Chair of
Software Engineering

Origo Home: Web Interface Design and Development for Origo

Master Thesis

By: Peter Wyss
Supervised by: Till Bay
Prof. Bertrand Meyer

Student Number: 02-920-544

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Abstract

The Origo web interface is the front-end to the Origo Core system and is based on the Drupal CMS. It provides all main functions to host and manage a project on Origo. This includes user and project management, wiki pages, blogs, issue tracker and releases. The Origo Home page is the central information point for any Origo user. Any changes in own and bookmarked projects are listed in several different workitems. With these workitems a user is always informed about what is going on in his project.

Acknowledgments

I would like to thank my coworkers Rafael Bischof and Patrick Ruckstuhl, who both helped me a lot with Eiffel related problems and with valuable input on how to improve Origo web.

A special thank goes to my supervisor Till Bay for his effort in acquiring all the infrastructure and services needed to run Origo, and of course for his nice graphic design used in many parts of the web.

Contents

Abstract **ii**

Acknowledgments **iii**

1 Introduction **1**
1.1 Goal . . . 1
1.2 Overview . . . 1

2 Design **2**
2.1 General . . . 2
2.2 Workitems . . . 2
2.3 Drupal Sites . . . 3
2.4 Scalability . . . 3

3 Drupal Modules **4**
3.1 Origo Auth . . . 4
3.1.1 User Registration . . . 4
3.1.2 Session Handling . . . 4
3.1.3 User Login . . . 6
3.1.4 User Logout . . . 6
3.1.5 Password Change . . . 6
3.1.6 Lost Password . . . 8
3.1.7 XML-RPC Wrapper . . . 8
3.2 Origo Home . . . 8
3.2.1 Origo Home . . . 9
3.2.2 Workitem Subscription . . . 10
3.2.3 Project Bookmarks . . . 10
3.2.4 Project List . . . 10
3.2.5 User Key Request . . . 10
3.2.6 E-Mail Change . . . 10
3.2.7 Project Settings . . . 11
3.2.8 Project Creation Request . . . 11
3.2.9 Administration Menu . . . 12
3.3 Issue Tracker . . . 12

3.4 Developer Pages . . . 12
3.5 Existing Modules . . . 13
3.5.1 Captcha . . . 13
3.5.2 Diff . . . 13
3.5.3 Form Store . . . 13
3.5.4 GeSHi Filter . . . 13
3.5.5 Google Analytics . . . 13
3.5.6 Google Co-op CSE . . . 14
3.5.7 Image . . . 14
3.5.8 Image Assist . . . 14
3.5.9 Pathauto . . . 14
3.5.10 PEAR Wiki filter . . . 14
3.5.11 Tag Query Language . . . 14
3.5.12 Wikitools . . . 15

4 Workitem Implementation 16
4.1 Issue Workitem . . . 16
4.1.1 Origo API Call . . . 16
4.1.2 Drupal Integration . . . 18
4.2 Release Workitem . . . 18
4.2.1 Origo API Call . . . 18
4.2.2 Drupal Integration . . . 18
4.3 Commit Workitem . . . 18
4.3.1 Origo API Call . . . 19
4.3.2 Subversion Integration . . . 19
4.4 Wiki Workitem . . . 19
4.4.1 Origo API Call . . . 19
4.4.2 Drupal Integration . . . 20
4.5 Blog Workitem . . . 20
4.5.1 Origo API Call . . . 20
4.5.2 Drupal Integration . . . 20
4.6 Access Control . . . 20
4.7 Notification . . . 21
4.8 Workitem Retrieval . . . 21

5 Implementation Details 22
5.1 Releases . . . 22
5.2 Issues . . . 23
5.3 Project Creation . . . 24
5.4 Drupal Installation Profile . . . 24
5.5 Drupal Theme . . . 24
5.6 Drupal Core Changes . . . 25
5.6.1 XML-RPC . . . 25
5.7 Drupal Cron Job . . . 25

6 Deployment **26**
6.1 Dependencies 26
6.2 Installation 26

7 Future Work **28**

List of Figures **29**

Bibliography **30**

Chapter 1

Introduction

1.1 Goal

The goal of this project is to provide a web interface for the Origo software development platform [12] which is based on Origo Core [13]. It should provide all necessary functionality to manage a project in Origo using and extending use cases in Origo Core. Additionally a workitem system should be created displaying so called workitems on the Origo Home page. Workitems can be used by a developer to keep track of changes in his own and bookmarked projects.

1.2 Overview

This thesis is structured in several chapters. Chapter 2 discusses some general design concepts. The implementation of the Drupal modules is described in chapter 3, followed by the workitem implementation in chapter 4. Some other implementation descriptions can be found in chapter 5. How to deploy the system is documented in chapter 6.

Chapter 2

Design

2.1 General

The Origo web interface uses the free open-source content management system Drupal [1]. Drupal provides a very good extension system with themes, modules, hooks, etc. It also has a big user community and therefore many existing modules and a good documentation. Drupal also provides powerful tools like the Forms API which allows the creation of forms that can be designed using the theme system.
The Origo theme for Drupal is built using the theme system and an existing theme. It uses some PHP files defining the structure and CSS files to format the page.
To add the needed functionality Drupal can be extended with modules. These modules use hooks to interact with internal Drupal processes. With this system it is normally not necessary to change code in Drupal Core itself, however there is one exception (see Section 5.6).

