Review der technischen Archtiktur

Kurzbeschreibung:
Die technische Architektur von Jake wurde einem Review unterzogen. Ein erster Überblick über die notwendigen Änderungen für die Umsetzung der neuen Features von Jake (Multi-Projekt, Implementierung einer realen Netzwerkstelle) wurde 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

1 Umstellung 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.

2 Eindeutige 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.

3 Feature "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.

4 Konfiguration Programm/Projekt trennen

Die Konfiguration von Projekteinstellungen und Programmeinstellungen sollen in einzelne Tabellen aufgeteilt werden, um unabhängige Dinge voneinander zu trennen.

5 Notes

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.

6 LogEntrys

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.

7 Optional: 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.

8 (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).

9 JakeObject 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)

10 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))

11 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:

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.)

12 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 explizieten Flags geladen werden.

13 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.

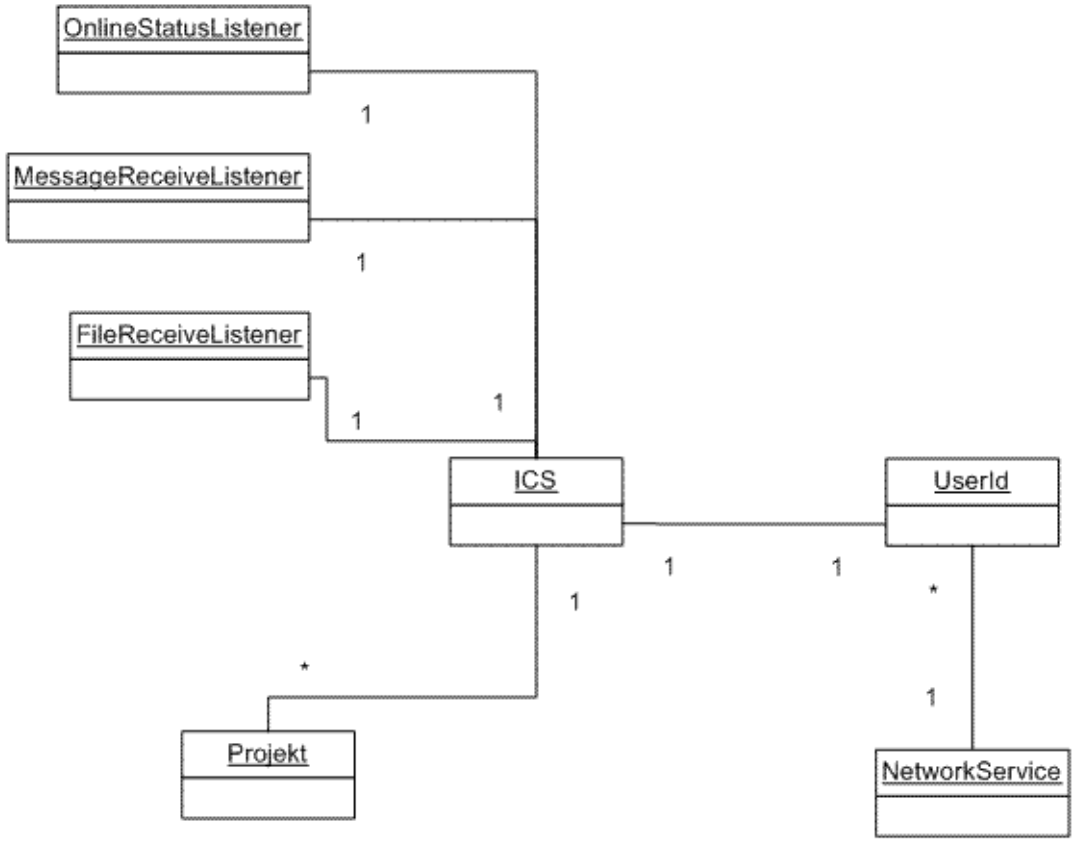
getUserID() → MultiProject
Auch diese Funktion muss entsprechend angepasst werden.

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.

Umgestaltung des ICS



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.

14 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.

15 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.

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
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.