Review der technischen Archtiktur

Kurzbeschreibung:
Die technische Architektur von Jake wurde einem Review unterzogen. Außerdem wurde ein erster Überblick über die notwendigen Änderungen für die Umsetzung der neuen Features von Jake erstellt.

Autor:	Dominik Dorn, Christopher Cerny
Review:	
Gruppe:	QSE/ASE04

Nr	Datum	Autor	Änderung
1	16.10.08	DD, CC	Erste Version TXT
2	20.10.08	DD	Umwandlung in RTF, Prosa Text
3	24.10.08	DD	Umstrukturierung gem. Anforderungen JB
4	26.10.08	CC	Abschnitt Änderungen für neue Version anhand Vorlage von DD und JB erstellt.
5	26.10.08	DD	Ergänzungen der Änderungen von CC mit Überschriften zur leichteren Strukturierung der Informationen.

1. Aktuelle Probleme

1 Core/JakeGUIAccess-Interface

Das JakeGUIAccess-Interface in der aktuellen Fassung ist zu groß.

Patterns

Verwendete Patterns:
Observer (setConflictCallback, jakeObjectStateChangeListener)

Verbesserungswürdige [V]/ auszutauschende Patterns [A]:
[V] Observer (setConflictCallback, jakeObjectStateChangeListener)

Interface

Status: [OK]/[NOK]
[NOK] Interface ist zu groß, keine Wiederverwendbarkeit gegeben, überarbeitung dringend notwendig.

Datentypen [OK]/[NOK]:
[NOK] Die verwendeten Datentypen (List, Set, boolean, Boolean, etc.) müssen auf ihre korrekte Verwendung überprüft werden. Typkonvertierungen sowie primitive Datentypen (boolean, int) sollen vermieden werden.
[NOK] Es befinden sich statische Daten für die Synchronisation im Interface.

Datenfluss [OK]/[NOK]:
[?] Bedarf einer genaueren Untersuchung

Logikaufteilung

Das Interface muss überarbeitet werden, hierzu müssen große Teile der Logik in eigenständige Komponenten verschoben werden.
Diese wären:

1. Sync-Komponente / Locking-Komponente, etc.
2. Authorisierungs-Komponente
3. Konfigurations-Komponente
4. Logging-Komponente
5. Messaging-Komponente
6. Projektverwaltungs-Komponente
7. FileObject-Komponente
8. ProjectMember-Verwaltungs-Komponente

Ziel muss es sein, möglichst kleine, leicht testbare Interfaces zu erhalten.
Eine genaue Beschreibung der einzelnen Komponenten folgt weiter unten.

Caching

Die derzeit vorhandenen Caching-Mechanismen zur Entlastung der Datenbank sind unzureichend.
Aktuelle Probleme:

1. Resizen eines Fensters, teilweise auch das "Überfahren" einer JTable mit der Maus erzeugt Requests im Core
2. Diese Requests werden entweder direkt an die DAOs weitergegeben (hohe Datenbanklast) oder jedes mal neu berechnet (SyncStatus) → hohe CPU-Last

Ziel:
Jede Komponente muss einen (privaten) Object-Cache sowie eine refresh()-Methode implementieren. Das Holen/Berechnen neuer Daten darf nur beim Aufruf der refresh-Methode erfolgen oder wenn die Daten noch nicht verfügbar sind.

Die Verwendung bereits bestehender Caching Lösungen ist zu überlegen,
vgl. http://www.developer.com/open/article.php/3700661

Performanz

Die Performanz des derzeitigen Systems ist unzureichend.
Dies hat mehrere Gründe:

1. Unzureichendes Caching (siehe oben)
2. Unzureichende/unperformante Algorithmen
    1. Hashes für Files
    2. Berechnung des Sync-Status
3. Verwendung falscher Datentypen für File-Operationen (Byte-Arrays anstatt FilePointer/Streams, etc.)
4. Verwendung falscher Datentypen in Interfaces → erfordert neuerliche Umwandlung in z.B. GUI
5. Unvollständiges Datenbankmodell, mangelhafte DAOs
    1. Datenbankstruktur nicht auf Performanz ausgelegt
    2. Keine Indizes definiert
    3. SQLs zu generisch
6. Mangelnde Vertrautheit mit Swing für das GUI
    1. unnötige Methoden-Aufrufe in den Core-Komponenten (z.B. beim Fenster resizen)
    2. Falsche Verwendung der Datentypen bei JTables (JtableModel)
7. Unvollständige UML-Diagramme / POJOs
    1. Fehlen der Size eines Files
    2. Keine Möglichkeit den SyncStatus eines Files/Note mitzuspeichern
    3. Fehlen der Überschrift einer Note → Extraktion erfolgt aus GUI

2 ICS

Das ICS in der jetzigen Fassung muss durch eine konkrete Implementierung ersetzt werden. Unperformante Methoden müssen abgeändert werden.

Patterns

Verwendete Patterns:

Observer (IObjectReceiveListener, IOnlineStatusListener)

Verbesserungswürdige [V]/ auszutauschende Patterns [A]:
[V] Observer (IObjectReceiveListener, IonlineStatusListener)
→ Listener Pattern?

Interface

Status: [OK]/[NOK]
[NOK] Unnötige Überladung von Methoden (isLoggedIn)
[NOK] Fehlerhafte Verwendung von Datentypen → Performanz
[NOK] Methoden die eventuell nicht von allen Backends unterstützt werden (Firstname, Lastname)
[NOK] Check der Konsistenz einer UserID im Interface anstatt eigenen Datentyp UserId für jedes Netzwerk inkl. Konsistenzcheck zu haben (isOfCorrectUseridFormat)
[NOK] getServiceName() → ServiceType getServiceType()

Datentypen [OK]/[NOK]:
[NOK] getServiceName() sollte einen ServiceTyp zurück geben.
[NOK] sendObject() sollte einen User übergeben bekommen.
[NOK] sendObject() sollte einen FilePointer übergeben bekommen, kein byte[].
[NOK] sendMessage() sollte ein User-Objekt uebergeben bekommen
[NOK] sendMessage() sollte ein Message-Objekt uebergeben bekommen
[NOK] login() sollte ein User-Objekt uebergeben bekommen

Datenfluss [OK]/[NOK]:
[?] Bedarf einer genaueren Untersuchung

Caching

Nicht notwendig.

Performanz

Die aktuelle Performanz der Komponente ist unbekannt.
Durch Verwendung von:

1. Streams und
2. Threads + Threadpools

eigenen Threads+Threadpools für das versenden von Nachrichten dürfte die Performanz aber ausreichend sein.

3 FSS

Das File System Service (FSS) ist für Dateizugriffe zuständig. In der derzeitigen Fassung ist es leider nicht optimal und muss überarbeitet werden.

Patterns

Verwendete Patterns:
Observer/Event Listener (IProjectModificationListener)

Verbesserungswürdige [V]/ auszutauschende [A] / zu überprüfende Patterns [C]:
[C] Observer/Event Listener (IProjectModificationListener)

Interface

Status: [OK]/[NOK]
[NOK] fileExists/folderExists/getFullPath kann durch eine FileObject getFile(String relpath) ersetzt werden. Die Methoden können dann auf ein richtiges File-Objekt angewendet werden.
[NOK] isValidRelpath → boolean fileIsInProject(FileObject file)
[NOK] public String joinPath → hat das wirklich was im Interface zu suchen, sind interne Geschichten/Hilfsfunktionen → raus aus Interface
[NOK] getRootPath() macht nur Sinn bei einer Instanz des FSS / Projekt
[NOK] listFolder() sollte wohl am ehesten noch eine List<FileObject> zurueck geben
[NOK] recursiveListFiles() → entweder ganz kicken oder umbenennen in listFolderRecursive() oder noch besser listFolder(..., recursive=true). Darf keine Exception werfen wenn eine einzelne File nicht gültig ist.
[NOK] byte[] readFile() → kicken, da es eine Methode zum Erhalt eines FileObjects geben muss und man von diesem direkt lesen kann (stream, byte[] oder was auch immer) → keine Betriebssystem-Funktionen nachbilden!
[NOK] setRootPath/unsetRootPath → wirklich sinnvoll? (Features: Multiproject)
[NOK] writeFile(): Es ist sehr unwahrscheinlich, dass das FSS wirklich selbst ein File schreiben muss, ausser wir fangen an Notes zu persistieren. Am ehesten kommt da noch ein ImportFileIntoProject() in Frage, welches ein bestehendes (temp)-File ins Projekt importiert.
[NOK] calculateHashOverFile(String relativePath) → Externe Lib mit Übergabe eines File-Objektes.
[NOK] calculateHash(byte[] bytes) → siehe calculateHashOverFile, vor allem kein byte[] verwenden.
[NOK] int getHashLength(); was hat das im Interface des FSS zu tun!?
[NOK] long getFileSize(String relativePath) → file.getSize()
[NOK] public long getLastModified(String relativePath) → file.lastModified