2.2 Workitems

Workitems are some sort of notification, they help a developer to keep track of changes in his project. They can also be used by other users who want to be informed about the development of their favorite projects. Workitems are created on several key events in the system, like wiki page edits or Subversion commits. Some of them are triggered in Drupal, some within Origo Core use cases and some in Subversion hooks. Each workitem has its own icon which can be seen in figure 2.1.
Workitems are stored and managed within Origo Core and are therefore not limited to a specific project page. The idea is that a user can access

Figure 2.1: Workitem Icons

his workitem list on the Origo Home site no matter what project site he is currently on. Several Origo Core use cases are available to create, view and manage the workitems. To distinguish between new and old workitems a workitem is marked read when the user reads the workitem. For implementation details see chapter 4.

2.3 Drupal Sites

Drupal provides a sites system to host several sites with only one Drupal code base. Each project has its own directory in the *sites* directory and can have its own modules or themes. Modules that should be available in all projects are stored in *sites/all*. The correct site is determined by looking at the entered URL, if no match is found the default site is displayed. For Origo we do not have project specific modules, therefore all additional modules are located inside *sites/all*. To provide better maintainability the site specific settings file is modified to use an include file. This makes it easier to make changes to the settings file. Only the database settings remain in the project specific file.

2.4 Scalability

Origo Core [13] is designed to be very scalable. The web interface is also built to provide good scalability. For only a few projects both the web server and the database server can be on the same machine. To support more projects the database can be moved to another server. It is even possible to use several database servers because the database server can be set individually for each project instance. Another possibility is to use several web servers on different machines. Either with some sort of load balancing or using manually assigned servers. The implemented session system can handle logins over several web and database servers.

Chapter 3

Drupal Modules

3.1 Origo Auth

The Origo Auth module handles user registration, login, logout, password change and password reset. It also has a rewritten session handling and a XML-RPC wrapper included.

3.1.1 User Registration

When registering a user with the normal Drupal user registration, a user is created in the Drupal database belonging to the project page the registration form was filled out. For Origo we need a global user registration which creates an Origo user by using XML-RPC.

To achieve this we modified the existing user registration. Using the Drupal hook system this is possible without changing code in any of the Drupal core modules. First the *hook_form_alter* replaces the validate and submit functions of the registration form. The form now calls our own functions instead of the functions defined in the user module.
The submit function *origo_auth_register_submit* then makes an Origo API call to *internal_user.add* to create a new user. On success the function *origo_auth_authenticate* (see Section 3.1.3) creates the Drupal user and logs him into the system.

3.1.2 Session Handling

Drupal provides a simple session management using PHP session. This system works of course fine for single Drupal instances, but does not meet

all requirements for Origo.
For one thing we would like to keep the user logged in not only in the local project. If he goes to another project (ie. another Drupal instance) he is still within Origo and should be logged in. One way would be to store the PHP session ID in a cookie accessible in all projects. While this is possible for a single server environment, it does not work with multiple servers because the session itself would have to be transfered to the other server. Because of security reasons neither the storage of the session on the client is a solution. Another problem is that each Drupal instance has its own user database. So if a user is logged in into one project we cannot simply log him into another project instance because the user might not exist in the other user database. Using just one user database would limit scalability, and keeping them synchronized would be very complicated.
The third problem is that Origo itself also has a session system. If an Origo session expires we have to relogin to get a new session. Therefore we need the username and the password.

The solution we came up with is to store an additional cookie on the client. This cookie stores the Origo username and the encrypted password. With this information available in all Drupal instances we can simply log in the user in every instance using the login function.

The Origo session system is an extended Drupal Session System using PHP sessions and the additional cookie. The green part in figure 3.1 shows the session system in a flowchart. When accessing a page the system checks if the Drupal cookie and/or the Origo cookie are available.

- If both cookies are missing the user is just an anonymous user.
- If the Drupal cookie is missing and we only have the Origo cookie a login using *_sess_load_origo_user()* is performed.
- If the Drupal cookie is available and the Origo cookie is missing we remove the PHP session and log out the user in this instance. This means deleting the Origo Cookie performs a logout on all projects.
- If both cookies are available we check if they are valid and both for the same user. If so a login using the PHP session and the normal Drupal session system is performed. If the user data does not match a login using the Origo session and *_sess_load_origo_user()* is performed.

The above mentioned function *_sess_load_origo_user()* extracts the user and the encrypted password from the cookie. After decrypting the password *internal_user.login* is called using XML-RPC to log in the user into Origo and get the Origo session. The Origo session is stored in the Drupal user object. After the Origo login the user has to be logged in into Drupal. A check on

the local user database shows if the user is already available. If so, his data is updated, otherwise he is added to the database. To assign the correct access rights it is then determined if the user is an administrator, a project owner, a project member or just a normal Origo user. See the orange part in figure 3.1 for a graphical representation of the function *_sess_load_origo_user()*.
Because at the time the session code is executed most of the Drupal code is not yet loaded, we cannot use the integrated *xmlrpc()* function. We use the functions from the PEAR package XML_RPC [10] instead.

3.1.3 User Login

To intercept the Drupal login system the form validate callback for the login form is changed using the hook *hook_form_alter*. We use the validate handler instead of the submit handler in this case because the XML-RPC request validates the entered data and because we do not want to overwrite the rest of the login performed in the submit handler. The new validate handler calls *origo_auth_authenticate* function to perform the login on Origo. First a XML-RPC request to *internal_user.login* is executed. This call returns the Origo session which is stored in the Drupal user session. After the successful call the role of the user is determined using *authorization.is_allowed_project* XML-RPC requests. Finally the Origo cookie containing the username and encrypted password is created.

3.1.4 User Logout

As described in section 3.1.2 the complete logout in all project instances is performed by destroying the Origo cookie. Therefore we use *hook_user* to hook into the user logout and destroy the Origo session.

