")
local template_fh = io.open(source_dir .. "/template.html")
local template = template_fh:read("*a")
template_fh:close()
local function render(content, basename)
  local tmp = template:gsub("(\r?\n) *($content$) *(\r?\n)", function(m1, m2, m3)
    return m1 .. content .. m3
  end)
  if basename then
    tmp = tmp:gsub("$page%$", basename)
  end
  return tmp
end
for filename in io.popen("ls " .. source_dir .. "/lemma"):lines() do
  print("Processing " .. filename)
  local basename = string.match(filename, "^([a-zA-Z0-9_.-]+)%.markdown$")
  if not basename then
    print("ignoring " .. filename)
  else
    table.insert(pages, basename)
    local content_fh = io.popen(wikiparser .. " " .. source_dir .. "/lemma/" .. filename)
    local content = content_fh:read("*a")
    content = content:gsub("%[([a-zA-Z0-9_.-]+)%]", '<a href="%1.html">%1</a>')
    content_fh:close()
    local page_fh = io.open(target_dir .. "/" .. basename .. ".html", "w")
    page_fh:write(render(content, basename))
    page_fh:close()
  end
end
local index_parts = {}
table.insert(index_parts, "<h2>List of lemmata</h2>")
table.insert(index_parts, "<ul>")
for i, page in ipairs(pages) do
  table.insert(index_parts, '<li><a href="' .. page .. '.html">' .. page .. '</a></li>')
end
table.insert(index_parts, "</ul>")
local index = table.concat(index_parts)
local index_fh = io.open(target_dir .. "/all_lemmata.html", "w")
index_fh:write(render(index))
index_fh:close()
os.execute("rm -f " .. target_dir .. "/files/*")
for filename in io.popen("ls " .. source_dir .. "/files"):lines() do
  if string.match(filename, "^([a-zA-Z0-9_.-]+)$") then
    os.execute(
      "cp " .. source_dir .. "/files/" .. filename .. " " .. target_dir .. "/files/"
    )
  end
end
```

Source code 5.1: A minimalistic wiki engine for use with lf4rcs

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Wiki driven by lf4rcs using LiquidFeedback and a repository</title>
    <style>
      .head { background-color: #0ca; padding: 5px; }
      .head .logo { font-size: 200%; font-weight: bold; }
      .content { font-family: sans-serif; margin-top: 3ex; }
      .edit { border-top: solid 10px #0ca; clear: right; }
      img[alt=w400] { width: 400px; float: right; }
    </style>
  </head>
  <body>
    <div class="head">
      <span class="logo">Wiki on Fruits</span> <a href="index.html">Home</a>
    </div>
    <div class="content">
      $content$
    </div>
    <div class="edit">
      <p><a href="/revision-control/lf3/unit/show/1.html?mode=timeline">
           Decision history (LiquidFeedback)
         </a> |
         <a href="/revision-control/hg/fruit-wiki/log/tip/$page$">
           Lemma revision history (repository)
         </a></p>
      <p>This wiki is edited democratically using a revision control system with
         <a href="http://liquidfeedback.org/">LiquidFeedback</a>.
           The participants are deciding together about which changes are made.
           Background information on the technology used is available in a
         <a href="http://www.interaktive-demokratie.org/XXX/">
           news release of Interaktive Demokratie</a>.</p>
      <p>For accreditation, please contact Interaktive Demokratie by email.
         Accredited participants can request changes to this or other pages, or request
         to create new pages. To do so, checkout the wiki's repository:</p>
      <code>hg clone http://dev.liquidfeedback.org/revision-control/hg/fruit-wiki</code>
      <p>Create an initiative in the correct
         <a href="/revision-control/lf3/unit/show/1.html">
           LiquidFeedback unit</a>
         and create a new corresponding branch in the repository, e.g. if the
         initiative ID is 123, execute:</p>
      <code>hg branch i123</code>
      <p>Make your changes and commit them in one or multiple changesets:</p>
      <code>hg commit -m 'Created Apple and Banana lemma'</code>
      <p>Push your changes to the server repository:</p>
      <code>hg push</code>
      <p>Advertise your LiquidFeedback initiative and wait for it to win. ;) If so,
         your changes will be applied to the trunk and the wiki pages will be updated
         accordingly.</p>
    </div>
  </body>
</html>
```

Source code 5.2: A minimalistic template for the minimalistic wiki engine for use with lf4rcs

VI. Conclusion

VI.1. Democratic file revision control is possible

Revision control of files held in a repository can be done collectively in a democratic way. An approach to extend the already existing software LiquidFeedback to utilize its unique proposition development and decision making process for that purpose has been shown.

VI.1.a. Eliminate privileged moderators

The described approach can be used to get rid of the need of privileged persons moderating the process of incorporating changes and controlling the write-access to the repository.

VI.1.b. Increase decision quality by finding specialists for decisions

Instead, the idea of Liquid Democracy can be used to dynamically delegate votes based on subject areas and issues to find experts which are specialists for these subject areas or issues and able to make decisions with a higher quality than the average user would be able to.

VI.1.c. Potential of quality increase

The described approach has the potential to drastically increase the quality of incorporated changesets to files created collectively by larger groups. Therefore, projects making use of the described approach could increase their overall output quality.

VI.2. Affected fields

Any use of revision control systems to coordinate the work of multiple persons (e.g. authors) is from a technical point of view exactly the same. It is controlling versions and tracking changes of files held in a repository. Therefore, it can be deduced that the mechanisms shown in this paper are applicable to any possible use case of revision control systems.

VI.2.a. Application fields already using revision control systems

Revision control systems are used e.g. for

- software development,
- configuration management,
- article, book, etc. writing,
- construction and engineering,
- scientific purposes,
- business purposes,
- art and design, and
- media.

This incomplete list of actual use cases of revision control systems shows that the described approach opens a very wide range of new application fields of LiquidFeedback and its proposition development and decision making process.

VI.2.b. Applications in democratic context

At the same time, new application fields for revision control systems are opened in contexts which did not allow use of such systems before, because the final control needs to be executed democratically.

VI.2.c. New application fields created by synergistic effects

It is possible to create complete new application fields, based on the synergistic effects of using LiquidFeedback's proposition development and decision making process to control revisions of files in a repository.

It has already been proposed to use revision control systems in the context of tracking governmental resources. [Simmons]

VI.2.d. Applications integrating aspects of revision control, e.g. Wiki, MediaWiki and Wikipedia

There are more application fields where aspects of revision control systems are integrated in software systems while normal users are sometimes not even aware of it.

Famous examples are the hundreds of existing Wiki systems, most famously the MediaWiki used by the Wikipedia, which is edited by thousands of authors. Such systems are using a revision control system to track changesets and the current version of the documents presented by the Wiki system.

But in conflict situations, edit wars may arise and then the platform administrators need to "lock down" the article and review all changes manually and decide about them eventually. To get rid of this problem, the described approach can also be adopted by Wikis, allowing them to organize their internal processes in a more democratic way.

Therefore, the described approach is also a prototype, how infrastructure platforms like the Wikipedia could be further democratized.