Datentypen [OK]/[NOK]:
[NOK] List<String> listFolder(): siehe oben.
[NOK] void LaunchFile(): gibt es keine Moeglichkeit vom Betriebssystem zu erfahren, ob das geklappt hat?
[NOK] public byte[] readFile... gemischte Verwendung von byte[] und Byte[].
[NOK] public void writeFile(): Rückgabewert void wirklich sinnvoll?
[NOK] String getTempDir(): Statt String → File zurückgeben...
[NOK] String getTempFile(): Statt String → File zurückgeben...
[NOK] deleteFile(String ..): Sollte ein FileObject übergeben bekommen.
[NOK] calculateHashOverFile(String ...): Sollte ein FileObject übergeben bekommen

Datenfluss [OK]/[NOK]:
[?] Keine Kontrolle des Datenflusses durchgeführt.

Caching

Eine Methode zum Cachen der Hashes von Files muss überlegt werden bzw. die vorhandene (FolderWatcher: private HashMap<File, String> hashes = new HashMap<File, String>();) auf Effizienz kontrolliert werden.

Performanz

Die aktuelle Performanz der Komponente ist unbekannt, es ist jedoch durch die vorläufige Kontrolle davon auszugehen, dass diese unzureichend ist.
Dies hat mehrere Gründe:

1. Verwendung falscher Datentypen (byte[] statt Übergabe von File-Objekten)
2. Erfindung eigenständiger Hash-Algorithmen anstatt vorhandene zu verwenden (FileHashCalculator)
3. Assozieren von Arbeitsspeicher für gesamte Größe von Files, z.B.
   FolderWatcher.java : CalculateHash
   ```
   FileInputStream fr = new FileInputStream(f);
   len = (int) f.length();
   buf = new byte[len];
   n = fr.read(buf, 0, len);
   ```
4. Multiple Instanzen eines FileHashers, z.B.:
   FolderWatcher.java Constructor
   ```
   hasher = new FileHashCalculator();
   ```
   → FileHasher singleton oder die Methoden ganz statisch machen, am besten den FileHasher ganz weglassen.

Unschönheiten

1. FolderWatcher implementiert keine Interfaces (z.B. Runnable, Observer, etc.)
2. Eigener Hashing Algorithmus
3. (Scheinbar) Keine Möglichkeit den FolderWatcher aus dem Gui/Config heraus zu deaktivieren
4. Unbekanntes Verhalten wenn Dateisystem kein LastModificationDate unterstützt.
5. Merkwürdiges herumhantieren mit Strings anstatt Objekten

4 GUI

Das “GUI” ist die vorerst vorhandene grafische Oberfläche von Jake. Das Artefakt sollte eine konkretere Bezeichnung, z.B. SwingGUI erhalten, da andere mögliche “grafische Benutzeroberflächen” ebenfalls möglich sind.

Patterns

Verwendete Patterns:
Event Listener

Verbesserungswürdige [V]/ auszutauschende [A] / zu überprüfende Patterns [C]:
keine vorerst

Interface

Status: [OK]/[NOK]
kein Interface vorhanden.

Datenfluss [OK]/[NOK]:
[?] Keine Kontrolle des Datenflusses durchgefürt.

Caching

Obwohl bereits der Core gewisse Dinge cached, muss darauf geachtet werden, auch im GUI einfach zu implementierende Caches (xTableModel) zu verwenden und dort nicht jeden Aufruf an den Core weiterzugeben (z.B. Wenn ein Fenster resized wird)

Performanz

Bei korrektem Einsatz von Caches ist nicht mit Performanz-Problemen seitens der GUI zu rechnen. Denoch muss auf lose Kopplung der Komponenten geachtet werden.

Unschönheiten

1. Klassen für unterschiedliche Use-Cases liegen alle im gleichen Package
2. Eigene Implementierung eines “Translators”. Ersetzen durch eine MessageSource (siehe z.B.: http://rizafar.blogspot.com/2007/07/internationalization-spring-upon-it.html – bezieht sich zwar in weiterer Folge auf Spring-Web, aber ist trotzdem bei uns leicht verwendbar – siehe ggf. Spring Desktop)
3. Generelle Probleme – Cacheanomalien, Files die nicht angezeigt werden, etc.

2. Änderungen für neue Version

Hier werden die Änderungen, welche für die neue Version von Jake vorgesehen sind, beschrieben.
Prioritätvergabe: 10 Punkte (höchste Priorität) – 1 Punkt (geringste Priorität)

1 Multi Project

Abstract

Jake soll auf eine Multi-Projekt-Struktur umgestellt werden, d.h. in einer Jake-

Instanz können mehrere Projekte gleichzeitig geöffnet sein und werden von Jake verwaltet. Dies soll verhindern, dass zum gleichzeitigen Verwalten mehrere Projekte mehrere Instanzen von Jake geöffnet werden müssen, was eventuell zu einem Speicherproblem werden würde.

Priorität
8

Aufwandsschätzung
15 h GUI-Design und Implementierung
25 h Erstellen der neuen Architektur
60h Anpassung des Sourcecodes, Test

Beschreibung der Änderung:
Ein weiteres Problem einer Single-Projekt-Struktur mit mehreren offenen Instanzen ist die Zustellung von Nachrichten über das darunterliegende Messaging-Protokoll. Eine Nachricht, die von einem Messaging-Client an einen Benutzer gesendet wird, wird nicht an alle Messaging-Clients des Benutzers (und somit nicht an alle offenen Jake-Instanzen) zugestellt. So erhalten die verschiedenen Jake-Instanzen möglicherweise nicht jene Nachrichten die sie betreffen.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil
- ohne sie das gleichzeitige Verwalten mehrere Projekte praktisch unmöglich ist
- all jene Benutzer nicht mehr angesprochen werden würden, die mit mehreren Arbeitsgruppen arbeiten
- ansonsten nicht sichergestellt werden kann, dass Nachrichten für einen Client auch wirklich bei der richtigen Jake-Instanz eintreffen.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Es muss praktisch das gesamte Projekt umstrukturiert werden:
- Die GUI muss an die neuen Gegebenheiten angepasst werden,
- in das ICS muss eine weitere Schicht eingezogen werden, um die gleichzeitige Verwendung eines Messaging-Protokolls für mehrere Projekte nutzbar zu machen.
- Der ObjectListener im ICS muss entsprechend der MutliProjekt-Philosophie umgestaltet werden.
- Die SendMessage() Methode muss entsprechend erweitert werden, um mit dem richtigen Account zu senden und bei jeder Übermittlung die entsprechende ProjectID (siehe weiter unten) mit zu senden.
- GetUserID ist nicht mehr eindeutig, sobald es mehrere Projekte (mit mehreren, potentiell verschiedenen User-IDs) gibt. Die Überprüfung derselbigen mittels isOfCorrectUserIdFormat() sollte in das entsprechende NetworkService wandern.
- Das Fss muss mehrere Ordner gleichzeitig überwachen. Eventuell könnte man auch pro Projekt ein eigenes FSS starten.
- Der Core ist ebenfalls betroffen, eventuell könnte man pro Projekt eine eigene Sync-Komponente starten; dies ist jedoch nicht zwingend notwendig.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der nur gering betroffen.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind eine stärkere Strukturierung der gesamten Anwendung sowie eine erhöhte Funktionalität.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so ist damit zu rechnen, dass Benutzer, die gerne mehrere Projekte verwalten würden, Jake nicht verwenden werden.

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil sie sehr umfangreich ist und daher im letzten Semester nicht eingeplant wurde.

Verstößt gegen folgende vorherige Assumptions
Die Änderung verstößt gegen die 'single_project'-Assumption

Diese müssen geändert werden zu:
Die 'single_project'-Assumption muss komplett durch eine 'multi_project'-Assumption ersetzt werden.

