

Vempain technical specification

Matti Dahlbom, Jussi Ropo, Paul-Erik Törrönen

3rd October 2002

Abstract

This document describes the inner workings and specifications of the Vempain Publishing System (VPS)

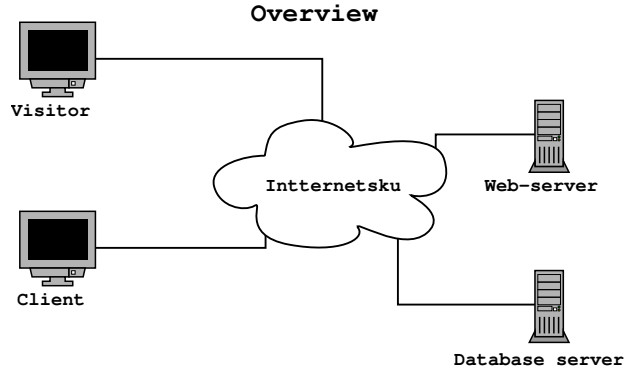
Revision : 1.3

Contents

| | | |
|----------|----------------------------|-----------|
| 1 | Introduction | 2 |
| 1.1 | License | 2 |
| 2 | Architecture | 2 |
| 2.1 | Web-server | 2 |
| 2.2 | Publisher-server | 4 |
| 2.2.1 | Procedures | 5 |
| 2.2.2 | Binary upload | 9 |
| 3 | Publisher client | 10 |
| 4 | Database | 10 |

1 Introduction

Vempain is a publishing system that supports remote updating of pages as well as remote administration. The distributed architecture makes it very scalable and thus suitable even for highvolume sites.



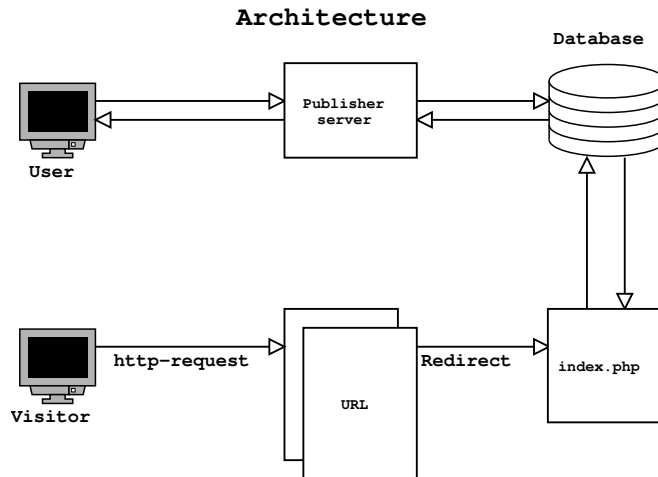
Vempain includes such features as ACL for both pages and active components, serverside scripting support (currently only PHP), SSL-support, themes and user profiles among others.

1.1 License

The software is based on open software and is itself licenced under the 2nd version of GPL. Since the components and pages are considered as separate work, they are not covered by the GPL license. This means that proprietary modules can be created and used with the VPS. VPS comes with some modules which, like the VPS itself, is covered under the GPL licence. Any changes made to the GPL licensed code should be handled according to the license.

2 Architecture

Vempain is divided into four separate parts, the web-server, publishing-server, publisher-client and the database, the two first ones both connecting to the database.