VI.3. Prospects

VI.3.a. Further extending the approach

The presented approach will be incorporated into the official software package of LiquidFeedback Frontend [Frontend], which is maintained by the Public Software Group [PSG]. It is possible to further extend this approach by using LiquidFeedback for a more fine-graded control of a repository, e.g. to limit changesets to certain modules, directory and/or file name patterns, or other applicable criteria depending on the area and/or the policy chosen for the issue. It is also thinkable to associate changesets with suggestions. Furthermore, it is possible to integrate a complete visualization of the information and meta information held by repositories in LiquidFeedback.

Deeper integration can be achieved by allowing modification of the files of a repository branch directly in LiquidFeedback (e.g. integrating a WYSIWYG text editor or other file editors) and automatically generate changesets representing the changes made to the files and associate them with LiquidFeedback initiatives. Combining this with a sophisticated Wiki engine rendering the files held in the trunk or master branch of a repository would provide a fully integrated and complete democratic development and publishing platform for generic use in different fields of application.

VI.3.b. Meta level

On the meta level, this paper also shows that LiquidFeedback's proposition development and decision making process is not limited to conventional democratic decisions in parties and other organizations, but it can also be adopted to completely different application fields. This should endorse examination of other electronic systems, which are used by a larger group of people, regarding how to take advantage of LiquidFeedback's proposition development and decision making process.

VI.3.c. Social impact

As with any technological change, the broader use of LiquidFeedback in the context of revision control and other application fields has to go along with a cultural adoption of the new technology.

The presented approach allows to collectively organize any type of data related work which is carried out by a larger group of people without the need of a moderator or a decision hierarchy.

Companies, organizations, and voluntary projects can rethink their organizational scheme to master the challenges of the digital revolution, to set free the wisdom and abilities of their workers, and to benefit from reduced overhead. Therefore, the details of application in different fields and the consequences for companies and organizational structures but also for the working people needs further research and discussion in different scientific fields.

[Apache] The "Apache HTTP Server ('httpd')" (web server) by The Apache Software Foundation. Website http://httpd.apache.org/ (interactive)

[ArchLinux] The "Arch Linux" Linux distribution. Website https://www.archlinux.org/ (interactive)

[Core] Software "LiquidFeedback Core". Website http://www.public-software-group.org/liquid_feedback_core

[Debian] The "Debian" operating system. Website https://www.debian.org/ (interactive)

[Evolution] Jan Behrens: The Evolution of Proportional Representation in LiquidFeedback. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 1 (2014-03-20). ISSN 2198-9532. Published by Interaktive Demokratie e. V.

[FreeBSD] The "FreeBSD" operating system. Website https://www.freebsd.org/ (interactive)

[Frontend] Software "LiquidFeedback Frontend". Website http://www.public-software-group.org/liquid_feedback_frontend

[Git] The "Git" ditributed revision control system. Website https://git-scm.com/ (interactive)

[GitWeb] Software component "GitWeb" of the 'Git' revision control system, see https://git-scm.com/book/en/v1/Git-on-the-Server-GitWeb

[git-http-backend] Software component "git-http-backend" of the 'Git' revision control system, see http://git-scm.com/docs/git-http-backend

[HgWeb] Software component "HgWeb" of the 'Mercurial' revision control system, see https://mercurial.selenic.com/wiki/PublishingRepositories?action=recall&rev=192#hgweb_-_introduction_and_prerequisites

[IAD] Association "Interaktive Demokratie e. V.", Berlin, Germany, an association founded by the inventors of LiquidFeedback. Website http://www.interaktive-demokratie.org/ (interactive)

[LDJournal] "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems". ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/

[LF] Project page of "LiquidFeedback" at http://www.public-software-group.org/liquid_feedback

[Lighttpd] The "Lighttpd" web server. Website http://www.lighttpd.net/ (interactive)

[liquidfeedback.org] Website http://liquidfeedback.org/ (interactive)

[Lua] The programming language "Lua". Website http://www.lua.org/ (interactive)

[Markdown2] Python implementation "markdown2" of the 'Markdown' markup language. Website https://github.com/trentm/python-markdown2 (interactive)

[MediaWiki] The "MediaWiki" wiki system, written in PHP. Website https://www.mediawiki.org/wiki/MediaWiki (interactive)

[Mercurial] The "Mercurial" distributed source control management system. Website https://mercurial.selenic.com/ (interactive)

[Moonbridge] The "Moonbridge Network Server for Lua Applications". Website http://www.public-software-group.org/moonbridge

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[PSG] Association "Public Software Group e. V.", Berlin, Germany, publisher and copyright holder of LiquidFeedback. Website http://www.public-software-group.org/ (interactive)

[PostgreSQL] The "PostgreSQL" object-relational database system. Website http://www.postgresql.org/ (interactive)

[Roundup] http://sourceforge.net/p/roundup/code/commit_browser as of July 26, 2015

[Schulze] Markus Schulze: "A New Monotonic, Clone-Independent, Reversal Symmetric, and Condorcet-Consistent Single-Winner Election Method, draft, May 19, 2014". http://m-schulze.9mail.de/schulze1.pdf

[Simmons] Shannon N. Simmons, Justin M. Grimes, Elizabeth M. Bonsignore: "Tracking 'Change': The Importance of Applying Version Control to Government Resources", issue date Feburary 8, 2009, published on April 3, 2010 on the Illinois Digital Environment for Access to Learning and Scholarship (IDEALS). http://hdl.handle.net/2142/15325

[Subversion] The "Apache Subversion" version control system. Website http://subversion.apache.org/ (interactive)

[TieBreaker] Jan Behrens: Search for a Tie-breaker. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 2 (2014-10-07). ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/2/The_Liquid_Democracy_Journal-Issue002-05-Search_for_a_Tie-breaker.html

[Torvalds] http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git as of July 26, 2015

[WebMCP] The "WebMCP" web application framework. Website http://www.public-software-group.org/webmcp

[Wikipedia] The free online-encyclopedia "Wikipedia". Website https://www.wikipedia.org/ (interactive)

[Wikipedia2] https://en.wikipedia.org/w/index.php?title=Wikipedia&action=history as of July 26, 2015

[Wunstorf] https://wunstorf-direkt.de/lf/ as of July 26, 2015

A Finite Discourse Space for an Infinite Number of Participants

by Jan Behrens, Andreas Nitsche, Björn Swierczek, Berlin, July 28, 2015

LiquidFeedback's proposition development and decision making system has been designed with the goal of allowing a huge number of participants to take part in a structured discussion process and to make decisions without the need for a moderator or request commission with special privileges, [PLF] [QA1, question 6] and LiquidFeedback has already been successfully used with several thousand participants in the context of a political party. [Kling] Nevertheless, theoretic considerations in the past revealed that the LiquidFeedback proposition development and decision making process does not scale where the number of participants grows ad infinitum. [Evolution, p.33, p.40] While these limitations have been of theoretic nature without any effect on real life scenarios (to the knowledge of the authors of this article), future applications of the LiquidFeedback process in bigger scenarios might create practical implications of these shortcomings. Therefore, we want to present a modification of the LiquidFeedback proposition development process to allow for a potentially unlimited number of active participants while keeping the system usable and ensuring that minorities can still present their positions in an appropriate way.