3.1.5 Password Change

The Origo Auth module also overwrites the Drupal password change. The existing password change would only change the password in the project specific Drupal database. For Origo we need to change the password directly in Origo using a XML-RPC request. Drupal has a password change field on the user edit page. This page also has some fields that cannot be used with Origo, so we use the hook hook_menu to intercept the original edit menu and create a new one currently only containing the fields that allow changing the password. The corresponding submit function sets a new password calling *internal_user.change_password* over XML-RPC.

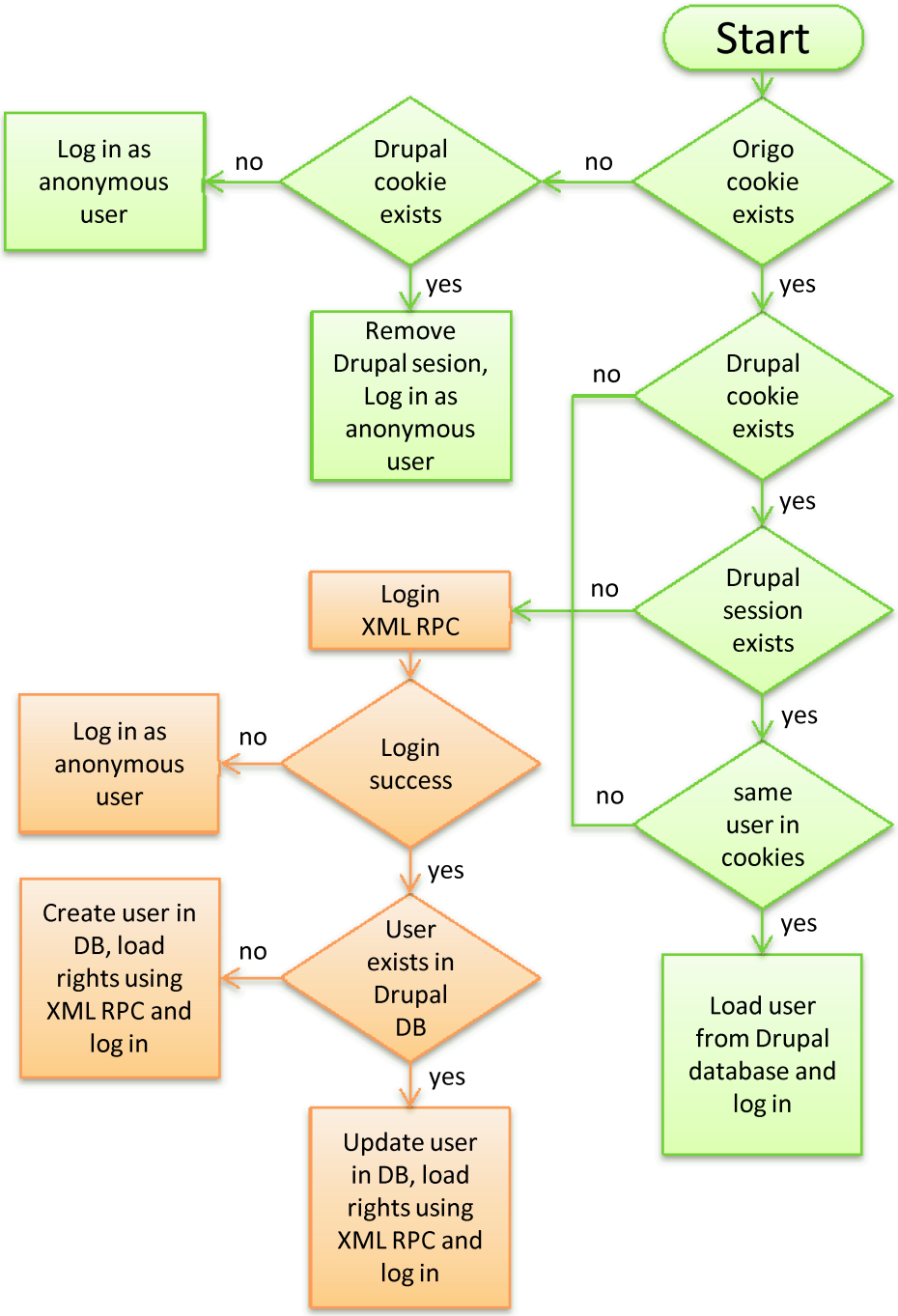

Figure 3.1: Session Handling and User Login

3.1.6 Lost Password

If the user forgets his password he needs a way to reset it. As for the user registration (see Section 3.1.1) we use the hook *hook_form_alter* to change the form validate and submit handlers of the existing password reset function. The new submit handler starts a XML-RPC request to *internal_user.reset_password*. This call starts the *USER_RESET_PASSWORD* use case in Origo Core which takes care of generating a new password and sending it to the user in an email. This generates also the password for all the external applications like SVN and FTP that the user belongs to.

3.1.7 XML-RPC Wrapper

Many of the Origo API calls require a valid session. This session is returned by the *internal_user.login* XML-RPC request and is only valid for a limited amount of time. If the session expires each request that requires a session will return an error. A relogin with *internal_user.login* is necessary to get a new session. To provide an automated relogin if a session is expired the Origo Auth module provides a wrapper to the Drupal *xmlrpc()* function.

There are two functions available: *origo_auth_xmlrpc()* which basically is the same as the original *xmlrpc()* and *origo_auth_xmlrpc_session()* which is used for API calls that require a session.
origo_auth_xmlrpc_session() adds the current Origo session as first argument and calls the Drupal *xmlrpc()* function. If this function returns an error indication the session is not valid, a relogin using *_sess_load_origo_user()* (see Section 3.1.2) is done. After the relogin *xmlrpc()* is executed again with the new session. If there is still an error it has to be a serious problem and the error is therefore given to the caller.