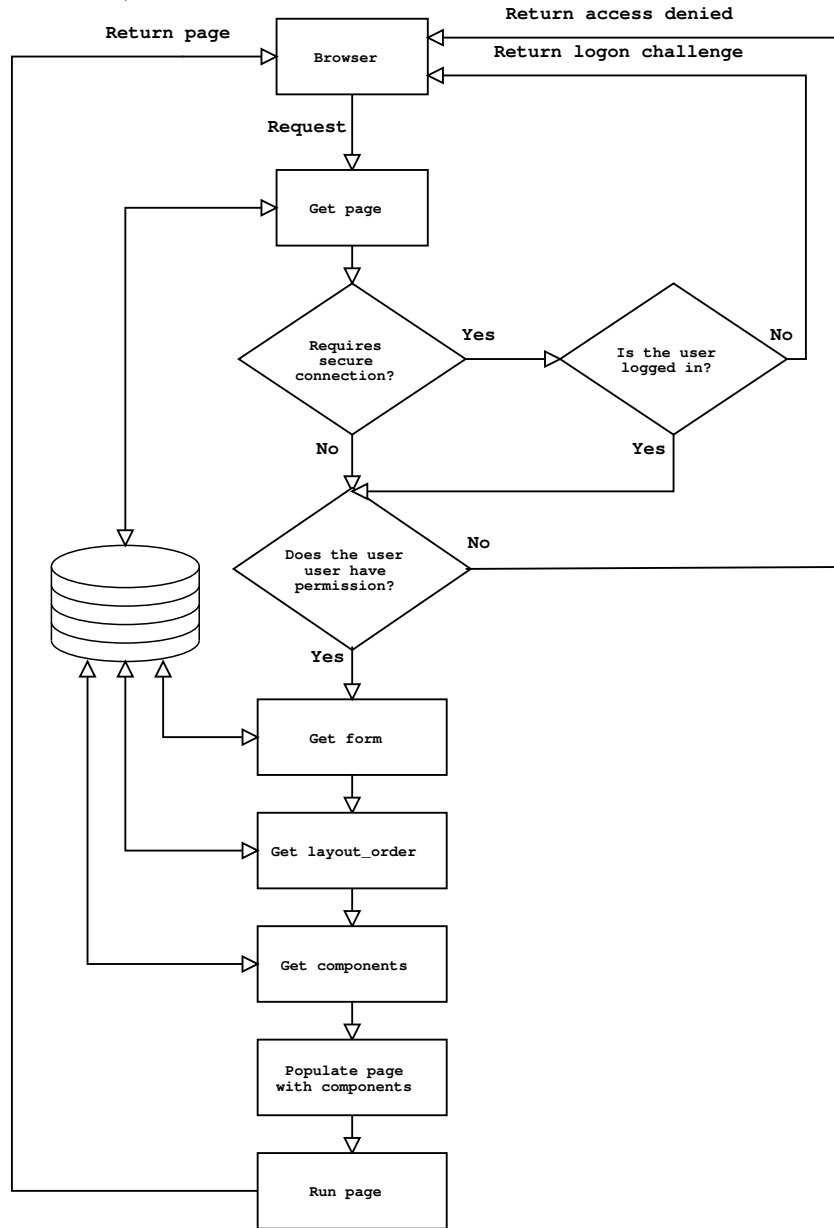


2.1 Web-server

The webserver runs the publication system which receives the http-request, parses it and decides what to do based on the suffix of the file. All requests to VPS-pages are redirected to a single page by the means of a webserver directive which in turn parses the request-string and searches the database for a corresponding document as well as its form and additional components which the script combines and runs. The product is then sent back to the requesting visitor.

If the requested page is marked restricted, then the credentials are checked.

Currently the VPS is written in PHP running on Apache, but in the future it may become necessary to rewrite the parser as a webserver module (Apache, NSAPI and Java Service are feasible).



Example:

We request for the page 'test.html'. The index.PHP to which all requests are routed to strips the file-descriptor (in this case '.html') and does the following query to the page-table:

```

SELECT form_id, title, header, body, secure
FROM page
WHERE path = 'test'
    
```

If the recordset would be empty, the index.php would do a subsequent request for the error-page. Otherwise we get the following result:

| form_id | title | header | body | secure |
|---------|-----------|-----------|-------------|--------|
| 100 | Test Page | Test Page | Hello world | 0 |

Now this tells us that the page uses the 100 form, it's title as well as the header is 'Test Page', its entire content is the 'Hello world'-string. The page security is set to 0 (not requiring SSL).