Previous work

When we consider the "discourse space" in real-life discussions, we are faced with its limitations regarding space and time: space in regards to how many people fit in a room or conference hall, for example, and time in regards to how much time the discussion takes for everyone to contribute their point of view (while the other participants should be listening). Using electronic media as discourse space, it seems like we can completely overcome the limits of the discourse space in the digital world, because an internet-based system like LiquidFeedback allows to have a huge amount of proposals to be discussed in parallel. But we demand from a truly democratic system that every participant should have the ability to directly influence every decision made, even if Liquid Democracy and thus LiquidFeedback does allow for a division of labor

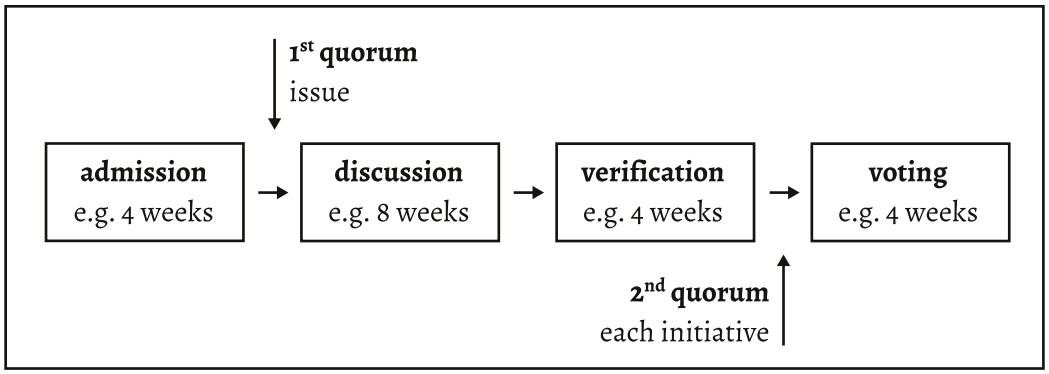


Figure 1: The four phases of LiquidFeedback's proposition development and decision making process

among the participants. This being said, and considering that the participants have only limited resources regarding their engagement in the system, it is still necessary to employ algorithms which relieve the participants from the need to deal with an unlimited number of proposals and instead allow the participants to focus on the most important issues without risking to abstain from a decision due to accident or overload.

It is noteworthy that a simple limit of the number of proposals posted by a single person in a given time frame (which is one of the measures currently implemented by LiquidFeedback) cannot limit the number of proposals posted in a given time in the system if we assume that the number of participants shall be allowed to grow ad infinitum (which is the presumption of this article). [Evolution, p.33, p.40]

For our further considerations, we shall first have a look at the LiquidFeedback process for proposition development and decision making: *See Figure 1*

Each issue in LiquidFeedback (which may consist of one or more competing proposals, which are called initiatives) can pass the four phases. In admission phase, at least one initiative in an issue must reach a given quorum of supporters (first quorum) in order to proceed to discussion phase. In order to avoid vote-splitting or tactical considerations, each participant may support as many initiatives as one wants to. The first quorum is intended to reduce the potential workload of the other participants by discussing only matters that are at least demanded by a given fraction of participants. Despite this first quorum, the number of proposals in discussion phase could unfortunately grow beyond any limit if we assume that a minority of a size bigger than the first quorum might keep supporting more and more initiatives.

Furthermore, the number of issues in admission phase is not reduced by this quorum at all since the first quorum can only reduce the number of issues in discussion, verification, and voting phase. As we still want minorities

smaller than the first quorum to be able to put up their proposals for discussion (either in admission phase or as competing initiatives in existing issues which have passed the first quorum), LiquidFeedback employs sophisticated algorithms to ensure that minorities get a fair display position within the list of issues in admission phase ("Proportional Runoff" algorithm) and the list of initiatives within an issue ("Harmonic Weighting" algorithm). By assigning display positions in a proportional fashion, these algorithms ensure that minority viewpoints cannot be drowned by noisy minorities (or majorities) which support a huge number of other initiatives.

The algorithms are explained in detail in the book "The Principles of LiquidFeedback" [PLF] and also in the article "The Evolution of Proportional Representation in LiquidFeedback" [Evolution]. A numeric example is given in Appendix A and B of [PLF].

Using these algorithms for a proportional representation, LiquidFeedback ensures that all minorities can put up their point of view for discussion and will get a fair display position, hence a fair chance to campaign for their position within the system, even under the condition of noisy minorities being present, who post a potentially unlimited number of other viewpoints. A short proof regarding the Harmonic Weighting algorithm can be found in [PLF, p.78].

While it is up to the participants to deal or not to deal with any issues in the admission phase (the only consequence of not dealing with an issue in admission phase is that it could be canceled and not be voted upon), participants must be more cautious with issues that have proceeded to discussion phase. In discussion phase, each issue is of importance because it may progress to voting phase and can result in an actual decision being made. Therefore, the "Proportional Runoff" algorithm is not applied for sorting issues that are in discussion, verification, or voting phase. [PLF, subsection 4.10.3] [Evolution] LiquidFeedback instead relies on the first and second supporter quorum to limit the potential workload of the participants.

In the following two sections we shall have a look at the effect of the first and second supporter quorum.

First quorum (as of LiquidFeedback 3.0)

As already mentioned in this article and as noted in [Evolution], the simple mechanism of a first supporter quorum does not prevent a discussion system to be flooded with a potentially unlimited number of proposals under the assumption that the number of participants grows without bound. To give an example: if the first supporter quorum is set to 10%, a minority of the size of 11% of the participants could flood the system with a potentially unlimited number of proposals. This even holds if each participant is limited in regards to the number of proposals he or she may post within a given time frame, assuming the total number of participants is not limited. Since any issue that is admitted for discussion phase requires attention of the participants, we can conclude that the mechanism of the first supporter

quorum does not scale if the number of participants grows ad infinitum.

Second quorum (as of LiquidFeedback 3.0)

The second quorum filters those alternative proposals, which do not have enough supporters at the beginning of the final voting phase. If an initiative doesn't have enough supporters, it will not be eligible for final voting. Here as well as in regards to the first quorum, it is possible for each participant to support as many alternative initiatives as he or she desires. (This is necessary to fulfill the Independence of Clones criterion.) [PLF, p.74, p.87] The second supporter quorum filters the number of issues and initiatives but also does (like the first supporter quorum) not impose a hard limit on the number of proposals. Therefore, a potentially unlimited number of alternative initiatives could enter the voting procedure hence appear on the ballot during voting. Considering appropriate changes to the user interface, this is not a real problem because the most relevant proposals will be listed first on the virtual ballot due to the Harmonic Weighting algorithm. A user interface might offer to disapprove a potentially unlimited number of alternative proposals in bulk. Minorities' rights would not be affected, because the Harmonic Weighting algorithm respects minorities in a proportional fashion when determining their display position on the ballot.

