Projekt **Jake**

Jake Specification

Kurzzusammenfassung:
This document contains the SRS, technical and UI specification for Jake 1.0

Autor:	Simon Wallner
Review:	-
Gruppe:	Gruppe 3950

Nr	**Datum**	**Autor**	**Änderung**
1	2008-11-09	Simon Wallner	document created

Inhaltsverzeichnis

Teil I

Software Requirements Specification

1.1 Introduction

Jake is a application that simplifies sharing files in small project groups. It's main focus lies on sharing and replication of files and notes over a network.

1.1.1 Core Audience

Jakes core audience are people with minimal to medium IT training, who are comfortable working with internet and email applications

1.1.2 Example Usage Scenario

A project group of 6-12 project members that shares about 5-100 files (ordinary office files mainly). All project members have internet access and are online most of the time, while they are working. Usually only one person works on a file at one time, therefore merging conflicts are rare.

This model scenario may be a small corporate project, or a student project. It is especially useful for projects in distributed teams like architects working in Vienna and Abu Dabi.

1.2 Terms and Definitions

This sections contains terms and definitions that are used in this document. If you are already familiar with Jake, you might want to skip this section for now and come back later to clarify certain expressions.

Jake The Name of the application this software requirements specification is written for.

project A project is the core unit of Jake. It consists of *project members*, *project files* and *project notes*.

project member a project member may share project files with other project members within a project. It is uniquely identified by its *user ID*.

project files Files that are shared within a project.

project folder Every project is bind to one local project folder in which the project files are stored.

local files files that reside in the project folder but are not shared.

project notes Besides files, a project may contain notes. Notes are associated to the whole project (in contrary to a note to a file)

local notes notes that are not shared.

file system the service provided by the operating system to movev, copy or remove files and folders

affecting the project a certain action is said to be affecting the project if the action is visible to other project members. If Alice modifies a file, this action does not affect the project (i.e no one knows that Alice made a modification) unless Alice decides to announce it.

trust relation If the *trust relation* holds true for two project members, it means that they may directly or indirectly exchange files. It is a transitive ($a \sim b \wedge b \sim c \Rightarrow a \sim c$) but not symmetric ($\exists a, b : a \sim b \wedge b \not\sim a$) relation.

member list the member list is a set of *project members* a user explicitly trusts.

explicit trust relation if one project member explicitly trusts another project member. It is written as *Alice* $\sim^*$ *Bob*. It is a transitive ($a \sim b \wedge b \sim c \Rightarrow a \sim c$) but not symmetric ($\exists a, b : a \sim b \wedge b \not\sim a$) relation.

neighborhood of project member a_0 all project members that a_0 explicit trusts: a_0, $\{a_i : a_0 \sim^* a_i\}$

project graph all project members along with their explicit trust relations form the project graph

web of trust in a web of trust each member of the web trusts each other member.

Besides the above definitions this section uses keywords proposed in RFC 2119[1]. In general, every part of this specification MUST be fulfilled except the parts that are explicitly marked as OPTIONAL or similar.

1.3 Functional Requirements

1.3.1 Manage Projects

Single Instance with many Projects

At any time, there MUST NOT be more than one instance of Jake running per user session. Within this single instance many projects are supported. The number of concurrent *projects* MAY be limited Every project MUST have its own *project folder*. Files MUST only be associated with one project, therefore sharing a file with multiple projects is not supported.

If Jake is started the last application state is restored, (as far as one exists), i.e. the last opened projects are reopened on start.

[1] `http://www.ietf.org/rfc/rfc2119.txt`

Creating a new Project

In order to create a new project, the user specifies a *project folder* and a *project name.* All existing files in the project folder become *local files.* No *user ID* and network connection is required to create a new project but the project must be bind to a *user ID* in order to invite other users and start sharing files and notes.

1.3.2 Manage Files

In Jake all files (i.e. *local* and project files) are organized hierarchically in folders. Empty folders MUST be allowed. New folders may be created within Jake. Files may be moved within the project folder or be deleted from it. If a file is deleted from the project folder, it MAY either be moved to the OS trash or to a dedicated trash folder inside the project folder. One of this options MUST be chosen for a concrete implementation. If the file is moved to a dedicated trash folder, this trash folder MUST be hidden from the user within Jake, but the user MUST BE able to examine its contents, restore files and empty it.

Files may be added to the project either by moving them into the project folder via the file system or by *importing* them in Jake. These files are threated as *local files* as long as they are not announced.

Changes in the file system MUST be reflected in the application within an acceptable latency

If a filename violates filename contraints (length, illegal characters) the file name MAY be altered in order to conform with the constraints. This SHOULD happen transparently and obvious to the user (i.e. Übung 1.pdf → Uebung_1.pdf). The user MAY be notified about it.

Batch operations MUST be provided where applicable.

Cached files from the operating system (e.g. `.DS_STORE`, `thumbs.db`, etc.) MUST NOT be part of a project and are therefore hidden from the user.

Announcing Files

Only *project files* are shared among the *project members.* In order to contribute a *local file* to the project it must be *announced.* Changes or new files do not affect the project unless announced.

If the user announces a file (or changes to it) he/she may specify a *announce message* to provide additional informations for other project members about the modification/new file.

In order to delete a file from the project the delete operation is announced to the other project members.

1.3.3 Manage Notes

In Jake, a *note* is associated generally with a project or with a specific file. Both *local* and *project notes* exist, where *local notes* are notes that are not shared. They are NOT hierarchically ordered and reside in one common location within Jake. Notes are only accessible through Jake. Notes may be created, displayed, modified and deleted. A note consists of a *title* and a *body*, where the first line of body represents the title. The length of the title MAY be limited. The length of the body MAY be limited.