Betroffene Module (core, gui, fss, ics):
core, gui, fss, ics

2 One-Database

Abstract
Zur besseren Unterstützung der Multi-Projekt-Struktur soll nur mehr eine einzige Datenbank (für alle Projekte) verwendet werden, anstatt pro Projekt eine eigene Datenbank zu halten.

Priorität
4

Aufwandsschätzung:
10 h neues Datenmodell & Implementierung
20 h Änderungen im Code

Beschreibung der Änderung:
Innerhalb der Datenbank müssen Projektespezifische Einstellungen/Filesharing-Informationen immer mit einem Projekt-Identifier angesprochen werden. Als Projekt-Identifier wird NICHT die Projektbezeichnung, wie sie der User kennt, verwendet.
Prinzipiell muss die Datenbank in drei Bereiche aufgeteilt werden:

- Jake-weite Einstellungen
- Projekt-spezifische-Einstellungen (evtl. Push-Pull-Einstellungen pro Projekt unterschiedlich)
- Projektspezifisches Logging der Fileaktionen
  Dieser Punkt könnte eventuell ein Bottleneck darstellen, da diese Informationen in höchstwahrscheinlich innerhalb einer Tabelle repräsentiert werden. Als Lösung schlagen wir vor, Log-Einträge, die nicht mehr benötigt

werden, zu löschen, d.h. bei gewissen Aktionen auf einer Datei (oder in einem Hintergrundtask) werden Log-Einträge, die logisch zu überschriebenen Änderungen oder Dateien gehören, entfernt.

Dadurch wird Projektkonfiguration und Programmkonfiguration getrennt.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil

- der Zugriff auf eine einzige Datenbank aus Entwicklersicht komfortabler ist
- für Projektunspezifische Informationen sowieso ein Platz zur Speicherung notwendig ist
- das handling mehrere User-Datenbanken unhandlich ist
- der User "aus versehen" die Datenbank eines Projektes löschen könnte
- es in möglichen zukünftigen Versionen von Jake möglich sein könnte, die bestehende Datenbasis von einer Jake-Version auf die nächste "upzudaten", dies geschieht am leichtesten mit einer einzigen Datenbank anstatt vielen verschiedenen.
- so sichergestellt werden kann, dass empfangene Nachrichten auch gesichert werden, wenn ein anderes Jake-Projekt zurzeit geöffnet ist.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Das Modul, das hauptsächlich betroffen ist, ist der Core – hier müssen Data-Objects und Zugriffe auf diese verändert werden.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung betroffen.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind einfacherer Zugriff, weil nur mehr eine Datenbankressource vorhanden ist, sowie eine bessere Verflechtung mit dem Multi-Projekt-Ansatz.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so ist damit zu rechnen, dass der Multi-Projekt nicht verfolgt werden kann.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Die bisherigen Konzepte ändern sich

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil es dem bisherigen Konzept widersprach.

Verstößt gegen folgende vorherige Assumptions:
die Änderung verstößt gegen die 'single_project'-Assumption, laut der die Daten jedes Projekts in einer eigenen Datenbank gespeichert werden. Die Änderung ändert dies auf eine einzige Datenbank.

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
TODO

3 Project-ID / Net-Import

Abstract

Jedes Projekt erhält eine kurze, semi-eindeutige Kennung die leicht vom Benutzer einzugeben ist. Hierdurch wird es ermöglicht, einfach und fehlerunanfällig Projekte von anderen Mitgliedern zu importieren bzw. in diese einzusteigen.

Priorität

6

Aufwandsschätzung

Einbinden in der GUI 5h
Vergabe der ID und Einbinden der ID in die Datenbank 10h
Importieren eines Projekts 10 h

Beschreibung der Änderung

Neben dem Namen des Projektes (z.B. *Projekt zur Verteilung der Urlaubsbilder zwischen mir und meinen Freunden*) soll bei Projekterstellung eine *Semi-Eindeutige ProjektID* in der Form eines Zufallsstrings fixer Länge (z.B. 8 Zeichen, A-Z, 0-9) erstellt werden.
Darüber hinaus muss es möglich sein, ein Projekt eines Freundes herunterzuladen, wenn man

- die Benutzeridentifikation des Freundes
- die Projekt-ID
- und die entsprechenden Passwörter/Zugriffseinschränkungen

kennt. Dies werden wir in weiterer Folge als Net-Import (Teilnahme an einem bestehenden Projekt) nennen.

Mithilfe der ProjektID wird das Projekt in allen Nachrichten zwischen den Clients identifiziert. Auch erleichtert es die Zusammenarbeit bzw. die Integrierung neuer Mitglieder – diese müssen nur die UserID des Projekterstellers/eines Projektmitgliedes (TBD mit Security) und diese ProjektID kennen um zu einem beliebigen Zeitpunkt in das Projekt einzusteigen (sofern die entsprechende Erlaubnis vorhanden ist).
Die Eingabe der Projekt-ID ist ein möglicher Knackpunkt – der Zielgruppe des Projekts ist nicht zuzumuten, kryptische Identifier (z.B. kK3ssang) direkt einzugeben bzw. sich zu merken. Vielleicht wäre es besser, die aus den möglichen Projekten, denen man beitreten kann, auswählen zu lassen. Dies wäre selbstverständlich mit einem erhöhten Entwicklungsaufwand (v.a. bei der GUI) verbunden.

Warum ist die Änderung notwendig?

Die Änderung ist notwendig, weil

- eine eindeutige Zuordnung von Daten zu Projekten möglich sein muss.
- User einfach und fehlerunanfällig in Projekte einsteigen können müssen.
- Die Verwendung eines anderen Tastaturlayouts nicht den Zugang zu einem Projekt behindern darf (z.B. wenn der Projektname Umlaute enthält, man selbst aber keine Umlaute auf der Tastatur eingeben kann)

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig
Die GUI muss ebenso umstrukturiert werden, wie das ICS; Methoden zur Kommunikation müssen die neue Project-ID verwenden.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung betroffen, die neue Project-ID ist auch in der Datenbank zu verwenden.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind Vereinfachung der Kommunikation und Vereinfachung der Verwaltung der Projektzugehörigkeit.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so ist damit zu rechnen, dass die Zuordnung verschiedenster Daten zu Projekten nicht sauber realisiert ist (Zuordnung über Projektnamen?).

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Die bisherigen Konzepte ändern sich nicht. Diese Änderung setzt die projectId-Assumption um.

Warum wurde es vorher nicht so implementiert/konzipiert?
Fehlende Zeit und Notwendigkeit für den ersten Prototypen.

Verstößt gegen folgende vorherige Assumptions:
Abgelehnte projectId-Assumption

Diese müssen geändert werden zu:
projectId-Assumption muss angenommen werden.

Betroffene Module:
ics, core, gui

4 File-ID & Rename-Move File

Abstract
Jedes File bekommt beim Import in das Projekt eine eindeutige FileID.

Priorität
4

Aufwandsschätzung
Ansprechen aller Dateien über File-ID 25 h
Umbenennungs/Move-Feature 20 h

Beschreibung der Änderung
Durch die eindeutige FileID wird das neue Feature “Move/Rename File” ermöglicht, welches erlaubt, Dateien im Projekt umzubenennen und diese Änderung einfach auf anderen Clients zu replizieren.

Die FileID ist ein Hash über einen Zufallswert (z.B. MD5(Aktueller Timestamp + Zufallszahl + UserID)) und wird in der Datenbank dem File zugeordnet.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil Dateien nur so nach Umbenennung eindeutig identifiziert werden können.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Das FSS muss dahingehend verändert werden, dass es Dateien nicht nur über relativen Pfad, sondern vor allem durch die File-ID ansprechen kann. Ein entsprechendes File-ID – Relpath – Mapping muss implementiert werden.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung betroffen, die File-ID wird zu allen File-Aktionen gespeichert.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind

- die Möglichkeit, Dateien umzubenennen
- geringere Netzwerk-Last (relative Pfade sind meist länger als die File-ID).
- Veringerung der Netzwerklast durch bekanntmachung einer Änderung anstatt DELETE und CREATE wie vorher (welches das File auf alle Clients neu übertragen würde, auch wenn sich nur ein Buchstabe im Filename geändert hat)

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so muss eine Umbenennung weiterhin mit dieser unoptimalen Variante gelöst werden.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
TODO

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil sie gegen das Konzept verstossen haben.

