Universität Potsdam
Institut für Informatik

Software Engineering II

SoSe 2013

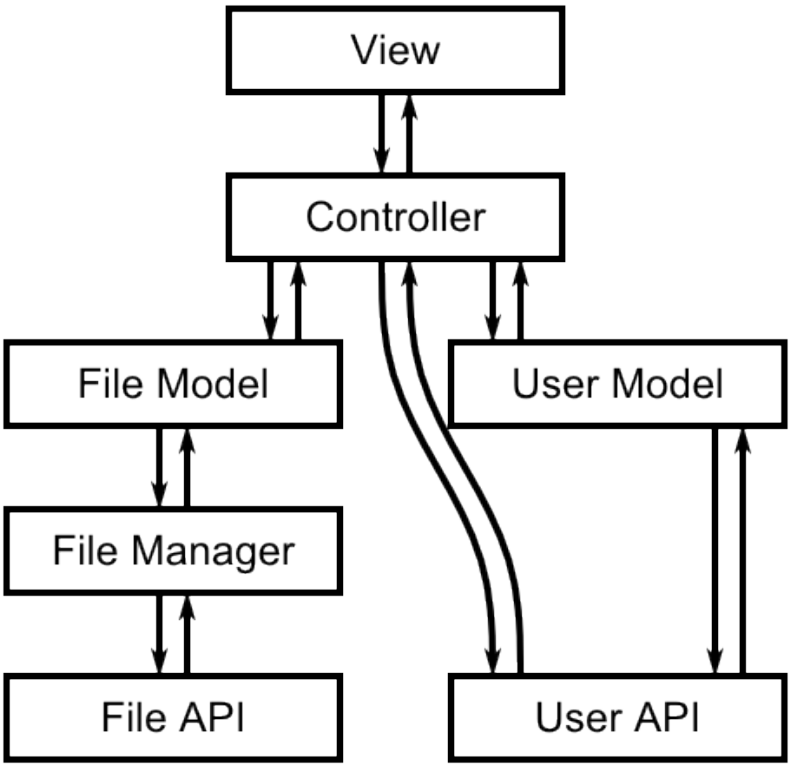
Projektarbeit „Sharebox Ultimate“

Abgabe 3: Entwurfsmodelle

Gruppe: SE2-04

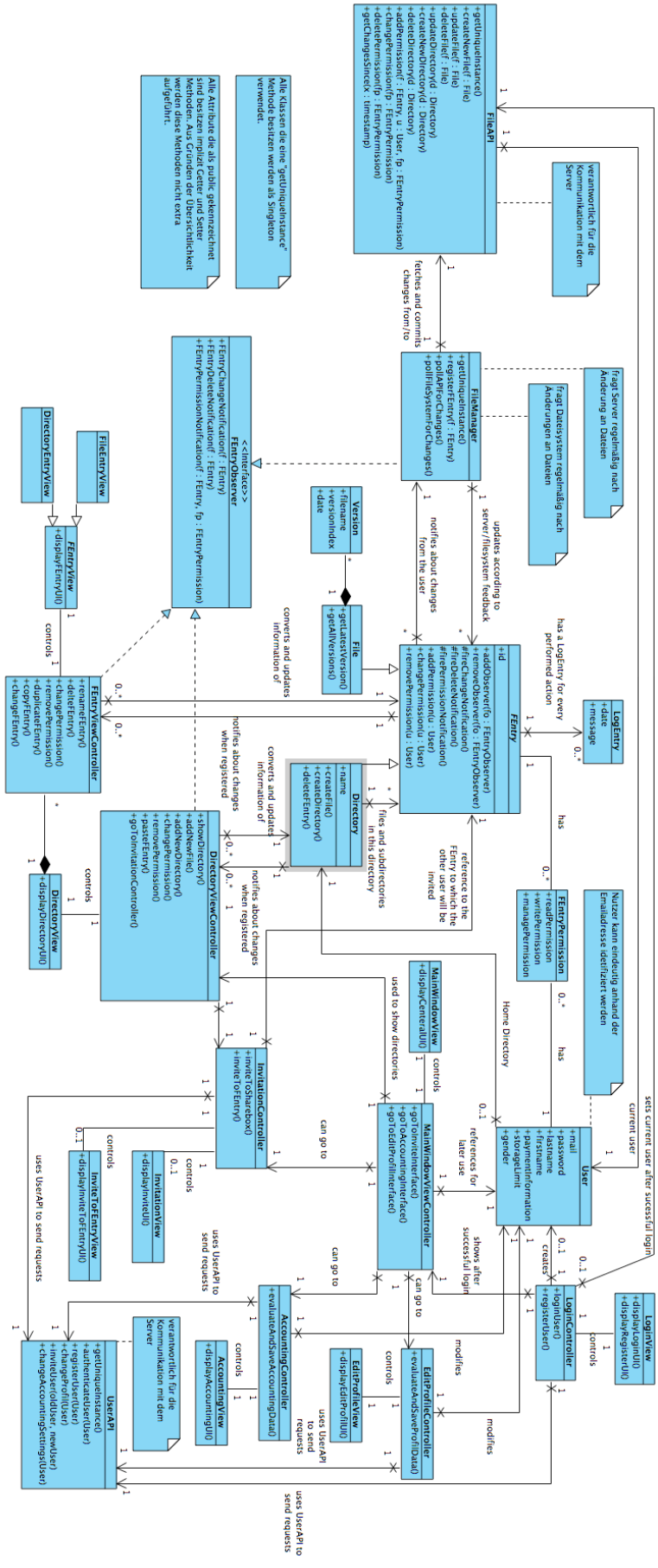
Benjamin Barth	762 747
Florian Gößler	761 698
Peter Jahn	731 486
Kay Thorsten Meißner	761 633
Julius Mertens	751 297