Announcing notes works analogous to files.

1.3.4 Tags

Both files and notes may have a *set*(in the mathematical sense; no duplicate elements, unordered) of *tags*. A *tag* is a word. Constraints (length, no spaces, etc.) MAY be enforced on tags. Changes to tags immediately *affect the project*.

1.3.5 Project Members and the Web of Trust

All project members are equal, no one has special rights or stands above others.

All project members in a project along with their *explicit trust relation* form the *project graph*. The*explicit trust relations* are represented by the edges and the project members are represented by the vertices.

Each project member has a set of project members that he explicitly trusts. It is called the *member list*. If two project members have each other in their *member list* an explicit *trust relation* is established.

Every connected *project graph* (or connected subgraph) represents a *web of trust*. In Jake, every participant in this web of trust may receive and send files from/to any other participant in the web, either directly or indirectly.

In general the *project graph* is not complete.

Adding and Removing Users

A user may be added to a project if he/she establishes an explicit trust relation to a project member by getting invited by an already existing project member. When the new user accepts the invitation, he/she becomes a regular project member. Every project member may invite users.

Removing project members is not easy. A project member may only be completely removed from a project if he/she is not connected to the project graph anymore. This means, no other user has a trust relation to that user.

For Example: $Alice \sim^* Bob$, $Bob \sim^* Eve$ and $Alice \sim^* Eve$. If Alice removes Eve from her *member list* she might still be able to exchange files with Eve even after she removed her from her *member list*. Users SHOULD be made aware of the web of trust and its implications.

Global Member List

Jake MUST provide a *global member list*. This list contains all project members of the project i.e. all project members that are connected to the project member who creates the list. This list MUST be created with best effort, it is not reliable. This fact MUST be clearly communicated to the user.

Add/Remove Member Alert

A project member SHOULD be alerted if someone in the project adds or removes a member. Best effort, non reliable.

1.3.6 Auto Add/Remove

A project member may set an *auto add/remove flag* at every project member in his/her member list (written as $\sim^{\pm}$). Lets make things clear with a short example: $Alice \sim^{\pm} Bob$. If Bob adds Carol Alice automatically adds Carol as well. If Bob removes Carol from his list Alice does that as well. The following problem may occur when removing project members: $Alice \sim^{\pm} Bob$, $Alice \sim^{\pm} Carol$, $Bob \sim Eve$ and $Carol \sim Eve$. Now Bob removes Eve from his member list. In that case Alice does not remove Eve. *Auto add* is always stronger than *auto remove*. Alice does not remove Eve unless Carol removes Eve and therefore no *auto add* relations persist.

The *auto add/remove relation*: $a \sim^{\pm} b$ is reflective: $a \sim^{\pm} a$, transitiv: $a \sim^{\pm} b \wedge b \sim^{\pm} c \Rightarrow a \sim^{\pm} c$ but not symmetric: $a \sim^{\pm} b \not\Rightarrow b \sim^{\pm} a$

Attention: under certain conditions auto add/remove might become a WMD with an unpredictable outcome due to race conditions.

1.3.7 Sharing and Synchronization

A peer only directly communicates with its neighbors. A peer may download files/notes from its neighbors. A peer MUST know which files are available from its neighbors and offer this list to the user. If the user announces a file/note, the peer MUST notify its neighbours about it.

The peer MUST always offer the user to download the latest available version of a file.

A *synchronization conflict* may occur under the following condition: A project member modifies a project file. If another version of the same file becomes available that is newer (timestamp) then the base version of the modified file (i.e the version of the file before it was modified) a conflict occurs.

Conflict Resolution

Both versions of the conflictuous file and additional information (last edit, last edited by, size, etc) are offered to the user. The user may examine both files and do either one of the following options:

- choose the locally modified file and overwrite the remote version

- choose the remote version and discard the local changes by updating the local file to the remote version
- open both files and merge the files manually, then try to announce the merged file. A new conflict may occur.
- postpone the conflict resolution to a later time. The Process of conflict resolution may be restarted at any time.

Soft Lock

A file or note may be *soft locked*. A *soft lock* consists of a *locking message*. This *locking message* MAY be limited in length. Every user may append, modify or delete a soft lock.

Whenever an action is to be performed on a locked file or note that may change the file/note, the user MUST be prompted to review this operation and either decline or approve it. The locking message MUST be displayed along the prompt.

As with everything, the propagation of the soft lock can only be done with best effort.

1.3.8 Application Start

Jake may be started without any project loaded and without being logged in. In this case, new projects may be created, the user may create a new jabber account, the user may login into the network or receive invitations to projects (requires login)

1.3.9 Searching

Jake MUST provide appropriate means for searching files, notes, tags, project members.

1.3.10 i18n

Jake SHOULD provide appropriate means to support internationalization, in terms of a user language specific interface.

1.3.11 Preferences

Project specific preferences are stored along the corresponding project. Global preferences MUST be stored in one global file/location. User credentials MUST NOT be stored along the project so that the user might not give away his/her credentials with the project accidentally.

Preference files SHOULD be hidden from the user.

1.3.12 Creating a new Jabber Account

Jake MUST provide sufficient means to create new Jabber accounts from within the application.

1.3.13 General Constraints

Characters

Jake MAY enforce additional constraints on strings if necessary. Though these constraints SHOULD affect the user as little as possible (replacing illegal characters in file names with '_' is ok, only allowing `[a-zA-Z]` is not!)

Exceptions

Jake MUST be able to tackle exceptions like `not enough disk space`, `no read/write access to file`, missing config files and things alike in a convenient manner.

1.4 Change History

This section contains changes to this document. Each change must be described detailedly.

Rev.	Date	Author	Change
1	2008-11-9	Simon Wallner	created document