Verstößt gegen folgende vorherige Assumptions:
Die Änderung steht in direktem Widerspruch zu der move-Assumption, die die bisherige Lösung, eine Datei zu löschen und neu anzulegen, beschreibt.

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
core, gui, fss, ics

5 Umstellung Struktur LogEntries

Abstract
LogEntrys sind ab sofort “unveränderbar”, d.h. es wird keine *Update-Funktion* mehr

im DAO der Logs geben, jede Änderung erzeugt einen neuen LogEntry.

Priorität
7

Aufwandsschätzung
Logentry-DAO 10h
Logentry-Abarbeitung 20 h
Logentry-Cleanup 10 h

Beschreibung der Änderung
Die Log-Entries müssen insbesondere die Projekt-ID und die File-ID des betreffenden Projekts und Files enthalten.
Um die Log-Entry-Tabelle nicht zu groß werden zu lassen, werden veraltete Logentries von einem Background-Task gelöscht. Veraltete Logentries sind jene Logentries, deren Änderungen bereits durch weitere, spätere Änderungen überschrieben wurden.
Jeder Logentry enthält einen Zeitstempel mit seiner Erzeugungszeit. Dadurch kann aus den Logentries eine Änderungshistorie rekonstruiert werden bzw. Noch durchzuführende Änderungen können erkannt werden.
Um die Synchronisation und Konflikterkennung zu erleichtern, wird eine extra Tabelle erstellt, welche *Dirty-Files* enthält (Files die lokal/remote geändert wurden und nicht mehr auf dem aktuellsten Stand sind). Die genaue Struktur dieser Tabelle wird noch erarbeitet. Das Flag isLastPulledVersion ist aus der Datenbank zu entfernen.
Die Logentries, die noch nicht abgearbeitet sind (z.B. lokales File ist veraltet) müssen entweder von einem Hintergrundtask oder spätestens bei der Referenzierung des dazugehörigen Files abgearbeitet werden. Eine optimale, jedoch nicht durchführbare Strategie zur Abarbeitung der Logentries ist folgende:

Bearbeite jenen Logeintrag, dessen Datei als nächstes referenziert wird. Wird zwischen der Referenzierung und dem jetzigen Zeitpunkt dieser Logeintrag durch einen anderen obsolet, so behandle diesen Logeintrag überhaupt nicht.

Es muss nun eine Strategie gefunden werden, die der oben beschriebenen möglichst nahe kommt. Die Methode syncLogAndChanges() im SyncService sollte in SyncLog und SyncChanges aufgeteilt werden oder durch einen Background-Task ersetzt werden.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil die bisherige Logentry-Struktur ein tatsächliches ICS nur schwer unterstützt. Einen Logentry näher an einem Logentry aus der wirklichen Welt zu modellieren (Unveränderbarkeit – Erzeugungsdatum – Replizierbarkeit des Zustandes aus Logentries) schafft hier Klarheit.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Im Core müssen die entsprechenden DAOs und deren Verarbeitung geändert werden. Das Sync-Service soll als Hintergrundtask laufen oder aufgeteilt werden. Push() und Pull() sollen mit Streams arbeiten und sollen den Distributionsalgorithmus implementieren. Ist kein Projektmitglied online, so muss dies über Exceptions gemeldet werden – ein direktes Pushen mittels Push(userid)

entfällt, ebenso ist die Commitmessage zu entfernen.
Eventuell kann das Sync-Service/ICS so angepasst werden, dass die anderen, derzeit angemeldeten Projektmitglieder bekannt sind und Entfernungen zu den Projektmitgliedern berechnet werden. User, deren Entfernung gering ist, werden bei der File-Distributions bevorzugt.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung betroffen.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind klarere Repräsentation des Zustandes des Projekts sowie möglicherweise einfachere Algorithmen für Push und Pull.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so muss ein anderes Modell für Logentries gefunden werden, da die bisherige Implementierung den Projektzustand sehr umständlich beschreibt.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Die bisherigen Konzepte ändern sich nicht.

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil eine lauffähige Version des Projekts erstellt werden musste, und eine Neustrukturierung dem im Weg gestanden wäre.

Verstößt gegen folgende vorherige Assumptions:
TODO

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
TODO

6 Projekt Import/Export

Abstract
Es soll möglich sein, ein Projekt, inklusive Konfiguration und aller seiner Dateien in ein Archiv zu exportieren und dieses “Projekt-Backup” wiederherzustellen.

Priorität
1

Aufwandsschätzung

Export-Funktion für Datenbank 5h
Import-Funktion für Datenbank 5h
Export-Funktion für Dateisystem 5h
Import-Funktion für Dateisystem 5h
Änderungen in der GUI 5 h

Beschreibung der Änderung
Hierzu sollen alle aktuellen *DB-Datensätze* eines Projektes und alle Files des Projektes in ein Archiv (z.B. ZIP/tar.gz) gepackt. Dies soll das Backup eines Projektes zu einem bestimmten Zeitpunkt ermöglichen. Auf Wunsch kann der Client zu einem späteren Zeitpunkt diesen Stand mithilfe eines Imports wieder herstellen.
Mithilfe eines LogSyncs mit einem aktuellen Client kann dann das Log abgearbeitet werden und der aktuellste Zustand hergestellt werden.

Warum ist die Änderung notwendig?
Die Änderung ist praktisch, weil dadurch Backups von kompletten Projekten durchgeführt werden könnten.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Die GUI und eventuell das FSS müssten erweitert werden.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung nicht betroffen.

Welche Vorteile ergibt die Änderung?
Die Vorteil der Änderung ist ein zusätzliches praktisches Feature.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, gibts es jedoch keine Konsequenzen.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Die bisherigen Konzepte ändern sich nicht.

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil sie ein optionales Ziel implementiert.

Verstößt gegen folgende vorherige Assumptions:
--

Diese müssen geändert werden zu:
--

Betroffene Module (core, gui, fss, ics):
core, gui

7 Trennung FileObject/NoteObject, Vererbung von JakeObject, Neue Verwendung von Notes

Abstract
Files und Notes sollen anders verwaltet werden, Notes soll es nur mehr zu Files und Projektmitgliedern, aber nicht zum Projekt im Allgemeinen gehören. Eigenschaften, die bisher Notes und Files gemeinsam hatten (Tags) sollen nur mehr für Files möglich sein.
Dadurch sollen Notes nicht mehr eigenständige Objekte sein, sondern ihrem ursprünglichen Zweck zugeführt werden. Es gibt **Notizen zu Files** – dies sind die

Notizen wie sie bisher bekannt waren. Jede Notiz ist *genau einem* File zugeordnet. Ein File kann mehrere Notizen haben.
Notizen zu Projektmitgliedern werden wie bisher gehandhabt. Eventuell sollte überlegt werden, die *Notizen zu Files* und *Noitzen zu Projektmitgliedern* anders - also nicht zwei mal "Notizen" zu benennen, um Verwirrung beim User zu vermeiden.

Priorität
3

Aufwandsschätzung

Entfernung nicht mehr relevanter Features aus der GUI 5h
Entfernung unnötiger JakeObject-Methoden 5h
Änderung des ICS-Interfaces zur unterschiedlichen Behandlung von Notes 5h
Implementierung des neuen ICS-Ansatzes 5h

Beschreibung der Änderung
Im Zuge der Trennung von Files und Notes werden Notes keine JakeObjects mehr sein. Es bleibt noch technisch zu diskutieren, ob das JakeObject komplett durch das FileObject abgelöst wird oder nicht. Auf jeden Fall müssen die generellen Methoden die im Moment noch für JakeObjects existieren, untersucht werden und entsprechend angepasst werden. Eventuell lässt sich hier ein Kompromiss finden, falls jemand die projektspezifischen Notizen nicht aufgeben will (FileID einfach NULL lassen). Tagging für Notes sollte jedoch unbedingt entfernt werden.