Next we make a request for the layout by the form_id:

```
SELECT lo.structure AS struct,
       fo.form_name AS form_name,
       fo.form_id AS form_id
FROM   layout lo, form fo
WHERE  fo.form_id = 100
AND    lo.layout_id = fo.layout_id
```

Form is nothing more than an abstraction between the page and layout so we don't have to make a new layout for every page. The layout can contain HTML or php-code, but (as below) it mainly uses the component- and page-tags.

| struct | form_name | form_id |
|---------------------------------------|-----------|---------|
| <!--comp_0--><!--page--><!--comp_2--> | Test page | 100 |

After this we loop through the result-set and replace the component- tags with the components above. Note that components can be either HTML or PHP. As all components are replaced, the index.php runs an eval-command on the whole and in our case should produce a nice simple Hello world-page.

```
<HTML>
<BODY BGCOLOR="#000000"{}>

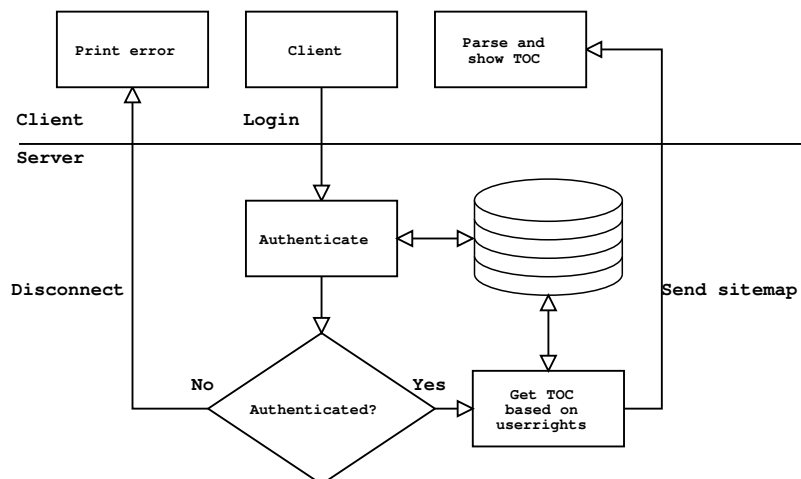
Hello world

</BODY>
</HTML>
```

2.2 Publisher-server

The publisher is written in Java using RMI as the c/s-protocol. The client in itself does not contain any editor, instead the user configures his/her favorite editor to be called (provided that the editor supports filename as command-line parameters).

The connection between the client and server is encrypted (at this time only SSL is supported) to ensure the security over open networks.



The log-in is a simple check of user/password combination against that found on the database. Also the version of the client and the MD5-hash of the classes as well as the protocol are checked, in case of incompatibility. If matched, the server sends the latest XML-representation of the site wherein the structure as well the rights (true/false) is stored. The XML is sent back to the client for a visual representation.

The general form of the XML is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<site-toc>
  <site-info>
```

```

    <protocol version="1.0"/>
</site-info>
<site-root>
  <permission type="user|unit" id="<user_id>" privilege="crmd"/>
  <permission type="user|unit" id="<user_id>" privilege="crmd"/>

  <directory id="<dir_id>" name="<dir_name>">
    <permission type="user|unit" id="<user_id>" privilege="crmd"/>
    <permission type="user|unit" id="<user_id>" privilege="crmd"/>