1.) Architektur



Das System wird als eine Kombination aus Schichtenmodel und MVC realisiert.
Auf der untersten Ebene befinden sich die API-Klassen (FileAPI und UserAPI), welche die Kommunikation mit der REST-Schnittstelle des Servers realisieren. Die FileAPI ist dabei für alle Aktionen zuständig, die direkt Dateien und Verzeichnisse betreffen. Die UserAPI übernimmt alles was sich nur auf den Nutzer an sich bezieht (zB. Logindaten überprüfen, Accountdaten ändern, Nutzer einladen, usw.).
Im Falle aller Dateien und Verzeichnisse bildet der File Manager die nächste Ebene. Dieser ist dafür zuständig das darüber liegende File Model auf Basis der Daten, die er regelmäßig vom Server abfragt, zu aktualisieren. Ebenso reagiert er durch die normalen Mechanismen des Betriebssystems auf Änderungen an den lokalen Dateien. Änderungen am File Model, die der Nutzer innerhalb der Applikation vornimmt, sendet der File Manager weiter an die FileAPI um den Server auf den neuesten Stand zu bringen.
Die File Model und User Model Schicht speichern die Daten innerhalb der Applikation. Sie stellen also das Model im MVC-Pattern dar.
Die Controller Schicht bereitet die Daten aus der Model Schicht für die View Schicht vor und reagiert sowohl auf Events, die Nutzer in den Views auslösen, also auch auch Events von der Model Schicht, wenn beispielsweise Dateienänderungen vom Server empfangen werden.
Auf der Seite der User API und des User Models gibt es keinen analoge Mechanismus zum File Manager, da hier nicht so viel Synchronisationsaufwand zu erwarten ist und die Controller einfache Änderungen direkt an die User API weiterleiten können.
Die View Schicht realisiert die konkrete Darstellung der Daten und des Interfaces. Sie ist allerdings dabei stark abhängig von der Controller Schicht, welche die Daten aufbereitet und auf Events reagiert. Eine direkte Kommunikation zwischen Model und View soll bei uns nicht stattfinden.