Scalability of LiquidFeedback 3.0

While the algorithms employed by LiquidFeedback successfully scaled for several thousand participants, it has been shown that the mechanism of the first supporter quorum does not necessarily scale if the number of participants grows ad infinitum. In the remainder of this article, we will propose a way to modify the mechanism of the first supporter quorum in such way that the proposition development and decision making process of LiquidFeedback also scales in cases where the number of participants grows beyond any previous limit.

Proportional representation and free-riding

An early proposal to solve the described limitation of scalability (in regards to the first supporter quorum) has already been outlined in [Evolution]: "One possible approach [...] is to not limit the number of proposals that are posted by each participant, but instead to limit the number of proposals that are allowed to proceed from admission phase to discussion phase for each subject area within a given time frame. In regard of the issues that are allowed to proceed, the limitation mechanism would need to provide a proportional representation of the participants. The development of such an algorithm may be interesting for future versions of LiquidFeedback and other electronic participation systems." Jan Behrens, author of that article and co-author of this article, thus demanded that any system which limits the number of issues proceeding from admission phase to discussion phase (previously limited by the first quorum as explained above) must ensure a proportional representation of the participants.

While proportional representation is a desirable property to support minorities with dis-

playing their viewpoints, it has to be noted that proportional representation comes at the price of encouraging tactical maneuvers known as "free-riding". [Evolution] Proportional representation in regards to admitting proposals for further discussion means that 10% of the participants may put up to 10% of proposals to discussion in a given time frame. While this sounds fair at a first glance, a deeper look reveals a problem: if a group may only put a limited number of proposals to discussion with their voting weight, it is advisable for that group to only vote for (i.e. support) those proposals which really need additional voting weight in order to be admitted for further discussion. In other words: "If enough other people support X, then I won't support X even if I want X to be admitted for further discussion, because not supporting X increases the number of other proposals I may promote successfully." In the context of Single Transferrable Vote systems, this effect is also known as "Hylland free-riding". [Hylland, p.150-151] [Schulze2] Numerous attempts have been made to avoid free-riding in proportional representation voting schemes, most notably a counting scheme known as "Schulze STV" invented by Markus Schulze (who also invented the "Schulze method").* However, while Markus Schulze shows that "Schulze STV" reduces the ability of Hylland free-riding (as well as vote management) to a minimum, he notes that the problem of Hylland free-riding cannot be solved completely if a proportional representation is desired:

> *We introduced a mathematical concept to describe Hylland free riding and vote management [...] and introduced an STV method [...] where the vulnerability to these strategies is minimized (i.e. methods that are vulnerable to these strategies only in those cases in which otherwise Droop proportionality would have to be violated).«*
>
> [Schulze2, p.50]

We may assume that it is also not possible for the LiquidFeedback proposition development process to limit the number of issues being admitted for further discussion while respecting the support of the participants in a proportional manner and at the same time avoiding the problem of free-riding.**

Trade-off

If proportional representation and avoiding the ability of free-riding are mutually exclusive, then we must weigh the impact of sacrificing either one.

Sacrificing proportional representation in regards to which issues are admitted for further discussion in LiquidFeedback appears to be a violation of LiquidFeedback's goal to protect minorities. However, even if a minority (or majority) could keep another smaller minority

** "Schulze STV" and the "Schulze method" are two distinct vote counting schemes. The first is creating a proportional representation of the voters by electing a number of candidates, and the second is a single-winner election method.*

*** A formal proof transferring the statement from the domain of Single Transferable Voting systems (STV) to the domain of LiquidFeedback's proposition development and decision making system is still outstanding.*

from having their supported proposals admitted for further discussion (beyond admission phase), minorities could still put up their points of view to discussion in admission phase. Any participant may post an initiative which is at least discussed during the admission phase, which is then sorted using the Proportional Runoff algorithm. Because competing initiatives pass the four phases together, any participant may also post alternative views to those initiatives which have proceeded for further discussion (and may later proceed to verification and voting phase). These alternative views are sorted using the Harmonic Weighting within each issue, hence ensuring that each minority may put up their opinion to be displayed as prominent as the size of the minority suggests. Every minority still gains proportional representation in each issue, and even new issues (i.e. initiatives that are not competing with any other existent initiative) can be put up for discussion by minorities at least in the admission phase.

Provoking free-riding, to the contrary, results in an unequal treatment of the voters. Those voters who honestly support all the proposals they want to vote upon would be punished for expressing their true wishes. Voters who support proposals in a strategic way gain advantages. Unless the supporter votes were cast hidden (which is not desirable during admission phase), some of the participants might even use automatic scripts (bots) to optimize their supporter votes for admitting initiatives for final voting. [GoD] This would certainly violate democratic standards. While advantages through tactical behavior can never be outruled completely, LiquidFeedback aims to reduce the possibility of tactical voting. [PLF, section 4.14] The problem of Hylland free-riding is only avoidable if we abstain from demanding a proportional representation of the voters in regards to which issues are admitted for further discussion.

These considerations in mind, we propose to sacrifice proportional representation when designing a process to limit the number of issues being admitted for further discussion and voting, hereby revising the concluding statement in [Evolution]. This allows for a protection against free-riding techniques in regards to which issues are allowed to progress to discussion phase. Proportional representation will still be applied when sorting issues in admission phase as well as for sorting initiatives within an issue (as already implemented since LiquidFeedback 2.2 and explained in [PLF]). The minorities' rights to adequately present their proposals within the proposition development and decision making process is thus still warranted.

A simple solution

Because proportional representation is not a goal when limiting the number of issues (for the reasons explained above), an algorithm to limit the number of issues that are in discussion, verification, or voting phase in LiquidFeedback is easy to design: a simple solution to this problem is to dynamically adjust the first quorum depending on the number of issues currently open in discussion, verification, and voting phase. As the number of issues in those

phases increases, the first supporter quorum would increase as well, hence leading to an equilibrium where the number of open issues in the system does not grow infinitely.

The algorithm in detail

Assuming the reader is familiar with all concepts presented in [PLF, chapter 4], we propose a modification to the LiquidFeedback process as follows. The first supporter quorum will be dynamically determined based on the number of issues currently in discussion, verification, and voting phase (i.e. the number of open issues that are neither closed nor in admission phase). As explained in the next section ("Dimensions of the discourse space"), the algorithm may be applied per subject area, per policy (selected rules of procedure), per organizational unit, or a combination thereof. We define 'N' as the number of issues currently in discussion, verification, or voting phase.