    <page id="page_id" checkout="0|<user_id>" name="<page_name>">
      <permission type="user|unit" id="<user_id>" privilege="crmd"/>
      <permission type="user|unit" id="<user_id>" privilege="crmd"/>
      <component id="comp_id" order="0" checkout="0|<user_id>" name="<comp_name>">
        <permission type="user|unit" id="<user_id>" privilege="crmd"/>
        <permission type="user|unit" id="<user_id>" privilege="crmd"/>
      </component>
    </page>
  </directory>
</site-root>
<site-user>
  <user id="<user_id>" name="<user_name>" />
</site-user>
<site-unit>
  <unit id="<user_id>" name="<user_name>" />
</site-unit>
</site-toc>

```

site-toc

This is the main data-branch.

site-info

Optional data, such as short description of the site.

site-root

Main branch of the virtual filesystem.

directory

Can contain other directories as well as files.

page

The single page which is shown via the web-server to the browser.

component

A component-list which makes the browseable page.

Site-user

This is a list of the users on the site. It will be used when the user assigns additional rights to objects to which he has modify-permissions.

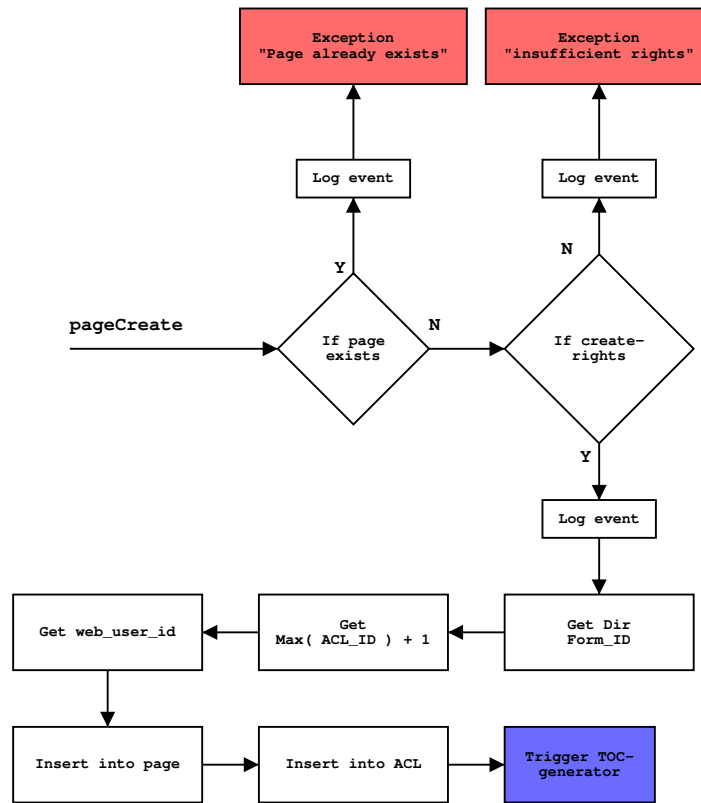
Site-unit

This is a list of the units on the site. It will be used when the user assigns additional rights to objects to which he has modify-permissions.

2.2.1 Procedures**Creating a new page**

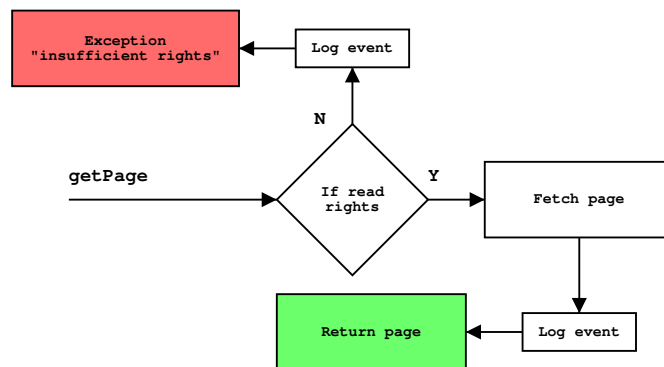
1. Check whether the page with the given path already exists.
2. Check the user-rights for create from the parent directory ACL.
3. Get the form for the parent directory, this will be the base of the new page.
4. Get the next available ACL-id.

5. Get the web-user id.
6. Create a new record in the page-table with the collected data.
7. Create an ACL-entry for the page. Web-user with read-privileges and user with full privileges.
8. Finally the TOC-generation is triggered and the new TOC is distributed to all connected clients.



Get page

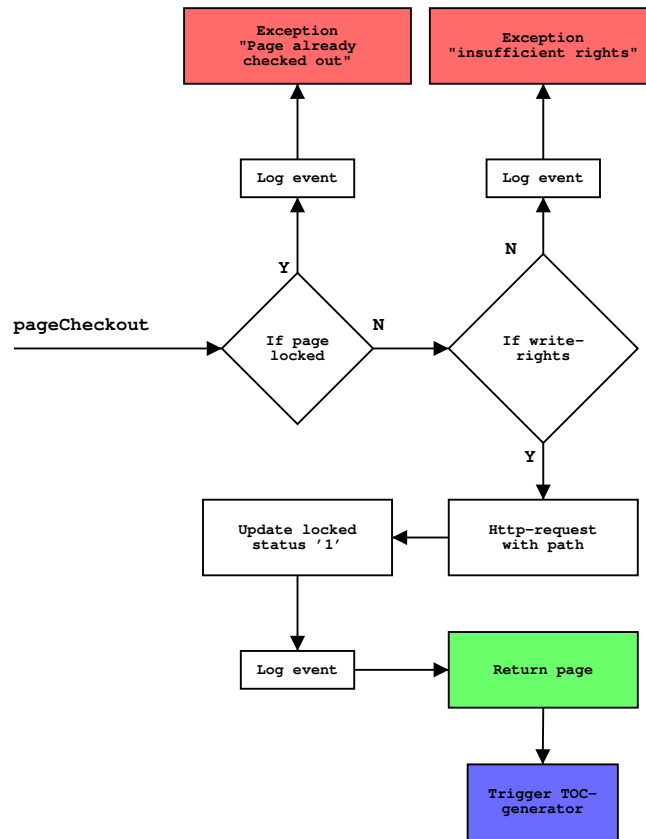
1. Check the user-rights for read from the page ACL.
2. Do a HTTP-request for the page with devel-parameter.
3. Send the page to the client.



Checkout page