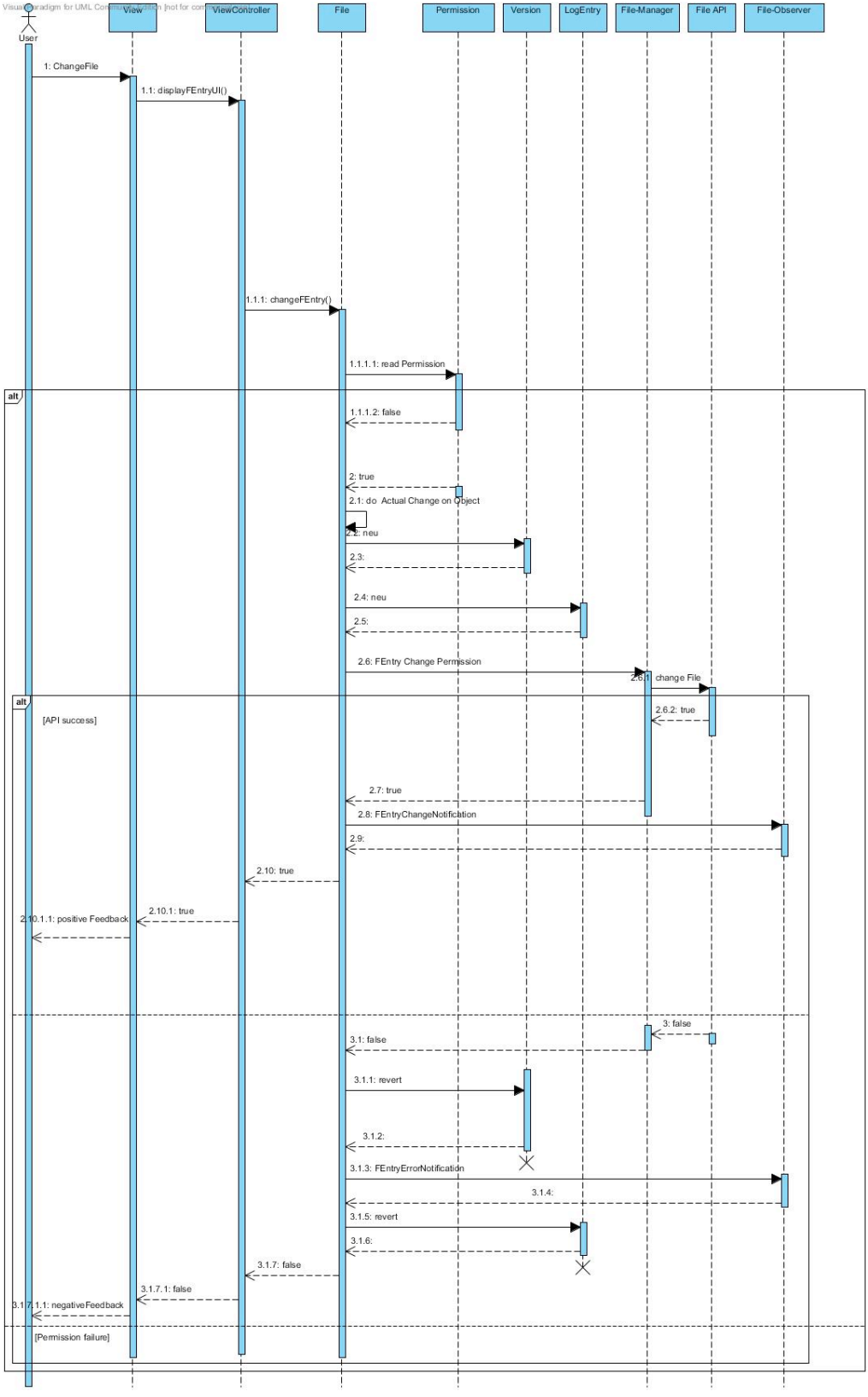
2.) Klassenstruktur



3.) Interaktions- und Kommunikationsverhalten

