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 an 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 connected to the internet 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 Dhabi.

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.

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. He/she is uniquely identified by its *user ID*.

user A user of jake, who is not part of a certain project.

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 move, 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.

announce Mark a file or note as "ready to be distributed".

announce message When announcing a file or note a *announce message* may be specified to provide additional information about the changes to the file, or the file itself.

member list The member list is a set(unordered, unique) of *project members* a user explicitly trusts.

explicit trust relation the *explicit trust relation* ($Alice \sim Bob$) holds true, if *Alice* explicitly trusts *Bob*. It is a neither transitive ($\exists a, b, c : a \sim b \wedge b \sim c \not\Rightarrow a \sim c$) nor symmetric ($\exists a, b : a \sim b \not\Rightarrow b \sim a$) relation.

transitive trust relation The *transitive trust relation*($Alice \sim^+ Bob$) holds true, if there exists a directed *path* ($Alice \sim^+ Bob :\Leftrightarrow \exists a_1, a_2, .., a_i : Alice \sim a_1 \sim .. \sim a_i \sim Bob$, i.e. the transitive hull) from *Alice* to *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: $\{a_i : a_0 \sim_i^a\}$

directed project graph All project members along with their explicit trust relations form the directed project graph

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

peer A *peer* is the abstraction of the client associated to a specific project, seen from a networking perspective.

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

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 several 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.

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 bound to a *user ID* in order to invite other users and start sharing files and notes.

Deleting a project

Projects may be deleted within Jake. If a project is deleted all files relating to this project MUST be either deleted from the file system or moved to the OS trash folder, whereas the latter method SHOULD be preferred. The user MUST be prompted to review the deletion and be either approve or decline it. The consequences of this operation MUST be clearly conveyed to the user.

1.3.2 Manage Files

In Jake all files (i.e. *local* and project files) are organized hierarchically in folders. New folders may be created within Jake. Files and folders may be moved within the project folder or be deleted from it. If a file/folder 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/folder 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/folders 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 *local files* as long as they are not announced.

Changes in the *project folder* MUST be reflected in the application within an acceptable latency.

Local files MAY violate filename constraints (length, illegal characters), *project files* MUST NOT. If a filename violates filename contraints 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). If filenames are changed, the user MUST be prompted to review the change. He/she may either approve or decline the change.

Batch operations MUST be provided where applicable.

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

Announcing Files/Notes

Only *project files/notes* are shared among the *project members*. In order to contribute a *local file/note* to the project it must be *announced*.

Changes or new files/notes do not affect the project unless announced.

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

In order to delete a file/note 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. 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 only of a *body*. The first line of a note SHOULD be emphasized. The length of the note MAY be limited.

Announcing notes works analogous to files.

1.3.4 Tags

Files have a *set* (in the mathematical sense; no duplicate elements, unordered) of *tags*. Tags for notes are OPTIONAL. A *tag* is a string. Constraints (length, no spaces, illegal characters, etc.) MAY be enforced on tags. Changes to tags immediately *affect the project*, and therefore don't need to be announced.

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 directed *project graph*. The *explicit trust relations* are represented by the edges and the project members are represented by the nodes.

Each project member has a set of project members that he explicitly trusts. It is called the *member list*.

the web of trust section will be reformulated shortly

In general the *project graph* is not complete.

Adding and Removing Users

A user is added to the project if an existing project member explicitly trusts him/her. A user remains a project member as long as at least one project member explicitly trusts this user.

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 an *explicit trust relation* to that user ($\nexists a_i : a_i \sim^+ a_0$)

For Example: $Alice \sim^+ Bob$, $Bob \sim^+ Eve$ and $Alice \sim^+ Eve$. If Alice removes Eve from her *member list* Eve still remains a project member, as long as there is at least one project member that trusts Eve ($\exists a_i : a_i \sim^+ Eve$). 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 transitive ($a \sim^{\pm} b \wedge b \sim^{\pm} c \Rightarrow a \sim^{\pm} c$) but not symmetric ($a \sim^{\pm} b \not\Rightarrow b \sim^{\pm} a$)

1.3.7 Sharing and Synchronization

A peer only communicates with other peers from the same project. A peer MUST only download files/notes from peers that are in its *member list*. A peer MUST know which files are available from the peers in the *member list* and offer this list to the user.