The first quorum Q_1 then calculates as follows:

See Figure 2

An example is given in the following Figure 3:

See Figure 3

An issue may proceed from admission phase to discussion phase if (a) the issue has spent a minimum time of T_{min} in admission phase, and at the same time (b) one of its alternative initiatives has a supporter count (including potential supporters, see [PLF, p.61, p.67]) of at least Q_1 multiplied with the reference population for that issue (see [PLF, section 4.9] for a definition of the reference population). If multiple issues fulfill this requirement, only that issue that belongs to the initiative with the highest quotient of supporters to the reference population is proceeding to discussion phase. (The creation time of the issues may serve as a tie-breaker.) Afterwards the dynamic quorum Q_{dyn} (and thus Q_1) is recalculated. Then, another issue may pass from admission to discussion phase if the newly calculated dynamic quorum is still reached by another initiative. The process is repeated until no issue reaches the dynamic quorum anymore. Finally all issues which are still in admission phase and which have been in that phase for a duration of at least T_{max} will be canceled by the system. The whole algorithm is executed at a regular interval.

The exponent β can be set to 1 to dynamically extend the admission phase proportionally when Q_1 is rising. This feature is intended to compensate a rising quorum by giving minorities a better chance to reach that quorum. Setting β to zero, in contrast, will result in a constant maximum length of the admission phase.

Introducing a minimum admission time T_{min} allows people who either delegate their vote in a subject area or for the organizational unit, or who are not member of the subject area (and thus do not belong to the reference population when a new issue is created) to intervene before a potentially big number of issues is accepted for entering discussion phase. Intervention is possible by revoking one's delegation, and/or by enlisting in a subject area or

$$Q_1 := \max(Q_{pol}, Q_{dyn})$$

$$Q_{dyn} := \begin{cases} (N+1) \cdot Q_{nom} & \text{if } N < a \\ \dfrac{((N+1)-a) \cdot 50\% + (b-(N+1)) \cdot Q_{nom}}{b-a} & \text{if } N \geq a \text{ and } N < b \\ \dfrac{((N+1)-b) \cdot 100\% + (c-(N+1)) \cdot 50\%}{c-b} & \text{if } N \geq b \text{ and } N < c \\ \infty & \text{if } N \geq c \end{cases}$$

$$T_{max} := T \cdot (Q_1 / Q_{pol})^{\beta}$$

Q_{pol}	a constant value defined by the used LiquidFeedback policy (the policy's issue quorum)
a, b, c	natural numbers (issue counts) with $a \leq b \leq c$
Q_{nom}	the quorum necessary to get an issue admitted to enter discussion phase when a−1 issues are currently open in discussion, verification, or voting phase
T_{min}	minimum admission time
T	maximum admission time if $Q_1 = Q_{pol}$
β	admission phase timing exponent

Figure 2: Calculation of dynamic quorum

declaring interest in an issue, hence increasing the reference population (see [PLF, section 4.9]). First of all, this compensates effects previously observed in LiquidFeedback where some participants with a high number of delegations could instantly lift an initiative's supporter count beyond the first quorum by simply supporting an initiative of a new issue (which inherits any existing delegations from the subject area). Secondly, this allows people who are not members of the subject area to intervene by enlisting in the subject area if they see that the participants in that subject area flood the system.

Dimensions of the discourse space

The algorithm may be applied per subject area, per policy (selected rules of procedure), or per organizational unit. This choice determines

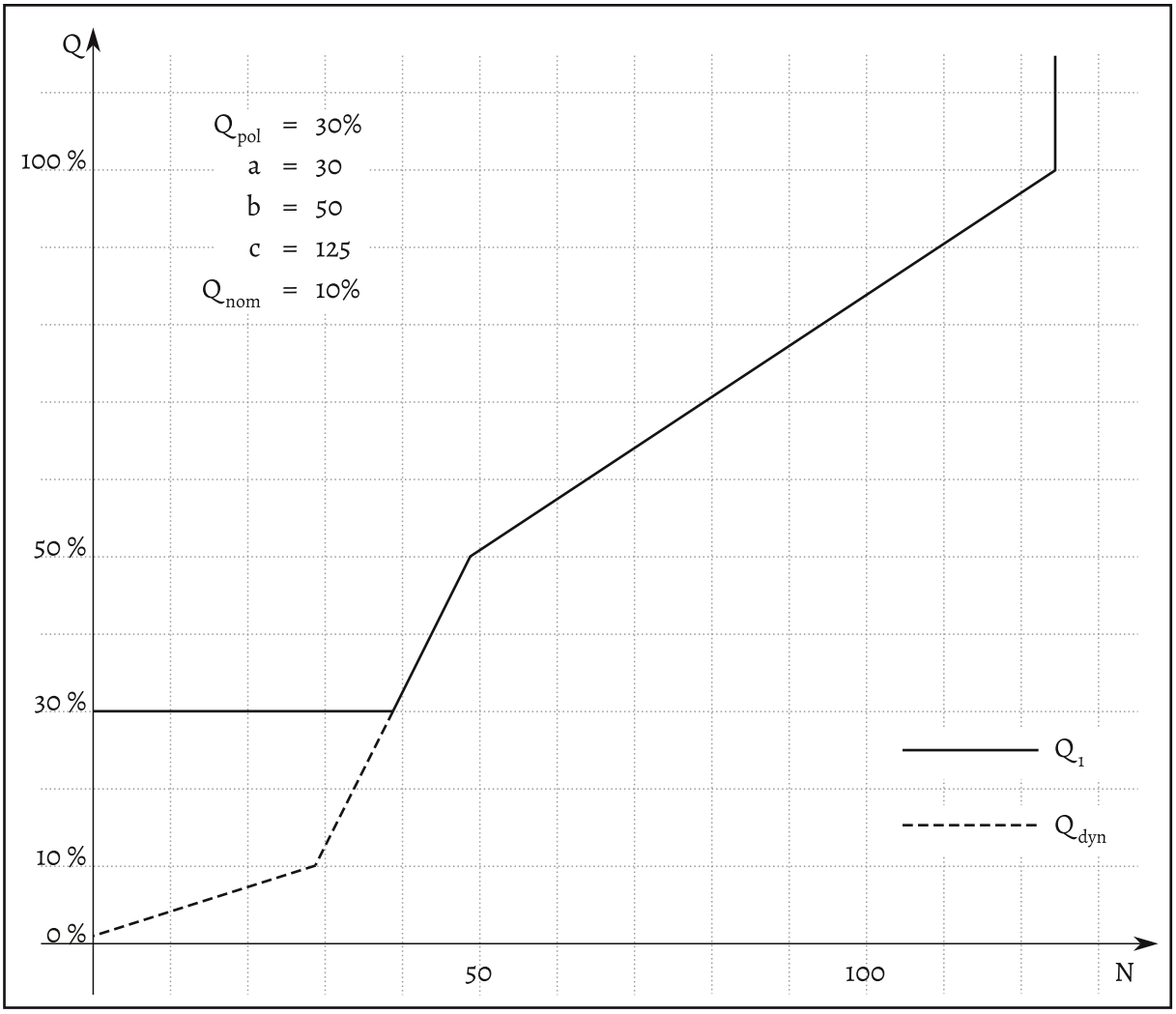


Figure 3: Example of dynamic quorum

how 'N' is defined; e.g. if we apply the algorithm per subject area, then 'N' is the number of open issues in that subject area which are not in admission phase. If we apply the algorithm per policy, then 'N' is the number of open issues not being in admission phase using this policy. We can also apply the algorithm to different sets of tuples of subject areas, policies, and/or organizational units (while for each set 'i' of tuples, we define a_i, b_i, c_i, and $Q_{nom,i}$). Even overlapping sets are thinkable, in which case we should determine the highest Q_{dyn}, i.e. $Q_1 := \max(Q_{pol}, \max(Q_{dyn,i}))$ (for all matching 'i').