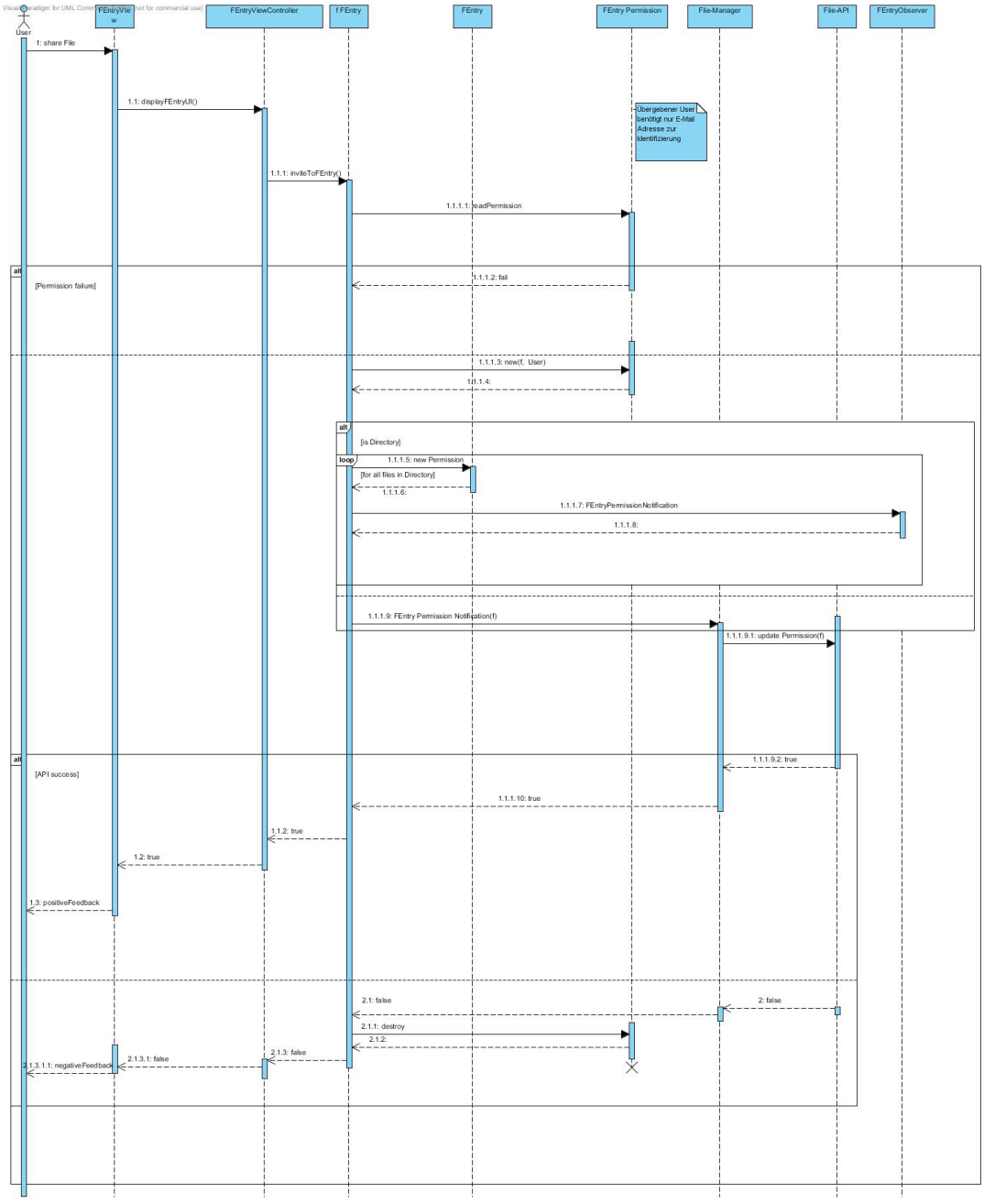
Hinweiß: Alle Abläufe in den Sequenzdiagrammen laufen synchron so wie aufgezeichnet ab.