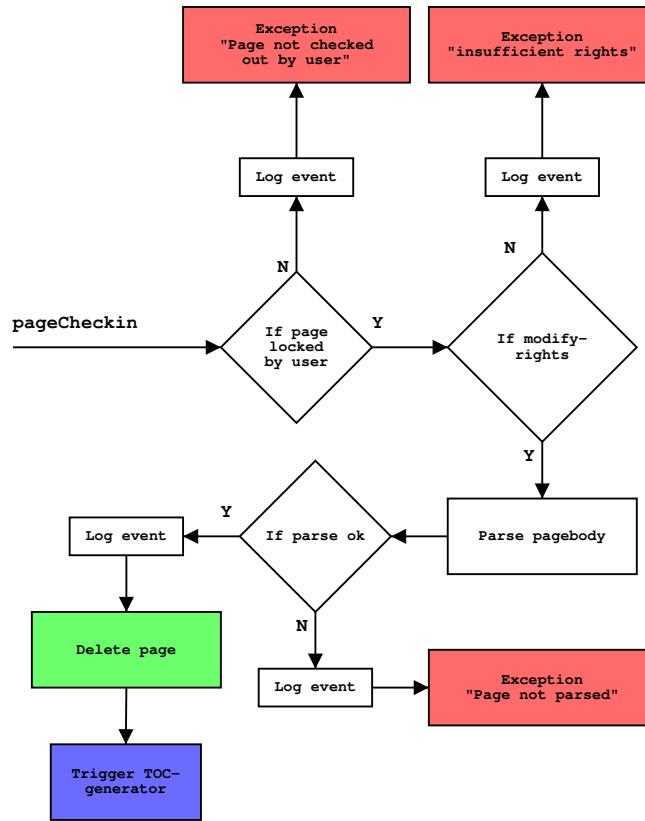
1. Check if the page is checked out.
2. Check the user-rights for modify from the page ACL.

3. Do a HTTP-request for the page with devel-parameter.
4. Update the page-table, set status to locked.
5. Send the page to the client.
6. Finally the TOC-generation is triggered and the new TOC is distributed to all connected clients.



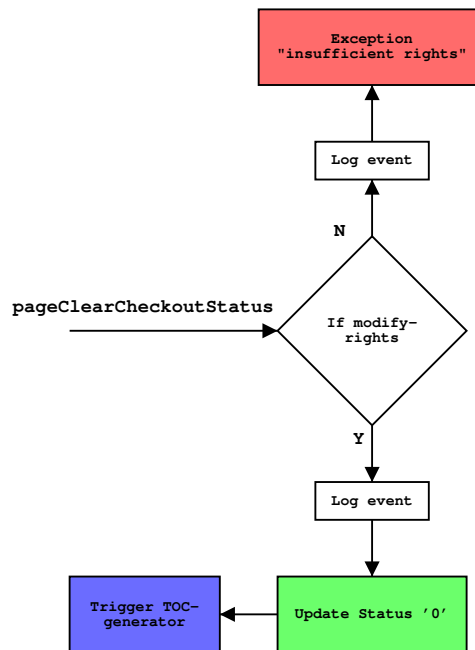
Checking page

1. Check if the page is checked out.
2. Check the user-rights for modify from the page ACL.
3. Extract the body of the sent page.
4. Unlock the page.
5. Finally the TOC-generation is triggered and the new TOC is distributed to all connected clients.



Clear checkout status

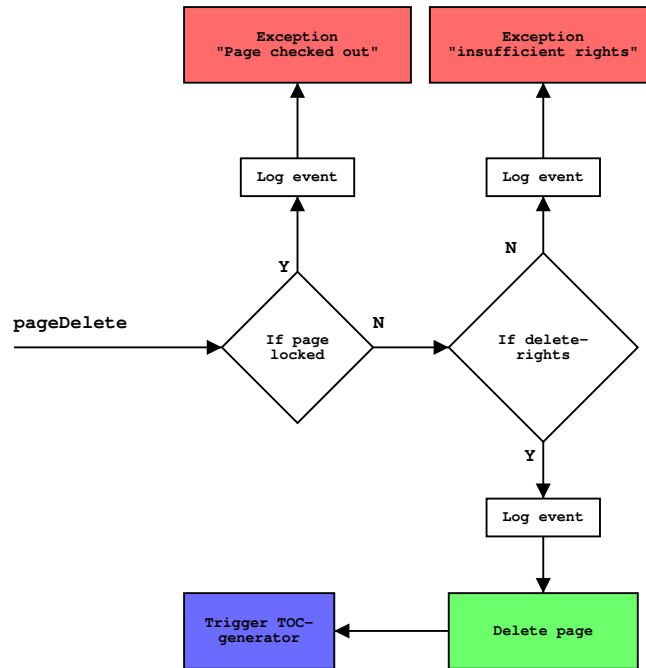
1. Check the user-rights for modify from the page ACL.
2. Unlock the page.
3. Finally the TOC-generation is triggered and the new TOC is distributed to all connected clients.



Delete page

1. Check if the page is checked out.

2. Check the user-rights for delete from the page ACL.
3. Delete page-entry from ACL-table.
4. Delete page-entry from page-table.
5. Finally the TOC-generation is triggered and the new TOC is distributed to all connected clients.



2.2.2 Binary upload

The binary files (image, executable, zip etc.) are handled differently from the pages in that they are stored on the harddisk and are listed in two tables on the SQL-server, `binary_file` and `binary_file_referer`. The first lists uniquely the path+filename as well as the ACL_id of the page (the owner), MD5Sum and of course the creator, modifier, dates of creation and last modification. The second table lists all the references to the file, including the ACL_id of the referencing page, order of referrer (the owner-referrer is the first), page_id of the referrer-page, creator and date of creation. The version of the binary file is checked by computing the MD5Sum of the file.

| bin_id | path | acl_id | md5sum | creator | created | modifier | modified | mime_type |
|--------|---------------------------------|--------|------------------|---------|---------------------|----------|---------------------|--------------------------|