3.2 Origo Home

The Module Origo Home is the Origo web main module and provides besides several other functions features for workitems, project settings, releases and Origo administration.
Functions are organized in groups and moved to include files whenever it was possible. However all functions are still defined in the hook *hook_menu* which defines the path a function is available at.

Figure 3.2: Origo Home showing the workitems

3.2.1 Origo Home

Origo Home is the main page to view the workitems. A project tab is shown for each project a user is either developer or has bookmarked. Normally only the unread workitems are shown and they become read by following the link or using the checkbox. When selected to show all workitems the read workitems are shown too.

This page makes several XML-RPC requests to Origo Core. First the own and the bookmarked projects are retrieved using *project.list_of_user* and *user.list_bookmark*. With *workitem.list_projects* the workitems for these projects are retrieved (see Section 4.8) and listed in a table for each project. Depending on the workitem type the information shown in the table is extracted and displayed.
To enable the fast tab switch a JavaScript is used. Drupal has the JavaScript library jQuery [6] included, which makes it quite easy to add fancy effects or AJAX to the page. A JavaScript together with an AJAX request is also used to set the read state of a workitem. A click on the checkbox fires an asynchronous request which sets the state in Origo using XML-RPC. As soon as this request completes the style of the workitem is changed to look greyed out or if marked unread to look bold and black again.

The Origo Home page also has a link to mark all workitems of the project as read. This function calls *workitem.set_read_status_project*.

3.2.2 Workitem Subscription

The Workitem Subscription Settings provide a simple way to manage the workitem notifications (see Section 4.7). Using calls to *project.list_of_user* and *user.list_bookmark* the own and the bookmarked projects are retrieved. For each of these projects *user.list_workitem_subscription* retrieves the currently set notifications. After submitting the form an XML-RPC request with *user.set_workitem_subscription* sets the new notifications.

3.2.3 Project Bookmarks

Origo Home module provides a list of all bookmarked projects which it gets by a XML-RPC request to *user.list_bookmark*. There are also two menu paths defined in the hook *hook_menu* to add and remove bookmarks. These paths can be used as links to quickly add or remove a bookmark and are implemented by calling *user.add_bookmark* or *user.remove_bookmark* XML-RPC. Both adding and removing also sets or removes all the workitem subscriptions (see Section 4.7) for the corresponding project.

3.2.4 Project List

The project list simply lists all projects hosted on Origo. This list uses the internal API call internal_project.list to get the projects because the list should also be available to anonymous user which do not have session. Projects flaged hidden are not shown in this list. Most of the hidden projects are empty student projects created in courses.

3.2.5 User Key Request

External software using the Origo API requires the user to enter a user key instead of his password. This page provides a way to request a key using the API function *internal_user.generate_key*.

3.2.6 E-Mail Change

This page allows a user to change his e-mail address and is implemented using a XML-RPC request to *internal_user.change_email*.

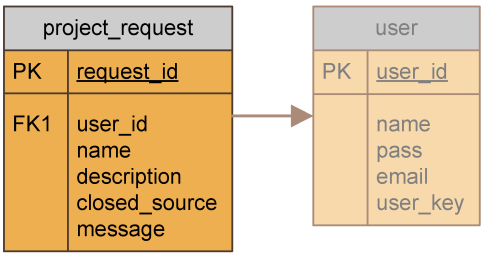

Figure 3.3: Project Request Table

3.2.7 Project Settings

On the Project Settings page a project owner can manage some project settings.
The members page allows adding and removing project members or owners. This is done by calling *project.change_group* with XML-RPC.
The description page allows changing the project description using the XML-RPC methods *project.retrieve* and *project.change_description*.
Changing the logo is possible on the logo settings page. Adding a logo uploads the picture to the local Drupal instance and uses the Drupal theme system to display the logo. Additionally the executed XML-RPC method *project.change_logo* updates the logo filename in Origo.

3.2.8 Project Creation Request

Every Origo user can request the creation of a new project. The project creation itself is done manually by an administrator for security reasons. However several features are implemented to automate this process.
A user can request a project on the Create Project page. He has to provide the name, a description and if it is a closed or open source project. A XML-RPC request to *project.request_add* checks if the project name is valid and still available and adds the project to a request table in Origo (see Figure 3.3). Using the Drupal mail function a mail to the administrators is sent including the entered data and a link to a creation form. This creation form defined in *origo_admin_create_project_form_request_page* is only available to administrators and loads the details for the requested projected with the XML-RPC request to *project.request_retrieve*. The administrator may now make changes to the entered data and the confirmation mail. Sending the form starts the project creation process which includes the project creation (see Section 5.3), adding the requesting user as project owner (see Section 3.2.8) and sending a mail to the user informing him about the created project.

3.2.9 Administration Menu

The admin menu provides an interface to the XML-RPC methods reserved for administrators.
There is a project list like the one open for all users (see Section 3.2.4). The difference to the open project list is the usage of the external API call *project.list* and that is also shows the hidden projects.
To send newsletters or important information to all users an administrator can use the mass mail function defined in *origo_admin_massmail_page()* which starts a XML-RPC request to *origo_system.mail_all*.
A project creation form allows the direct creation of a project (see Section 5.3) without using the request mechanism described in Section 3.2.8.
Finally the function *origo_admin_status_page* shows the result of the XML-RPC request to *origo_system.status* which returns some information about the running nodes.