Da Notes ab sofort zu einem File bzw. Projektmitglied gehören, ist die Änderung einer Note eine Änderung, die das File bzw. die Benutzerstruktur betrifft und kann entweder ähnlich wie die Änderung eines Tags zum File behandelt werden oder wie eine Änderung in den Usern des Projekts behandelt werden. Eine Note ist nur mehr durch Angabe ihres Files eindeutig identifizierbar, eine Member-Note nur mehr durch Angabe des Projektmitglieds, das sie beschreibt.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, da die Verwaltung von Notes unnötig kompliziert ist.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Im ICS muss die Änderung einer Note den anderen Clients anders mitgeteilt werden. Die GUI muss verändert werden, da Notes nicht mehr getagt werden können und nun auch keine Notes zum Projekte mehr möglich sind.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung betroffen, die Notes nun mit der File-ID/User-ID des zu ihnen gehörenden Files/Users gespeichert werden müssen.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind eine einfachere Verwaltung von Notes und deren Änderungen.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, müssen sehr ähnliche, aber doch verschiedene Funktionen für die Synchronisation von Notes und von Files geschrieben werden. Dadurch, dass Notes alleine stehen oder zu Files gehören können und auch getaggt werden können,

ist die Synchronisation von Notes eigentlich komplexer als die Synchronisation von Dateien, die eigentlich die Hauptaufgabe von Jake ist.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
TODO

Warum wurde es vorher nicht so implementiert/konzipiert?
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil die bisherige Funktionsweise als praktisch erachtet wurde.

Verstößt gegen folgende vorherige Assumptions:
Die all_equal-Assumption wird durch die Änderung verletzt, jedoch nicht mehr, als das sie das durch die bloße Existenz von Notes nicht ohnehin schon ist. Durch die unterschiedliche Zugehörigkeit von Notes war die all_equal-Assumption von Anfang an verletzt.

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
TODO

8 Aufteilung des IJakeGuiAccess-Interface

Abstract
Das JakeGUIAccess Interface wird in einzelne, logisch zusammenhängende Teile aufgeteilt. Ziel soll es sein, Interfaces kleinster Granularität zu erhalten, um die Wiederverwendbarkeit zu erhöhen und das gemeinsame Arbeiten am Code des Projektes zu vereinfachen (keine Merge-Konflikte mehr!).

Priorität
10

Aufwandsschätzung
Aufspaltung des Interfaces 10h

Beschreibung der Änderung
Nach vorläufigen Überlegungen, wird das Interface in die folgenden Teile aufgesplittet:

1. **Sync-Komponente / Locking-Komponente**, etc.
   Alles was die Synchronisation, das Locking & das Konfliktmanagement von Projekten/Files aus GUI-Sicht angeht, kommt in diese Komponente. Das Interface wird überarbeitet und gemäß vorhandener & bewährter Patterns umgestaltet (Callbacks → Event Listener, etc.)
2. **Authorisierungs-Komponente**
   Alles was mit Authorisierung im Projekt zu tun hat, kommt in diese Komponente. Angefangen von den Use-Cases zum Login am Netzwerk bis zu eventuellen neuen Features (**z.B.** Passwort-Eingabe für User-Änderungen an Projekten → User kann ein Projekt sichern, Jake übernimmt nur Änderungen die von anderen Clients kommen automatisch. Für User-Änderungen muss ein Passwort eingegeben werden)

3. **Konfigurations-Komponente**
   Alle Änderungen an der Konfiguration von Jake oder eines Projektes sollte in diese Komponente kommen.
4. **Logging-Komponente**
   Es ist zu überlegen, inwiefern das Logging zentralisiert werden kann, sodass nicht jede Komponente irgendwie unstrukturiert LogEntrys erstellt.
5. **Messaging-Komponente → Entfernen**
   Ursprüngliches Ziel von Jake war/ist es, keinen Chat nachzubilden. Die Unterstüztung von Nachrichten zwischen Projektmitgliedern arbeitet gegen dieses Ziel. Deshalb sollte diese Komponente entfernt werden. Es sind hierbei aber nur *Nachrichten zwischen Usern* und nicht *Nachrichten zwischen Jake-Instanzen* gemeint. LogEntrys werden weiterhin mit Messages zwischen den Clients ausgetauscht.
6. **Projektverwaltungs-Komponente**
   Alle Projektverwaltungs-Features (Projekt erstellen, Projekt laden, Projekt aus dem Netz importieren, Projekt import/export – siehe Punkt oben) sollten in diese Komponente kommen.
7. **FileObject-Komponente**
   Alles zur Verwaltung von Files + Notes sollte in diese Komponente kommen. Dazu gehören alle Operationen auf Files (Import, Run, Delete, Merge, Move), sowie Möglichkeit Notes und Tags zu Files hinzuzufügen. Zu Run/LaunchFile sollten vom Securitybeauftragten überlegungen bezüglich der Sicherheit dieser Funktion gemacht werden.
8. **ProjectMember-Verwaltungs-Komponente**
   Hinzufügen / entfernen von Members, editieren von Notes der Members, etc. sollen in dieser Komponente abgehandelt werden. Eine geeignete Authorisierungsstruktur für das hinzufügen / löschen von ProjectMembers muss überlegt werden (Eventuell: Nur Projektersteller kann Mitglieder hinzufügen/entfernen, andere Mitglieder können das hinzufügen neuer Mitglieder “vorschlagen”, der Projektersteller kann diese dann genehmigen/ablehnen. Die Clients nehmen LogEntrys zum erstellen neuer Mitglieder/löschen von Mitgliedern nur vom Projektersteller an.)

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil das bisherige Interface nicht einem eindeutigen Bereich zugeordnet werden kann.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Ja

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung nicht betroffen.

Welche Vorteile ergibt die Änderung?
Die Vorteile der Änderung sind eine bessere Aufteilung der Funktionalität und damit eine klarere Struktur, sowie weniger Merge-Konflikte bei gleichzeitiger Bearbeitung. Außerdem können weitere, optionale User-Interfaces strukturierter implementiert werden, indem ein Interface-Teil nur jene Teile des IjakeGuiAccess bekommt, die es benötigt.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
TODO

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Die bisherigen Konzepte ändern sich nicht, jedoch werden eigenständige Angelegenheiten auch als solche behandelt, Stichwort “seperation of concerns”.

Warum wurde es vorher nicht so implementiert/konzipiert?
Unwissenheit, Unerfahrenheit.
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil das Interface im Laufe der Entwicklung immer größer und größer wurde.

Verstößt gegen folgende vorherige Assumptions:
TODO

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
core, gui, fss, ics

9 Umgestaltung ICS / Prozess-per-UserId

Abstract

Das ICS soll so umgestellt werden, dass es für jedes Protokoll, über das es Daten übertragen kann, ein Network-Service enthält, an dem sich der Benutzer angemeldet hat. Darüber hinaus sollen verschiedene Listener empfangene Nachrichten verarbeiten.

Priorität

9

Aufwandsschätzung

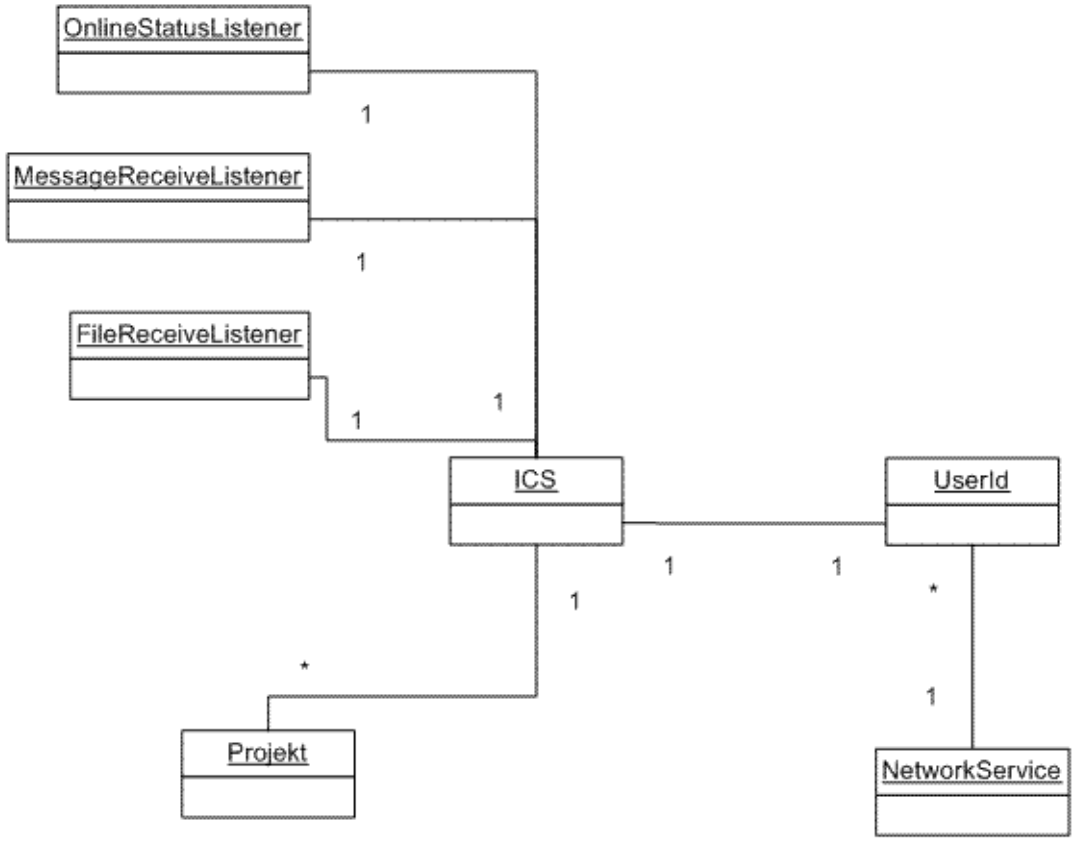
Anpassung der Methoden laut Plan 10h
Umstellung auf Multi-Threading 30h