It is thus possible to shape the dimensions of the discourse space. For example, an organization might decide to generally not discuss more than 150 issues in parallel, but also not more than 15 issues together in the areas "Education" and "Research". More complex setups are thinkable. Some organizations could use this feature to ensure that certain kinds of proposals do not stop other kinds of proposals being discussed and decided upon (e.g. a big number of issues regarding amendments to the statutes of an organization does not increase the quorum for other issues).

Vulnerability through flooding?

While it is possible that a minority keeps another smaller minority from having their supported issues in admission phase proceed to discussion phase (by simply supporting a huge amount of initiatives in separate issues, thus increasing the number of issues in discussion, verification, and voting phase, and thus increasing the dynamically adjusted first supporter quorum), any sufficiently larger group can still get further issues to be admitted for discussion phase. In other words: a minority cannot lift the required quorum much higher than their own size. For example, a minority consisting of 10% of the participants could deter a 9% minority from discussing their issues beyond admission phase, but those 9% could still use the admission phase to promote their proposals in order to gain a supporter quorum greater than 10%, hence allowing for a final decision during voting phase (after discussion and verification phase).

Technical challenges

LiquidFeedback currently relies on a background process, which counts the votes for each issue, one issue after the other. In order to provide a fair process where no issue is favored to any other issue, all issues that may influence each other's dynamic quorum have to be tested for their eligibility at the same time. Since the calculation requires some time to compute, a snapshot must be taken of the supporter situation of all those issues at once.

Such behavior might be achieved by processing all open issues in admission phase at the same time in a single database transaction. In case of the current implementation of LiquidFeedback, however, it might be problematic due to PostgreSQL's way of locking: database locks won't be released until the transaction has finished. Therefore, "snapshot synchronization functions" should be utilized instead. We refer to section 9.26.5 of the PostgreSQL manual,

version 9.4 for further information on this issue. [PostgreSQL]

Regarding huge numbers of initiatives, a practical problem might be the non-linear (yet polynomial) run time of the Proportional Runoff and Harmonic Weighting algorithms. Further research might reduce computational complexity, or not. Either way, the practical scaling of such a system by employing methods like clustering, etc. goes beyond the scope of this article.

Summary

It may be surmised that it is impossible to extend LiquidFeedback with a proportional algorithm to select issues that may progress from admission to discussion phase without the side effect of potentially favoring voters that employ free-riding techniques to maximize their influence. We therefore proposed a non-proportional algorithm that allows a potentially unlimited number of participants to take part in the proposition development and decison making process while limiting the number of open issues, so that each single individual or a small group of individuals may still work on all issues in the system, in a subject area, or within an organizational unit.

Applying these modifications to LiquidFeedback would empower a potentially unlimited number of participants to engage in a self-organized discussion process which meets highest democratic standards while maintaining feasibility even if the number of participants grows beyond any limits.

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

[QA1] Behrens, Kistner, Nitsche, Swierczek: Readers of the Journal Asked – LiquidFeedback Developers Answer (#1). In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 2 (2014-10-07). ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/2/The_Liquid_Democracy_Journal-Issue002-08-Readers_Asked_-_LiquidFeedback_Developers_Answer_001.html

[Kling] Christoph Carl Kling, Jerome Kunegis, Heinrich Hartmann, Markus Strohmaier, Steffen Staab: "Voting Behaviour and Power in Online Democracy: A Study of LiquidFeedback in Germany's Pirate Party". March, 2015. Published at http://arxiv.org/abs/1503.07723v1

[Evolution] Jan Behrens: The Evolution of Proportional Representation in LiquidFeedback. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems", Issue 1 (2014-03-20). ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/1/The_Liquid_Democracy_Journal-Issue001-04-The_evolution_of_proportional_representation_in_LiquidFeedback.html

[Hylland] Aanund Hylland: Proportional Representation without Party Lists. In "Rationality and Institutions : Essays in honour of Knut Midgaard on the occasion of his 60th birthday, February 11, 1991", pp. 126–153, by Raino Malnes and Arild Underda (editors). ISBN 82-00-21623-3 (978-82-00-21623-0). Published 1992 by Universitetsforlaget AS, Oslo.

[Schulze2] Markus Schulze: "Free Riding and Vote Management under Proportional Representation by the Single Transferable Vote, draft, March 14, 2011". http://m-schulze.9mail.de/schulze2.pdf

[PostgreSQL] http://www.postgresql.org/docs/9.4/static/functions-admin.html#FUNCTIONS-SNAPSHOT-SYNCHRONIZATION

Addendum to our Theorem Regarding Preferential Delegation and Negative Voting Weight

by Jan Behrens, Berlin, March 28, 2015

In our article on Preferential Delegation and Negative Voting Weight [PD], we have proven that a preferential delegation system with free choice of delegates may not fullfill the following 7 criteria at the same time:

- Precedence (respecting the freely chosen preferences in trivial cases),
- Anonymity*,
- Neutrality,
- Consistency,
- Directionality,
- Equality of Direct and Delegating Voters,
- No Negative Voting Weight Through Delegation.

While these 7 criteria make the proof easy to understand, it should be noted that if we define the absence of negative voting weight in a more general way, we could further reduce the number of conflicting properties to the following 5 properties:

- Precedence (respecting the freely chosen preferences in trivial cases),
- Anonymity*,
- Directionality,
- Equality of Direct and Delegating Voters,
- No Negative Voting Weight Through Delegation.

The definition for the absence of negative voting weight, however, needs to be redefined as follows in this case:

> *If a person A doesn't vote directly and doesn't delegate to anyone, and if (in a binary yes/no-decision) a person B votes via delegation in favor of a proposal that wins, then changing A's behavior to delegate to B instead of abstaining (i.e. neither voting directly nor delegating) must not cause the previously winning proposal to lose, and if person B votes via delegation against a proposal that loses, then changing A's behavior to delegate to B instead of abstaining must not cause the previously losing proposal to win.«*

* *not to be confused with anonymous/secret voting, see [PLF, p.148]*

As already explained in the original article, the property "Consistency" is implied by "Directionality". Furthermore, the use of "Neutrality" isn't necessary until Case XXVI of our original proof.* We may therefore copy the findings regarding Case I through Case XXII from our previous proof and consider 6 new cases.

For Case I through Case XXII, see [PD]. Case XXIII through Case XXVIII will be (re)defined as follows.

Case I through Case XXII

See [PD] for Case I through XXII.

Cases XXIII to XXVIII

For Case XXIII through XXVIII see figures on the following pages.

Contradiction

The property of "Anonymity", however, forbids that that "YES" wins in Case XXVI and "NO" wins in case XXVIII. Therefore, the 5 properties are contradictory, quod erat demonstrandum.