3.3 Issue Tracker

The issue tracker module is the web front-end to the Origo issue system described in section 5.2. Issues are implemented in Drupal as a new node type and issue replies are simple comments.
The hook *hook_insert* is used to intercept node insertion and make a call to *release.add* via XML-RPC. This call returns the project specific issue id which is stored in an additional table *issues* in the Drupal database. The standard comment system is also modified in the hook *hook_form_alter* to execute the XML-RPC method *release.comment*

3.4 Developer Pages

The Developer Pages module is a simple module that adds the possibility to flag pages as private. Private pages can only be accessed by project members and can be used to store project internal information. Workitems created from private pages are also only visible to project members.
This module is a Drupal node access module and uses several hooks to perform its task. *hook_node_grants* is used to define the node access rights, *hook_nodeapi* is used to keep track of node inserts and changes to update the table containing all private pages. *hook_form_alter* is used to add a checkbox at the end of a node edit form (see Figure 3.4) giving the possibility to mark this node private.

Figure 3.4: Checkbox to flag a page private

3.5 Existing Modules

3.5.1 Captcha

Because spam bots are everywhere nowadays it is necessary to protect all functions that can be accessed without a valid login. This includes user registration and password reset. The Captcha module provides a simple math challenge a user has to answer. The protection is not as strong as it would be with an image captcha, but the image captcha module had several bugs which made it impossible to use. Fortunately at the moment the current system suffices.

3.5.2 Diff

Diff shows differences between node revisions. It adds a new tab on top of nodes like wiki pages and shows all changed word in a colored view.

3.5.3 Form Store

Provides form information to other modules and is needed by the Captcha Module. (see Section 3.5.1)

3.5.4 GeSHi Filter

A filter to highlight sourcecode using GeSHi. [2]

3.5.5 Google Analytics

The Google Analytics module is used to gather advanced web statistics using Google Analytics [3]. It works by including a JavaScript on top of each page and can therefore retrieve information that is not available in web server logs. The included JavaScript is hosted on www.google-analytics.com which turned out to be a bottleneck, therefore we modified the module.

Instead of including the script from www.google-analytics.com we used a local copy on our local server. A simple daily cron job (see Figure 3.5) is

```
wget http://www.google-analytics.com/urchin.js -q
  -O /data/www/origo/static/urchin.js
```

Figure 3.5: Cron Job Command for Google Analytics

scheduled to download the script to make sure the script is up to date in case Google releases a new version.

3.5.6 Google Co-op CSE

Google Custom Search Engine [4] is a service to include Google search on your on website. We use this service to provide an Origo wide search over all projects.

3.5.7 Image

Allows uploading, resizing and viewing of images.

3.5.8 Image Assist

This module allows users to upload and insert inline images into posts. It automatically generates an Add image link below text fields.

3.5.9 Pathauto

Provides a mechanism for modules to automatically generate aliases for the content they manage. This is used to generate wiki links.

3.5.10 PEAR Wiki filter

Filter which uses the PEAR Text_Wiki [9] package for formatting.

3.5.11 Tag Query Language

A nice tag query language. This can be used to write queries to retrieve nodes with specific tag. For example one could to write a query for all open issues assigned to him.

3.5.12 Wikitools

Provides helper functionality to have wiki-like behavior.

Chapter 4

Workitem Implementation

workitem_id The workitem ID, unique in the system

type The workitem type (1=Issue, 2=Release, 3=Commit, 4=Wiki, 5=Blog)

creation_time Timestamp when the workitem was created

project_id ID of the project this workitem belongs to

project Name of the project this workitem belongs to

user Name of the user responsible for the workitem creation

is_read 1 if the user has already read this workitem, 0 otherwise

4.1 Issue Workitem

Issue workitems are created for new issues and issue replies. The following additional information is included:

project_issue_id The issue ID, unique in the corresponding project

title The title of the issue

description Detailed description or text provided in the issue

is_new 1 if this is a new issue, 0 if it's a reply

url Link to the issue web page

4.1.1 Origo API Call

Issue workitems are created after inserting the issue itself in the *ISSUE_ADD* and *ISSUE_COMMENT* use cases, which are started by the XML-RPC methods *issue.add* and *issue.comment*. Because issues are already stored in