| 1 | /Images/bgr.jpg | 300 | EG4TH9GJT74O3RU4 | 2 | 2001-08-11 13:22:54 | 3 | 2001-08-11 13:44:42 | image/jpeg |
| 2 | /Menot/Festarit/kuvat.zip | 542 | OIEWOIHF D98YFFW | 2 | 2001-08-21 19:42:42 | | | application/zip |
| 3 | /Pelit/FPS/Doom/sw_doom-1.1.exe | 745 | EOIEWOIWE9IE93R2 | 3 | 2001-08-25 22:12:42 | | | application/octet-stream |

The reason why the ACL of the owner-page is used is that all users who have the right to edit the page must also be able to remove the file. This may lead to complicated permissions regarding the file, but is a much cleaner implementation than using a separate permission-system for the binary-files.

When a user downloads a page which contains references to binary-files, only those referred by the CSS and IMG-tag is also sent. Other types of referencing tags are: A, CSS, OBJECT, APPLET and SCRIPT.

1. User adds a reference to a binary file.
2. User uploads the page to the server.
3. Publisher-client parses the page for any references to binary files and computes the MD5Sums for the files.
4. Publisher-client checks whether the user has permission to create files in the selected directory (may differ from the pagepath).
5. Client checks whether the server has any of the listed binaries and their MD5Sum.
6. If the server has the binary (the path matches), the MD5Sum matches and the user has correct permissions, the client notifies the user and server adds the user and edited page as a referer to the `binary_file-referer`-table.

7. If the file does not exist, the client notifies the user and uploads the file to the server. The user is set as the owner of the file and is added to the `binary_file-referer-table`, the file is added to the `binary_file-table`.
8. If the file exists but with a different MD5Sum and the user is not the owner of the file, the client notifies the user and asks whether the file on the server should be used or should the local file be uniquely renamed (in which case the user becomes the owner). If the user chooses to use the file on the server, he is added to the `binary_file-reference-table`.