This article has been published as an advance publication on March 28, 2015 at the following URL:

http://www.liquid-democracy-journal.org/advance_publication/2015-03-28/Addendum_to_our_Theorem_Regarding_Preferential_Delegation_and_Negative_Voting_Weight.html

---

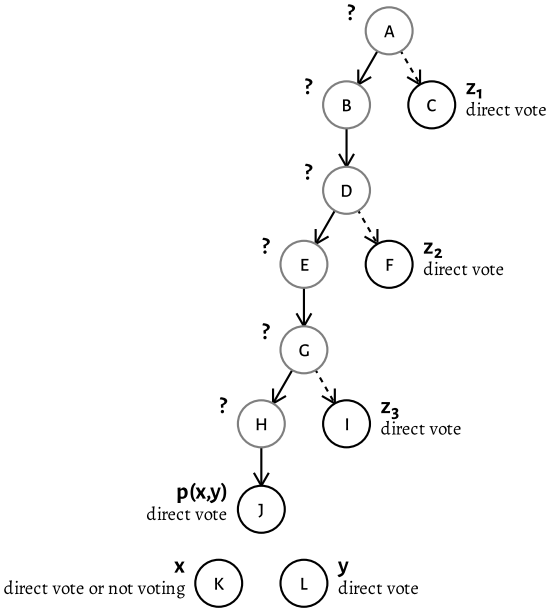
** The original proof states on page 8 that Property 3 (Neutrality) is used implicitly until Case XXIV inclusive. This is not necessary though, because for each case, "x", "y", "z_1", etc. are variable.*

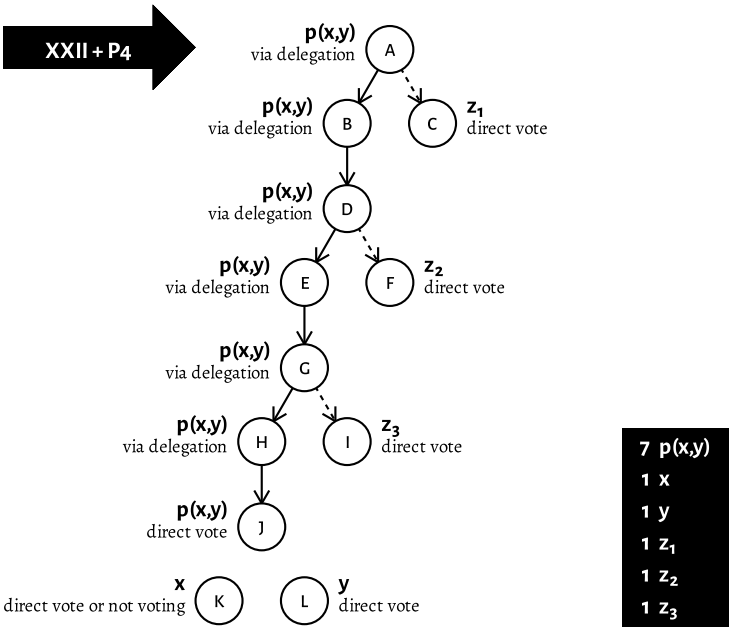
[PD] Jan Behrens & Björn Swierczek: Preferential Delegation and the Problem of Negative Voting Weight. In "The Liquid Democracy Journal on electronic participation, collective moderation, and voting systems, Issue 3" (2015-01-23). ISSN 2198-9532. Published by Interaktive Demokratie e. V., available at http://www.liquid-democracy-journal.org/issue/3/

[PLF] Behrens, Kistner, Nitsche, Swierczek: "The Principles of LiquidFeedback". ISBN 978-3-00-044795-2. Published January 2014 by Interaktive Demokratie e. V., available at http://principles.liquidfeedback.org/

Case XXIII

We consider a new Case XXIII that can be solved by using the previously solved Case XXII and applying the rules of Property 4 ("Consistency").





7 p(x,y)
1 x
1 y
1 z_1
1 z_2
1 z_3

Case XXIII

Case XXIV

We consider a new Case XXIV that can be solved by first applying the rules of Property 5 ("Directivity") to Case XX in order to determine all votes but one, and then, due to Property 6 ("Equality of Direct and Delegating Voters"), using the vote counts determined in Case XXIII to solve the last vote.