3.1.) Datei ändern



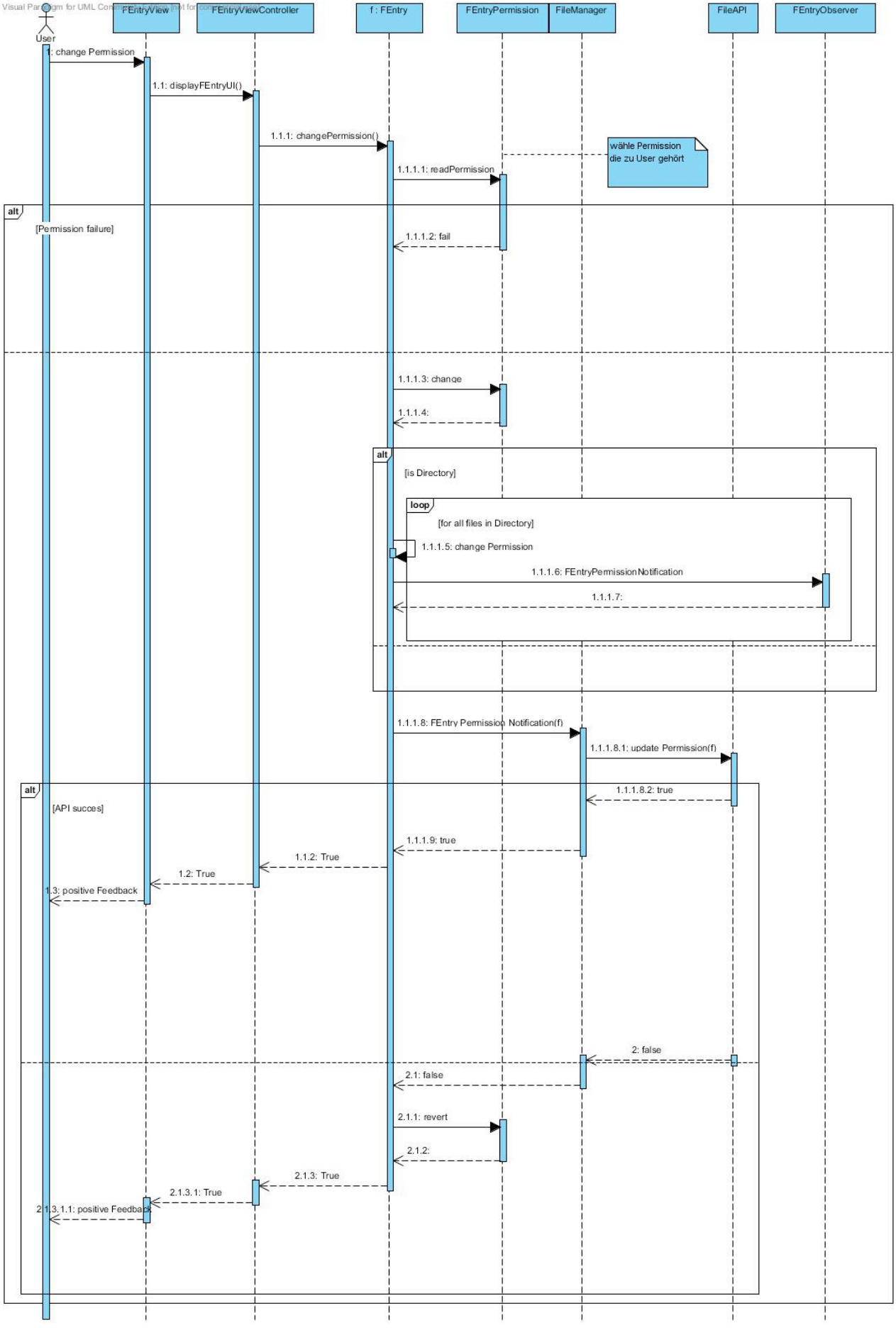
Dieses Diagramm beschreibt den Prozess der Veränderung einer Datei.
Zunächst stellt der Nutzer die Anfrage, dass er eine Datei ändern möchte, daraufhin wird überprüft, ob dieser auch die nötigen Rechte dafür besitzt. Ist dies nicht der Fall wird der gesamte Vorgang sofort abgebrochen, falls er die nötigen Rechte besitzt, wird zunächst eine neue Version und ein neues Log für die Datei erstellt.
Anschließend werden die Rechte neu gesetzt und abschließend die Datei geändert. Nun wird überprüft, ob dieser Vorgang einwandfrei funktioniert hat, wenn dies der Fall ist, wird eine Meldung an den Observer geschickt, der dann u.a. dafür sorgt, dass die Share-Box auf den neusten Stand gebracht wird. Schlägt das verändern der Datei wiederum fehl, wird der Vorgang abgebrochen und die neue Version und das neue Log werden wieder entfernt.
Am Ende wird noch ein Feedback an den User gegeben, ob der Vorgang erfolgreich abgeschlossen wurde.