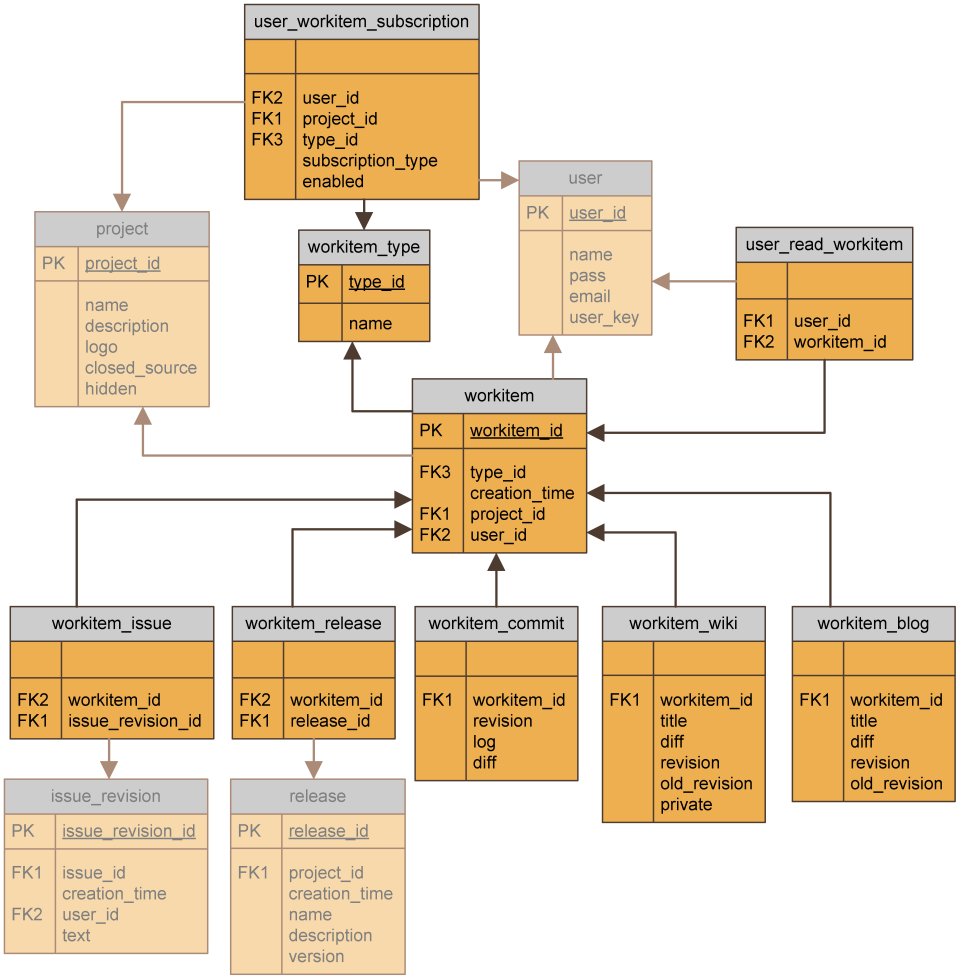

Figure 4.1: Workitem Tables

Origo (see Section 5.2) only the *issue_revision_id* needs to be stored in the table *workitem_issu*e. Figure 4.1 shows this relation.

4.1.2 Drupal Integration

The issue workitems are created on new issues and issue comments in the Issue Tracker module. (see Section 3.3)

4.2 Release Workitem

A release workitem is created on each new release. It contains the following information:

name The name of this release
description Detailed description
version Version of this release
url Link to the download page
file_count Number of files included in this release
file_name_X Filename of file X (X = {1 .. file_count})
file_platform_X Platform for file X

4.2.1 Origo API Call

When adding a release with the *release.add* API call Origo starts the *RELEASE_ADD* use case. After inserting the release itself it takes the ID of this releases and inserts a new workitem into the tables *workitem* and *workitem_release* as shown in figure 4.1.

4.2.2 Drupal Integration

When using the website for releasing files the workitem (and release) creation is triggered inside *origo_home_create_release_form_submit* in the *origo_home* module.

4.3 Commit Workitem

A commit workitem is created for each commit in the Subversion repository. It contains the following information:

revision The SVN revision associated with this commit

log Log describing the commit

url Link to the WebSVN page for this revision

diff Diff for committed files (this is truncated for large commits)

4.3.1 Origo API Call

Commit workitems are created in the internal XML-RPC method *internal_commit.add* which starts the *COMMIT_ADD* use case. As shown in figure 4.1 all data is stored in the table *workitem_issue*.

4.3.2 Subversion Integration

The commit workitem creation is triggered by a SVN post-commit hook. The used script is an adaptation of the standard commit mail script which uses XML-RPC instead of mailing the changes. The script gets the user, project, revision, commit log and generates a diff of all changes which are then used to call *internal_commit.add*.

4.4 Wiki Workitem

Wiki workitems are created for new and changed wiki pages and contain the following information:

title Wiki page title

diff Diff of wiki changes

revision Drupal node revision after change

old_revision Drupal revision before change

url Link to the wiki page

diffurl Link to the diff page for this wiki page

4.4.1 Origo API Call

The use case *WIKI_ADD* started in the XML-RPC method *internal_wiki.add* adds wiki workitems into the *workitem_wiki* table (see Figure 4.1). Besides a diff between the revisions which is generated using the PEAR Text_Diff class [8] the old and new node revision is stored.

4.4.2 Drupal Integration

Adding or editing wiki node types is intercepted in the Origo Home module (see Section 3.2) within the hook *hook_nodeapi*.

4.5 Blog Workitem

Blog workitems are created for new and changed blog posts and contain the following information:

title Blog title

diff Diff of blog changes

revision Drupal node revision after change

old_revision Drupal revision before change

url Link to the blog entry

diffurl Link to the diff page for this blog entry

4.5.1 Origo API Call

Adding blog workitems in Origo is similar to adding wiki workitems. It uses the API call *internal_blog.add* to store blog workitems in the table *workitem_blog*.

4.5.2 Drupal Integration

Adding or editing a blog node type is intercepted with the hook *hook_nodeapi* like for wiki workitems.

4.6 Access Control

While blog and release workitems are accessible for everyone, commit, wiki and issue items have an access control mechanism. Commit items for closed source projects are of course only visible for project developers to keep the source closed. Wiki pages and issues can be set private so we have to do the same with their workitems. If the corresponding wiki page or the issue is private the workitem is only visible for project developers.

4.7 Notification

Origo provides different ways to notify users about new workitems. First there is the workitem list on Origo Home (see Section 3.2.1). New workitems can also be queried via the API and there is a mail notification available.
A user can set how and for which workitem types he wants to get notified. This is done using the web interface (see Section 3.2.2) or directly using the API. The XML-RPC method to be used is *user.set_workitem_subscription*. The core use case *USER_SET_WORKITEM_SUBSCRIPTION* then adds the subscriptions into the table *user_workitem_subscription*. There is also a call *user.list_workitem_subscription* available to read out the current subscriptions.

4.8 Workitem Retrieval

To retrieve the workitems there is either the XML-RPC method *workitem.list* or *workitem.list_projects*. The workitem list on Origo Home is implemented with a call to *workitem.list_projects* which retrieves a given number of the newest workitems for each own and bookmarked project. The parameter *unread_only* is used to retrieve only unread workitems, otherwise read and unread will be retrieved. The method *workitem.list* also lists a given number n of workitems, but this method just returns a total maximum of n newest workitems in all own and bookmarked projects. If five workitems are requested and the first project already has ten new workitems then only five workitems of this project are retrieved and none from any other project.
A detailed single workitem is retrieved with a call to the XML-RPC method *workitem.retrieve*.
All retrieve calls implement the access control described in section 4.6 and the subscription settings described in section 3.2.2. So only workitems a user wants to see and is allowed to see are retrieved.