Folders are not synchronized explicitly. The user may not synchronize empty folders. Folders are only synchronized if they are needed by the contained files. (e.g if a file `foo/bar.txt` is created and synchronized, the folder `foo/` will be created to contain the file `bar.txt`. Empty folders MUST NOT be deleted automatically.

A peer SHOULD always deliver any file/note to another peer in the project if requested. It MUST provide information about the available files to other peers from the project. It MUST NOT deliver any data to peers that are not in the project (at the time of the request, as far as the peer knows)

The peer MUST always offer the user to download the latest available version of a file. New versions of notes MUST always be downloaded as soon as they become available.

Deleting Files and Notes

If a user announces a delete operation for a file/note every peer that receives the delete operation MUST delete the file/note. Conflicts may occur in the process of deleting files, folders and notes.

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 perform either one of the following actions:

- 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.

Additionally both con conflicting version may be saved to a different location.

Resolving conflicts in notes is limited to choosing one version or resolving the conflict later.

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 Pause/Resume a Project

Projects may be paused and resumed. If a project is paused, changes MUST NOT be synchronized with other project members, changes to the project folder in the file system MUST NOT be monitored. The user may not work on a paused projects in Jake (no adding, no changing, no announcing, no browsing, no whatever...). The user may delete a paused project.

Paused projects may be resumed at any time.

1.3.10 Open/Close a Project

Projects may be opened and closed. As in other applications a project must be opened in order work on it. Jake is not aware of closed projects, as if the project never existed.

1.3.11 log in/out

The user may log on/off to/from the network service (XMPP in that case).

1.3.12 Searching

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

1.3.13 i18n

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

1.3.14 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.

1.3.15 Creating a new Jabber Account

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

1.3.16 General Constraints

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.

1.3.17 Error Messages

Jakes error messages should be simple, clear and easy to understand. Problems should be described from a user's perspective and SHOULD NOT use terms and models that lie outside the user context (i.e. "Network Error: could not open port 666"). If the user wants additional technical information it SHOULD be presented along the error message.

1.3.18 Log Files

Jake MUST provide log files for files, notes and the whole project. A Log file is a chronological list that contains operations performed, the user who caused the operations, when the operation happened and which parts of the project have been influenced. It MAY also contain additional information. Compilation of a log is best effort, non reliable.

1.4 Change History

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

Rev. 1, 2008-11-9, Simon Wallner

created document

Rev. 2, 2008-11-15, Simon Wallner

Updated the SRS according the results of the SRS feedback meeting. Some parts are still unclear

Rev. 3, 2008-11-19, Simon Wallner

Updated the SRS according the last meetings and feedback. The following has changed:

- tags for notes are now OPTIONAL
- added option for saving conflicting files to another location
- fixed some typos
- clarified the definition of *transitive trust relation*, changed the notation from $\sim^*$ to $\sim^+$. It might be more easily mistaken for the *auto add/remove relation* but it is now consistent with the transitive hull syntax used in wikipedia.
- note MUST always be downloaded if available, aka autopull notes

Some points are still unclear:

- Version conflicts with deleted files are not specified.
- MUST delete operations always be announced immediately/automatically? aka auto announce delete?
- the web of trust section is still unclear and I'm not sure if we should keep it or drop it.
- user switching: what happens if the user logs off from the network? Will all associated projects be closed? How about working offline?

Rev. 4, 2008-11-24, Simon Wallner

Updated SRS according to a mail from Johannes: Folders are not synchronized directly anymore. It is not supported to share empty folders. Folder may not be tagged.