Beschreibung der Änderung

Die folgende Grafik gibt eine Übersicht über die nötigen Änderungen im ICS.

Zwischen ICS und Projekt haben wir eine 1 - * Kardinalität gewählt, da bei einem Net-Import noch kein Projekt vorhanden ist. Weiters ist diese Lösung notwendig, damit man beim Erstellen eines Projekts online sein kann.

Der Funktion isLoggedIn() muss ein UserObjekt übergeben bekommen, da die Funktion generell verwendbar sein sollte und man bei mehreren Projekten mit unterschiedlichen Usernamen online sein kann.



Pro UserID (also z.B. domdorn@jabber.fsinf.at) soll es aus Performanz/Speichergründen eine Instanz des ICS geben (siehe Grafik oben). Die Methoden müssen entsprechend um einen ProjectID bzw. ein Project-Objekt erweitert werden.

Die Methode getServiceName() könnte in getProtocolName() umbenannt werden.

Das ICS sollte einen klugen Distributionsalgorithmus implementieren, um eine effiziente Verteilung der hinzugefügten/geänderten Files zu ermöglichen. Die Sync-Komponente hält eine Liste der Clients/Files und übergibt diese dann dem ICS. Insbesondere die Propagation, falls der Ersteller offline geht, seine Änderung aber von einem zweiten erlangt wurde, muss implementiert werden.

Warum ist die Änderung notwendig?
Die Änderung ist notwendig, weil das bisherige ICS nicht effizient mit mehreren Netzwerk-Protokollen und mehreren Projekten umgehen kann.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig?
Teilweise.

Ist eine Änderung des Datenmodells notwendig?
Das Datenmodell ist von der Änderung nicht betroffen.

Welche Vorteile ergibt die Änderung?

Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Bleibt die Änderung aus, so ist damit zu rechnen, das Nachrichten nicht an den Client weitergeleitet werden, der sie bekommen soll. Außerdem wäre es nicht möglich bzw. sehr fehleranfällig, mehrere Projekte zur selben Zeit zu verwalten.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Ja

Warum wurde es vorher nicht so implementiert/konzipiert?
Unwissenheit & "denken im kleinen Rahmen".

Verstößt gegen folgende vorherige Assumptions:
TODO

Diese müssen geändert werden zu:
TODO

Betroffene Module (core, gui, fss, ics):
ics, core, gui

Alter Content:

Alter Inhalt:

10Umstellung auf eine Multi-Projekt-Struktur

Jake soll auf eine Multi-Projekt-Struktur umgestellt werden. D.h. In einer Jake-Instanz können mehrere Projekte gleichzeitig geöffnet sein und werden von Jake verwaltet. Es gibt nur noch eine einzige Datenbank (nicht mehr eine Datenbank pro Projekt) in der alle Daten der vom User angelegten Projekte verwaltet werden.

11Eindeutige FileID

Jedes File bekommt beim Import in das Projekt eine eindeutige FileID. Die FileID ist ein Hash über einen Zufallswert (z.B. MD5(Aktueller Timestamp + Zufallszahl + UserID)) und wird in der Datenbank dem File zugeordnet.

12Feature "Move/Rename File"

Durch die eindeutige FileID wird das neue Feature "Move/Rename File" ermöglicht, welches erlaubt, Dateien im Projekt umzubenennen und diese Änderung einfach auf anderen Clients zu replizieren.

13Konfiguration Programm/Projekt trennen

Die Konfiguration von Projekteinstellungen und Programmeinstellungen sollen in

einzelne Tabellen aufgeteilt werden, um unabhängige Dinge voneinander zu trennen.

14Notes

Notes sollen in diesem Projekt nicht mehr eigenständige Objekte sein, sondern ihrem ursprünglichen Zweck zugeführt werden. Es gibt
Notizen zu Files – dies sind die Notizen wie sie bisher bekannt waren. Jede Notiz ist *genau einem* File zugeordnet. Ein File kann mehrere Notizen haben.
Notizen zu Projektmitgliedern werden wie bisher gehandhabt. Eventuell sollte überlegt werden, die *Notizen zu Files* und *Noitzen zu Projektmitgliedern* anders - also nicht zwei mal "Notizen" zu benennen, um Verwirrungen beim User zu vermeiden.

15LogEntrys

LogEntrys sind ab sofort "unveränderbar", d.h. es wird keine *Update-Funktion* mehr im DAO der Logs geben, jede Änderung erzeugt einen neuen LogEntry.
Um die Synchronisation und Konflikterkennung zu erleichtern, wird eine extra Tabelle erstellt, welche *Dirty-Files* enthält (Files die lokal/remote geändert wurden und nicht mehr auf dem aktuellsten Stand sind). Die genaue Struktur dieser Tabelle wird noch erarbeitet.

16Optional: Import/Export eines Projektes

Sofern Zeit bleibt und der Aufwand vertretbar ist, soll es möglich sein, ein Projekt zu exportieren. Hierzu sollen alle aktuellen *DB-Datensätze* eines Projektes und alle Files des Projektes in ein Archiv (z.B. ZIP/tar.gz) gepackt. Dies soll das Backup eines Projektes zu einem bestimmten Zeitpunkt ermöglichen. Auf Wunsch kann der Client zu einem späteren Zeitpunkt diesen Stand mithilfe eines Imports wieder herstellen. Mithilfe eines LogSyncs mit einem aktuellen Client kann dann das Log abgearbeitet werden und der aktuellste Zustand hergestellt werden.

17(Semi-)Eindeutige ProjektID

Neben dem Namen des Projektes (z.B. *Projekt zur Verteilung der Urlaubsbilder zwischen mir und meinen Freunden*) soll bei Projekterstellung eine *Semi-Eindeutige ProjektID* in der Form eines Zufallsstrings fixer Länge (z.B. 8 Zeichen, A-Z, 0-9) erstellt werden. Mithilfe dieser ProjektID wird das Projekt nun in allen Nachrichten zwischen den Clients identifiziert. Auch erleichtert es die Zusammenarbeit bzw. die Integrierung neuer Mitglieder – diese müssen nur die UserID des Projekterstellers/eines Projektmitgliedes (TBD mit Security) und diese ProjektID kennen um zu einem beliebigen Zeitpunkt in das Projekt einzusteigen (sofern die entsprechende Erlaubnis vorhanden ist).