Chapter 5

Implementation Details

5.1 Releases

Releases are managed within Origo Core. A release consists of some general data and a number of files, where each file has a filename and a platform. This information is held in the two tables shown in figure 5.1. The files are stored on a web server.

To create a release the files have to be copied to a FTP server or uploaded using the web interface. The XML-RPC core method *release.add* can then be used to add the release. It starts the *RELEASE_ADD* use case which first adds the release to the database and then moves the files from the user FTP space to the download directory.

The two XML-RPC methods *release.list* and *release.retrieve* are used to get existing releases. Because the files are stored on a web server they can be downloaded using any available download program.

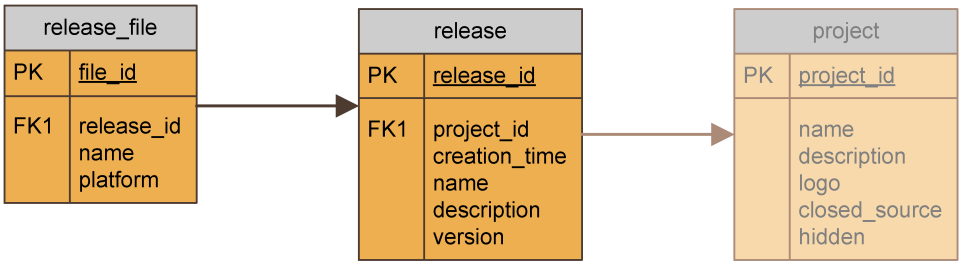

Figure 5.1: Release Tables

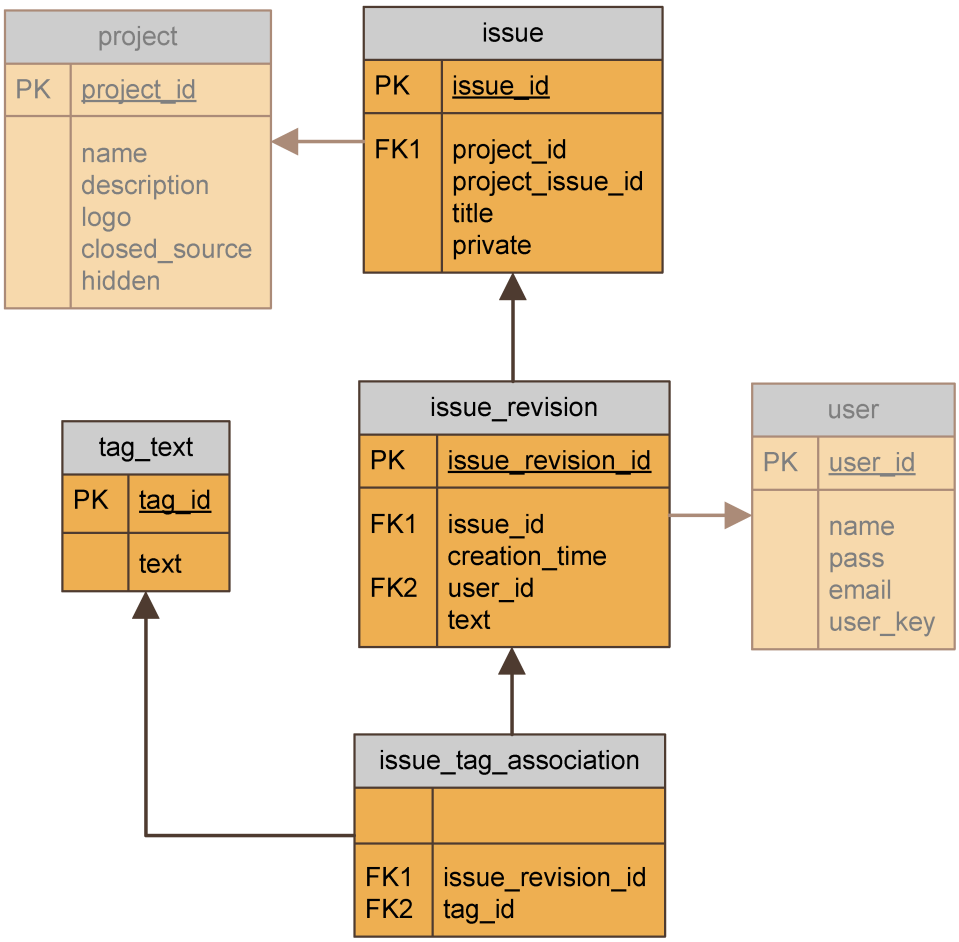

Figure 5.2: Issue Tables

5.2 Issues

Issues are stored in the four tables shown in figure 5.2. General issue information is held in the table *issue*, the issue text and each reply is held in *issue_revision*. Issue states are completely handled with tags. Open issues just have another tag than closed issues. Tags are stored in the *tag_text* table on their first use and associated to issue revisions with the *issue_tag_association* table.

A call to the XML-RPC method *issue.add* is used to add a new issue. The return value is an integer with the assigned issue ID. Issue replies are added with *issue.comment* using the given issue ID.
To retrieve issues there are the two methods *issue.list* and *issue.retrieve* available.