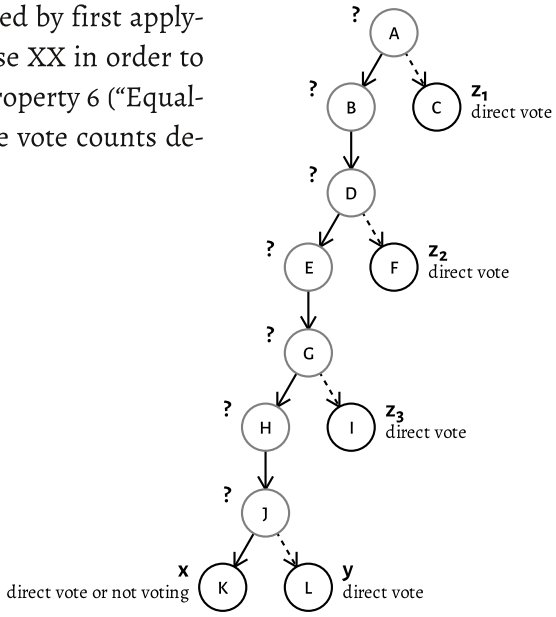
$x \in \{YES, NO, \varnothing\}$

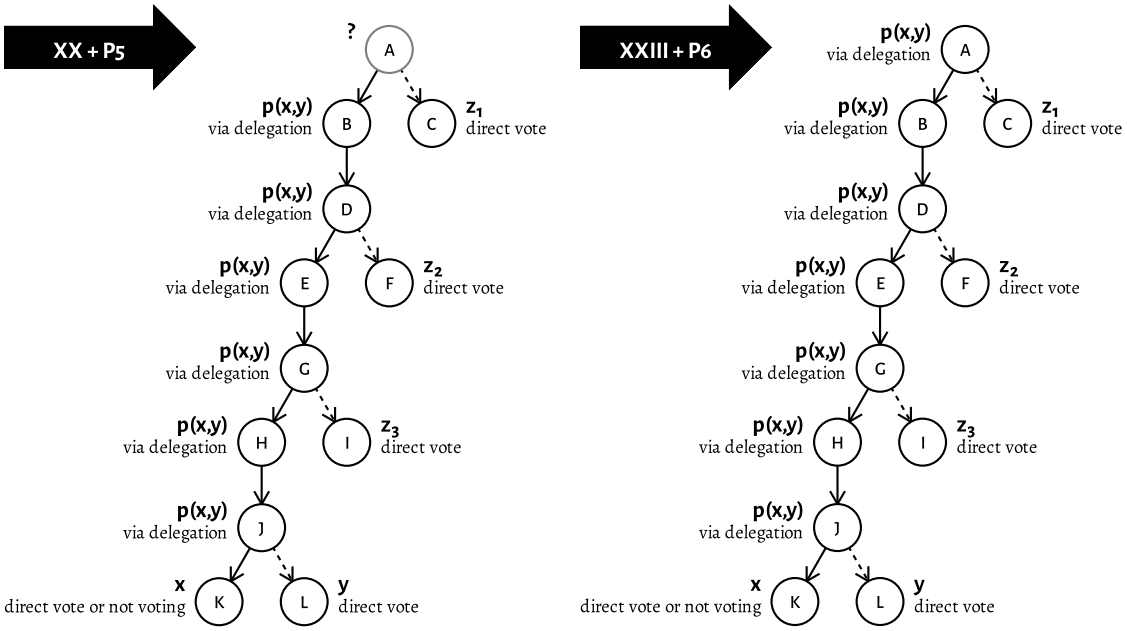
$y \in \{YES, NO\}$

$z_1 \in \{YES, NO\}$

$z_2 \in \{YES, NO\}$

$z_3 \in \{YES, NO\}$

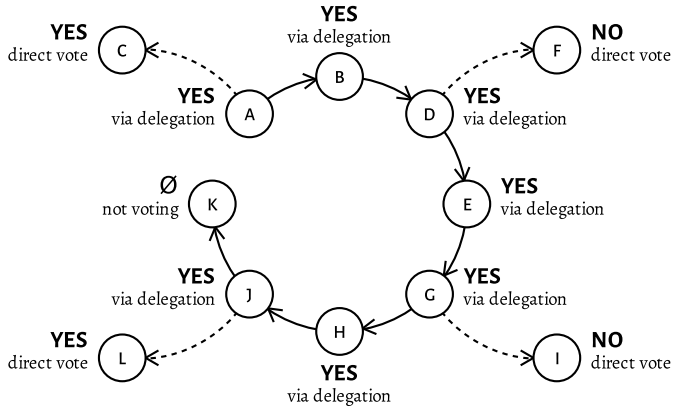




Case XXIV

Case XXV

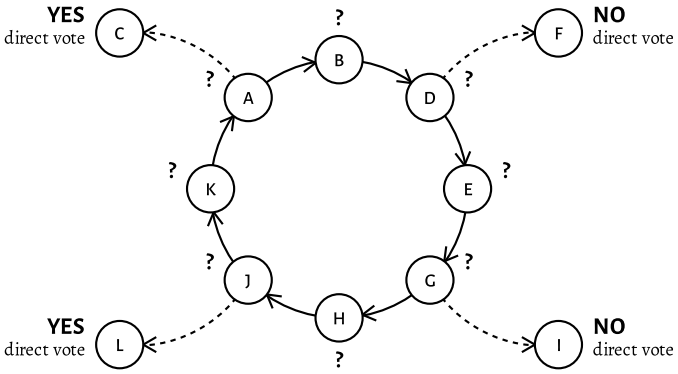
We consider Case XXIV and set x=∅, y=YES, z_1=YES, z_2=NO, z_3=NO to create a more specific Case XXV. The number of YES votes outnumbers the number of NO votes. Thus "YES" would win here.



Case XXV

Case XXVI

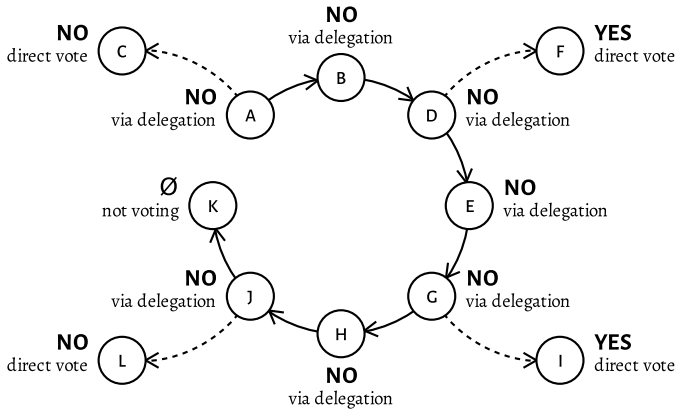
We create a Case XXVI equal to Case XXV but with the sole difference that voter K (who was previously abstaining) delegates to voter A (who was previously voting for YES through delegation). According to the requirement of the absence of negative voting weight through delegation, "YES" would need to win in Case XXVI (because it also wins in Case XXV).



Case XXVI

Case XXVII

We consider Case XXIV and set x=∅, y=NO, z_1=NO, z_2=YES, z_3=YES to create a more specific Case XXVII. The number of NO votes outnumbers the number of YES votes. Thus "NO" would win here.



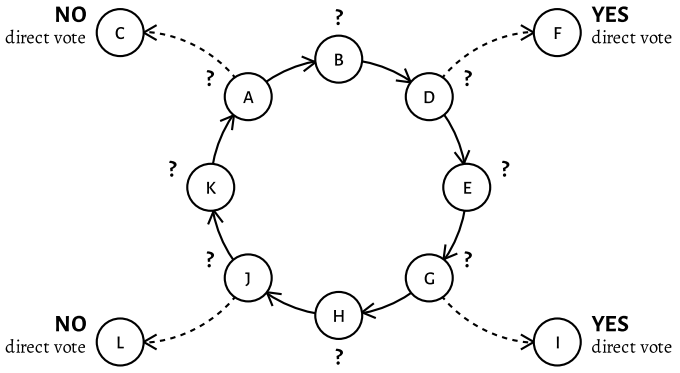
2 YES
9 NO

2 < 9
(NO would win)

Case XXVII

Case XXVIII

We create a Case XXVIII equal to Case XXVII but with the sole difference that voter K (who was previously abstaining) delegates to voter A (who was previously voting for NO through delegation). According to the requirement of the absence of negative voting weight through delegation, "NO" would need to win in Case XXVIII (because it also wins in Case XXVII).



Case XXVIII

Also published by Interaktive Demokratie e.V.:

The Principles of LiquidFeedback

This book gives an in-depth insight into the philosophical, political and technological aspects of decision making using the internet and the “secrets” of LiquidFeedback, a computer software designed to empower organizations to make democratic decisions independent of physical assemblies, giving every member of the organization an equal opportunity to participate in the democratic process.

The inventors of LiquidFeedback explain the principles and rules of procedure developed for LiquidFeedback providing the key features for democratic self-organization. They give a theoretical background about collective decision making and answers to practical questions. This is a must-read for anybody planning to make online decisions or to build online decision platforms and is also interesting for anybody interested in the future of democracy in the digital age.

More than 200 pages, including:

- detailed descriptions of the concepts of Liquid Democracy
- explanation of the structured discussion process in LiquidFeedback, including:
    - the collective moderation system
    - protection of minorities and the problem of "noisy minorities"
    - preferential voting
- reasons for the design principles of LiquidFeedback
- real-world integration into existing democratic systems
- analysis of the verifiability of voting systems
- glossary and an extensive index
- bibliographic references
- more than 20 illustrations

Order at bookstores world wide with the ISBN 978-3-00-044795-2 or at:

http://principles.liquidfeedback.org/