18JakeObject stirbt → Lang lebe das FileObjects

Im Zuge der Trennung von Files und Notes werden Notes keine JakeObjects mehr sein. Es bleibt noch technisch zu diskutieren, ob das JakeObject komplett durch das

FileObject abgelöst wird oder nicht. Auf jeden Fall müssen die generellen Methoden die im Moment noch für JakeObjects existieren, untersucht werden und entsprechend angepasst werden (siehe z.B. nächster Punkt)

19 Keine Tags zu Notes

Tags werden nur noch an Files angefügt. Da Notes zu Files gehören, macht es keinen Sinn, für Notes auch noch Tags zu verwalten → Keine Tags zu Notes! Funktionen müssen entsprechend angepasst werden (z.B. AddTag(JakeObject o) → AddTag(FileObject o))

20 Aufteiltung des IjakeGUIAccess-Interface

Das JakeGUIAccess Interface wird in einzelne, logisch zusammenhängende Teile aufgeteilt. Ziel soll es sein, Interfaces kleinster Granularität zu erhalten, um die Wiederverwendbarkeit zu erhöhen und das gemeinsame Arbeiten am Code des Projektes zu vereinfachen (keine Merge-Konflikte mehr!). Nach vorläufigen Überlegungen, wird das Interface in die folgenden Teile aufgesplittet:

9. **Sync-Komponente / Locking-Komponente**, etc.
   Alles was die Synchronisation, das Locking & das Konfliktmanagement von Projekten/Files aus GUI-Sicht angeht, kommt in diese Komponente. Das Interface wird überarbeitet und gemäß vorhandener & bewährter Patterns umgestaltet (Callbacks → Event Listener, etc.)
10. **Authorisierungs-Komponente**
    Alles was mit Authorisierung im Projekt zu tun hat, kommt in diese Komponente. Angefangen von den Use-Cases zum Login am Netzwerk bis zu eventuellen neuen Features (**z.B.** Passwort-Eingabe für User-Änderungen an Projekten → User kann ein Projekt sichern, Jake übernimmt nur Änderungen die von anderen Clients kommen automatisch. Für User-Änderungen muss ein Passwort eingegeben werden)
11. **Konfigurations-Komponente**
    Alle Änderungen an der Konfiguration von Jake oder eines Projektes sollte in diese Komponente kommen.
12. **Logging-Komponente**
    Es ist zu überlegen, inwiefern das Logging zentralisiert werden kann, sodass nicht jede Komponente irgendwie unstrukturiert LogEntrys erstellt.
13. **Messaging-Komponente → Entfernen**
    Ursprüngliches Ziel von Jake war/ist es, keinen Chat nachzubilden. Die Unterstüztung von Nachrichten zwischen Projektmitgliedern arbeitet gegen dieses Ziel. Deshalb sollte diese Komponente entfernt werden. Es sind hierbei aber nur *Nachrichten zwischen Usern* und nicht *Nachrichten zwischen Jake-Instanzen* gemeint. LogEntrys werden weiterhin mit Messages zwischen den Clients ausgetauscht.
14. **Projektverwaltungs-Komponente**
    Alle Projektverwaltungs-Features (Projekt erstellen, Projekt laden, Projekt aus dem Netz importieren, Projekt import/export – siehe Punkt oben) sollten in diese Komponente kommen.
15. **FileObject-Komponente**
    Alles zur Verwaltung von Files + Notes sollte in diese Komponente kommen.

Dazu gehören alle Operationen auf Files (Import, Run, Delete, Merge, Move), sowie Möglichkeit Notes und Tags zu Files hinzuzufügen. Zu Run/LaunchFile sollten vom Securitybeauftragten überlegungen bezüglich der Sicherheit dieser Funktion gemacht werden.

16. **ProjectMember-Verwaltungs-Komponente**
    Hinzufügen / entfernen von Members, editieren von Notes der Members, etc. sollen in dieser Komponente abgehandelt werden. Eine geeignete Authorisierungsstruktur für das hinzufügen / löschen von ProjectMembers muss überlegt werden (Eventuell: Nur Projektersteller kann Mitglieder hinzufügen/entfernen, andere Mitglieder können das hinzufügen neuer Mitglieder "vorschlagen", der Projektersteller kann diese dann genehmigen/ablehnen. Die Clients nehmen LogEntrys zum erstellen neuer Mitglieder/löschen von Mitgliedern nur vom Projektersteller an.)

1 Caching

Die im Core implementierten Komponenten müssen sich um ein effizientes Caching kümmern. Es ist damit zu rechnen, dass die GUI ggf. unnötig viele Anfragen an Resourcen des Cores richtet. Die Komponenten müssen sicherstellen, dass diese Anfragen nicht 1:1 auf die Datenbank weitergeleitet werden, sondern die Datenbankresourcen erst nach einem refresh() aufruf bzw. durch setzen eines expliziten Flags geladen werden.

Ab hier wird es teilweise sehr detailliert (aber auch gut ausgearbeitet). Die Änderungen, die zu einem speziellen Änderungsvorschlag gehören (etwa MultiProject) sollten dorthin verschoben werden.

2 Überarbeitung ICS

isLoggedIn()
Muss ein UserObjekt übergeben bekommen, da die Funktion generell verwendbar sein sollte und man bei mehreren Projekten mit unterschiedlichen Usernamen online sein kann.

Pro UserID eine ICS Instanz
Pro UserID (also z.B. domdorn@jabber.fsinf.at) soll es aus Performanz/Speichergründen eine Instanz des ICS geben (siehe Grafik unten). Die Methoden müssen entsprechend um einen ProjectID bzw. Ein Project-Objekt erweitert werden.

ObjectListener → MultiProject
Der ObjectListener muss entsprechend der MutliProjekt-Philosophie umgestaltet werden.

SendMessage() → MultiProject
Die SendMessage() Methode muss entsprechend erweitert werden, um mit dem richtigen Account zu senden und bei jeder Übermittlung die entsprechende ProjectID (siehe oben) mit zu senden.

getFirstname/getLastname()
Es muss überprüft werden, ob diese funktionalität wirklich im generellen Interface was zu suchen hat, nachdem nicht sichergestellt werden kann, dass diese Felder in allen IM-Protokollen abrufbar sind. Vielmehr sollte der Projektersteller diese Felder ausfüllen und die Informationen mittels normalen LogEntrys/Messages verteilt werden.
Da ist afaik das Fallback, das im MockService implementiert ist, geplant verwendet zu werden.

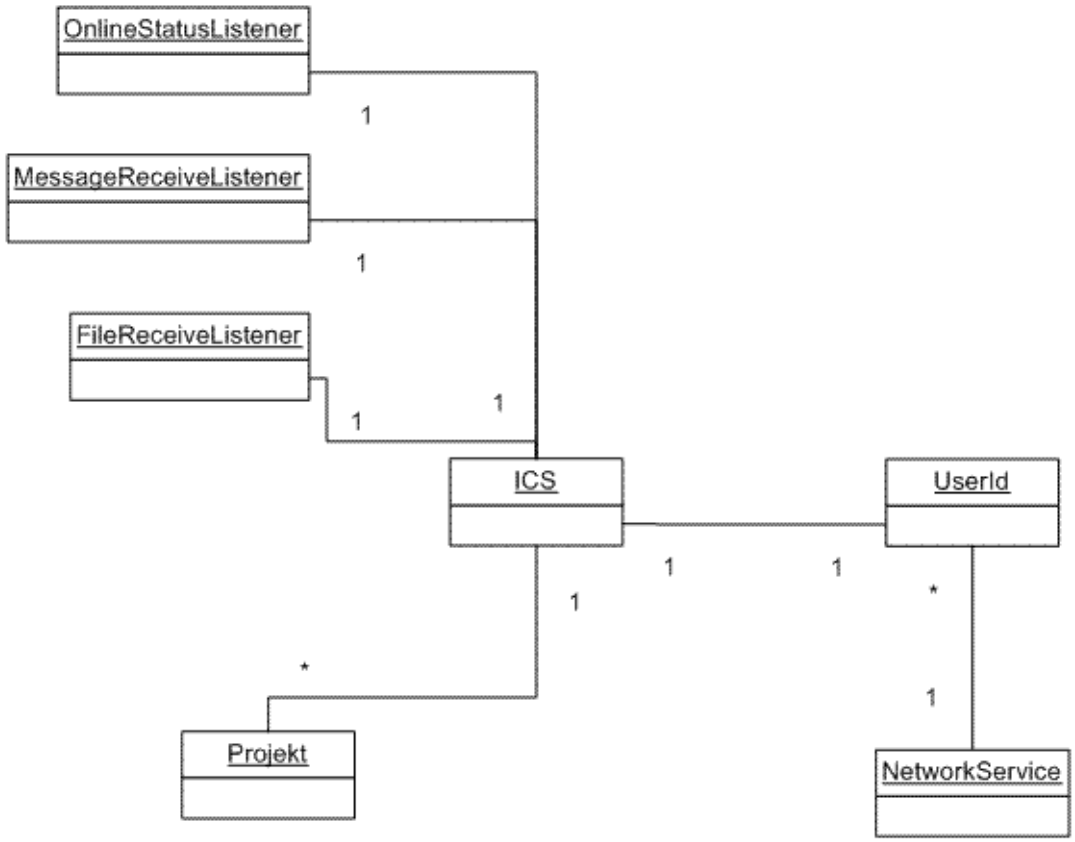
getUserID() → MultiProject
Auch diese Funktion muss entsprechend angepasst werden.

isOfCorrectUserIdFormat()
Es muss überprüft werden, ob wir diese Funktion brauchen. Falls ja, muss sie generisch sein und jede Implementierung eines *Networks* muss diese zur Verfügung stellen.

getServiceName() -> getProtocolName()
Methode sollte umbenannt werden.