5.3 Project Creation

Creating a new project includes several steps and is started by executing a *project.add* XML-RPC request. This starts the Origo Core use case *PROJECT_CREATE* in which the project is created in the Origo database and two scripts are started. Using the config node first a simple shell script creates the Subversion [11] repository. Then a second script is called to create the Drupal project.
This PHP script creates the Drupal project database, a database user and sets the permissions on the database. Then it creates the directories in the Drupal sites directory (see Section 2.3) and copies the initial files into it. The Drupal installer is executed by sending the POST data normally entered in a form directly to the script. As PHP has no built in function to send POST data with an HTTP request the PEAR package HTTP_Request [7] is used. Finally some modifications to the created project are made. This includes modifications where the project name and ID are needed and the modification of the settings file to have all non-project specific information in an include file.

Two important variables often needed in XML-RPC methods are the project ID and the project name. These two variables are set in the creation process and stored in the Drupal variable system as *origo_project_name* and *origo_project_id*. The function *variable_get* is used to retrieve these variables whenever needed.

5.4 Drupal Installation Profile

Drupal provides installation profiles to specify how a new instance is created. Such a profile is used to create the standard Origo project instance. The file specifies which modules to load, their settings and some example content.

5.5 Drupal Theme

The Origo theme is based on a standard Drupal theme and modified at a few points. To create the illusion of a page which has always a height of 100% a background image is used.

5.6 Drupal Core Changes

Despite the fact that Drupal has a very good extension system with modules, hooks, etc it was necessary to make changes to Drupal core code.

5.6.1 XML-RPC

Drupal has a built-in XML-RPC client which can be used to access Origo Core. It works by calling the function *xmlrpc*. The return value is either the result provided by the server or FALSE if an error occured. Drupal documentation states that one should check *xmlrpc_errno()* or *xmlrpc_message()* if the return value was FALSE.

Unfortunately there is no way to reset the XML-RPC error object *xmlrpc_error*. A second call to *xmlrpc* does only change *xmlrpc_error* if the XML-RPC call fails, on success *xmlrpc_error* is left untouched in its error state. Several texts on the Drupal page comment this issue with the fact that *xmlrpc_error* has only to be checked if the call returned FALSE and in this case *xmlrpc_error* would be set to a new value anyway. This statement is not correct for XML-RPC calls returning a boolean value. It is not clear if a return value FALSE is the correct server return value or indicating an error. On the first call this is not a problem because *xmlrpc_error* is not set yet. But if such a boolean call is used after a failed call then we have no way to say if the FALSE was an error or not, because *xmlrpc_error* is still set from the first failed call.

The only way to fix this issue was to make a modification to Drupal Core. We simply created a function to reset *xmlrpc_error*. This call is then called at the beginning of the function *xmlrpc* which resets the error with each new call and therefore solves our problem.

5.7 Drupal Cron Job

Drupal uses a cron job to call a specific PHP script. Modules can then use a hook which is called in this cron PHP. In the Origo setting there is such a file for each project. To combine all these scripts a helper script loops through all projects and calls the PHP script for each project. The project list is retrieved with an XML-RPC request.

Chapter 6

Deployment

6.1 Dependencies

Before Origo Web can be installed a few dependencies have to be fulfilled. Here is a list of the dependencies:

- Apache web server with mod_rewrite
- PHP 5
  - PHP memory_limit set it to at least 16 MB
- PEAR with the packages:
  - HTTP_Request $\geq$ 1.4.1 stable
  - Text_Diff $\geq$ 0.2.1 beta
  - Text_Wiki $\geq$ 1.2.0 stable
  - Text_Wiki_Mediawiki $\geq$ 0.1.0 alpha
  - XML_RPC $\geq$ 1.5.1 stable
- php-geshi
- php-mcrypt
- working Origo Core system

6.2 Installation

Detailed installation instructions are available in the Origo Wiki. [5]

1. Checkout Drupal from CVS
2. Checkout additional Drupal modules CVS
3. Checkout Origo Web
4. Copy Origo Web files into Drupal installation (using *examples/release-files.sh*)
5. Edit settings in *sites/all/settings.inc*

Chapter 7

Future Work

The current system as it is now provides a good base for further features and improvements.

One possible additional feature would be a ranking system for projects. This ranking could be based on the number of workitems with different weights according to the workitem type.

Instead of using Google CSE it would be possible to develop an own Origo wide search, perhaps including the above mentioned ranking system.

There is also the possibility to extend the current theme with new features like selectable color for project owners.

List of Figures

2.1 Workitem Icons 3

3.1 Session Handling and User Login 7
3.2 Origo Home showing the workitems 9
3.3 Project Request Table 11
3.4 Checkbox to flag a page private 13
3.5 Cron Job Command for Google Analytics 14

4.1 Workitem Tables 17

5.1 Release Tables 22
5.2 Issue Tables 23

Bibliography

[1] *Drupal, open source content management platform.* http://drupal.org.

[2] *GeSHi - Generic Syntax Highlighter.* http://qbnz.com/highlighter.

[3] *Google Analytics.* http://www.google.com/analytics.

[4] *Google Custom Search Engine.* http://www.google.com/coop/cse.

[5] *Installing Origo web.* http://origo.ethz.ch/wiki/installing_origo_web.

[6] *jQuery: The Write Less, Do More, JavaScript Library.* http://jquery.com.

[7] *PEAR Package HTTP_REQUEST.* http://pear.php.net/package/HTTP_Request.

[8] *PEAR Package Text_Diff.* http://pear.php.net/package/Text_Diff.

[9] *PEAR Package Text_Wiki.* http://pear.php.net/package/Text_Wiki.

[10] *PEAR Package XML_RPC.* http://pear.php.net/package/XML_RPC.

[11] *Subversion, version control system.* http://subversion.tigris.org.

[12] Till G. Bay. The origo software development platform. ETH Zürich, 2005.

[13] Patrick Ruckstuhl. Origo core: Middleware and controller for origo. Master's thesis, ETH Zürich, July 2007.