9. If the file exists but with a different MD5Sum and the user is the owner of the file, and the file has multiple references but only one user is listed (ie. the owner has multiple pages referring to the same file), the client asks whether the file should be updated or whether the local file should be uniquely renamed and uploaded.
10. If the file exists but with a different MD5Sum and the user is the owner of the file, and the file has multiple references as well as users (ie. the file is referenced by multiple pages made by multiple users), the client asks whether the user wants his own references to point to a updated page (and making the next user in the reference-table the new owner) or whether the file on the server is used instead of the local) or whether the local file should be uniquely renamed and uploaded.

When a reference to a binary file is deleted the server checks several things:

- Was it the last referrer, in case which the file is deleted and removed from the binary-tables.
- Was it the owner, in case which the next in the referer-order is promoted to owner.

Note that the order does not need to be a strict number order, as references in the middle can be deleted without changing the rest. The only rule is that the newest reference has the highest number.

In practice all files except those configured as page-files (usually `.html`) will be treated as binary-pages. This is in a way a limitation of the system. On the other hand there is no rule that demands that the pages contain anything but text with some restrictions (an explicit MIME type may be added for each page in later versions).

At some time in the future the system may be changed so that all files are stored on the SQL-server but insofar there have been no reasons to do this. In contrary the are reasons not to do this, netcongestion between the SQL- and Web-servers, lack of caching (think filecache).

3 Publisher client

4 Database

At this time MySQL is the database used and all the code is written against it. Because of this, the VPS requires only minor changes to use some other database such as Oracle, DB2 or MS SQL. There are no intentions to have support for other databases at this point partly because recent developement in MySQL and lack of need to support for other DB.