Intelligenter Distributionsalgorithmus
Das ICS sollte einen klugen Distributionsalgorithmus implementieren, um eine effiziente Verteilung der hinzugefügten/geänderten Files zu ermöglichen. Die Sync-Komponente hält eine Liste der Clients/Files und übergibt diese dann dem ICS.
Insbesondere die Propagation, falls der Ersteller offline geht, seine Änderung aber von einem zweiten erlangt wurde, muss implementiert werden.

Umgestaltung des ICS → MultiProject



Zusätzliche Erläuterung:
ICS <-> Projekt Kardinalitaet *, da bei Net-Import noch kein Projekt vorhanden ist oder wenn ein neues Projekt erstellt wird, man schon online sein sollte.

3 Überarbeitung FSS

MultiProject
Es ist zu überlegen ob pro Projekt eine Instanz des FSS (und des dazugehörigen Scanner-Dämons) laufen sollte (eher unwahrscheinlich)

calculateHashOverFile()
Die Performanz dieser Funktion ist zu überprüfen und es ist sicherzustellen, dass das Rad mit dieser Funktion nicht neu erfunden wird. Der Einsatz anderer Hash-Algorithmen (z.B. CRC32) muss geprüft werden. Der Algorithmus muss Speicheroptimal sein, d.h. Auch bei großen Files eine max. Menge an Speicher nicht überschreiten (inkrementeller Algorithmus, Verwendung von Streams)

FileLauncher
Der Security-Experte muss wie vorher bereits angesprochen, Überlegungen bezüglich der Sicherheit dieser Funktion anstellen.

CalculateHash(File f)

Siehe *calculateHashOverFile()*. Es darf nicht die komplette File ins RAM geladen werden.

4 Überarbeitung SyncService

syncLogAndChanges()
Sollte aufgeteilt werden in SyncLog und SyncChanges (holen, importieren, whatever). Eventuell als Background-Task.

pull()
Sollte mit Streams und nicht mit byte[] arbeiten (RAM Verbrauch).

push()
Benötigt eine noOtherProjectMemberOnlineException, falls kein anderes ProjektMitglied Online ist und das File deshalb nicht gepusht werden kann. Der Parameter (userId) muss entfernt werden, da es Aufgabe des Distributionsalgorithmus ist, zu entscheiden wem eine File gesendet werden sollte. Die CommitMsg hat nichts in dieser Funktion zu suchen.

UserListe
Sync hält eine Liste mit Usern welche Online sind. Das ICS kann diese Liste mit LatenzChecks erweitern und so eine optimale Distribution bewerkstellen.

Distribution
Idee: Sync hat eine Userliste (z.B. nach Latenz sortiert),
sagt dem User "habe file fuer dich", anderer Antwortet "ok schicke" oder "nein mag nicht" ... falls 1 schicken, sonst naechster User. falls keiner will, Error anzeigen.
Derzeit so, dass nur über neue Änderungen Best-Effort benachrichtigt wird. Andere Seite holt sich dann selbst (was ich ganz gut finde).

HandleFetchRequest()
Hat nichts im Interface zu suchen, raus damit.

setICService(), setDatabase(), setFSService():
Überlegung, diese aus dem Interface zu entfernen. Wenn schon, dann Constructor-Injection. Diese Dinge sollten wohl nicht zur Laufzeit geändert werden können.

Pro Projekt eine Sync-Komponente → Multiproject
Es ist zu überlegen, ob es pro Projekt eine eigene Instanz der Sync-Komponente geben sollte oder ob diese generell gehalten werden und mit statischen Funktionen arbeiten sollte.

Eigener Thread
Entsprechende Nebenläufigkeit sollte überprüft werden um die Leistung zu optimieren.

Notizen von Johannes

Änderungsideen:

Folgende Lösungsvorschläge werden dazu vorgebracht (keine neuen Features hier):

Bsp:

zu 7 LogEntries:

Beschreibung der Änderung:

Dynamische Attribute aus der Tabelle in Hilfstabelle (mit denselben Schlüsseln) auslagern. Bei den Anfragen, die diese Attribute verwenden, dazujoinen.

Warum ist die Änderung notwendig?

Um 1 LogEntries zu beheben.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig (mehrere Komponenten, viele Klassen)?

Nein.

Ist eine Änderung des Datenmodells notwendig?

Ja, geringfügig.

Welche Vorteile ergibt die Änderung?

Keine.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?

Keine (Unschönheit).

Ist das eine Änderung des bisherigen Konzepts/Modellierung?

Nein.

Warum wurde es vorher nicht so implementiert/konzipiert?

-

Verstößt gegen folgende vorherige Assumptions:

-

Diese müssen geändert werden zu:

-

Betroffene Module (core, gui, fss, ics):

ics, core

zu 1337 Byte-Arrays:

Beschreibung der Änderung:

Streams sollten die Byte-Arrays ersetzen. Es ist zu überprüfen, inwieweit dies beim Hashen möglich ist.

Warum ist die Änderung notwendig?

Um Performance zu verbessern und Bottlenecks zu vermeiden.

Ist eine Umstrukturierung eines großen Teils des Projekts notwendig (mehrere Komponenten, viele Klassen)?

Ja – Datentypen ersetzen, einige Methoden in verschiedenen Klassen ändern - mittlerer Aufwand.

Ist eine Änderung des Datenmodells notwendig?

Nein.

Welche Vorteile ergibt die Änderung?

Skalierbarkeit für größere Dateien, Performance. Man kann die Assumption, dass keine sehr großen Dateien behandelt werden, eventuell entfernen.

Welche Konsequenzen hat die Nichtdurchführung der Änderung?

Keine gravierenden, falls man die o.g. Assumption lässt.

Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Nein.
Warum wurde es vorher nicht so implementiert/konzipiert?
Da die Übertragungsmechanismen vermutlich ohnehin keine sehr großen Dateien zulassen, ist der Hauptspeicher nicht als Problem angesehen worden.
Verstößt gegen folgende vorherige Assumptions:
-
Diese müssen geändert werden zu:
-
Betroffene Module (core, gui, fss, ics):
ics, core, fss

Vorschläge für zusätzliche Features

Wieder nach obigem Schema, nicht zu ausführlich beschreiben (nicht gesamten Planungsaufwand machen), vor allem nicht alle möglichen Features beschreiben, nur die für uns eventuell interessanten.

Beschreibung der Änderung:
Warum ist die Änderung notwendig?
Ist eine Umstrukturierung eines großen Teils des Projekts notwendig (mehrere Komponenten, viele Klassen)?
Ist eine Änderung des Datenmodells notwendig?
Welche Vorteile ergibt die Änderung?
Welche Konsequenzen hat die Nichtdurchführung der Änderung?
Ist das eine Änderung des bisherigen Konzepts/Modellierung?
Warum wurde es vorher nicht so implementiert/konzipiert?
Verstößt gegen folgende vorherige Assumptions:
Diese müssen geändert werden zu:
Betroffene Module (core, gui, fss, ics):

Abstract
Priorität: (10..1)
Aufwandsschätzung: xh
technische Details
Die Änderung ist notwendig, weil
Folgende Teile des Projekts müssen umstrukturiert werden:
Betroffene Module (core, gui, fss, ics)
Das Datenmodell ist von der Änderung (nicht) betroffen.
Die Vorteile der Änderung sind:
Bleibt die Änderung aus, so ist mit folgenden Konsequenzen zu rechnen:
Die bisherigen Konzepte ändern sich (nicht).
Verstößt gegen folgende vorherige Assumptions (+ Änderung)
Möglicherweise wurden die von dieser Änderung notwendigen Schritte noch nicht durchgeführt, weil