3.2.) Datei/Ordner sharen



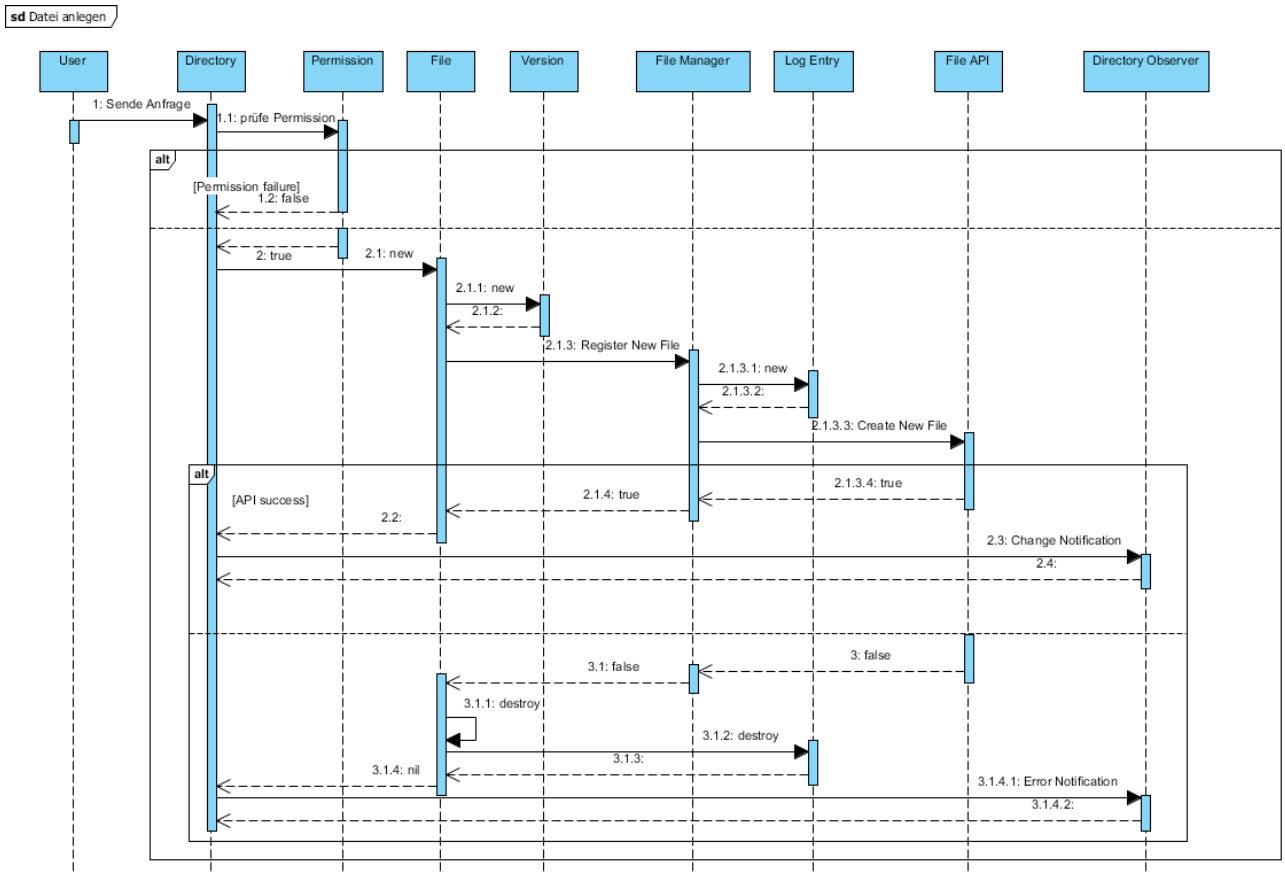
Dieses Diagramm beschreibt den Prozess des sharen eines Ordners oder einer Datei. Hier sendet der Nutzer zunächst eine Anfrage, dass er eine bestimmte Datei sharen will. Daraufhin werden seine Zugriffsrechte überprüft. Wenn seine Rechte nicht ausreichen wird der gesamte Vorgang abgebrochen. In dem Fall, dass der Nutzer die notwendigen Rechte besitzt wird der Prozess fortgesetzt. Zunächst wird eine E-Mail an den anderen User versendet. Wenn es sich um ein Ordner handelt, werden rekursiv die Rechte für alle Unterdateien vergeben und anschließend werden die Observer von den Änderungen benachrichtigt.
Wenn es sich um eine Datei handelt, werden die neuen Rechte für diese Datei auf den neusten Stand gebracht. Wenn dieser Vorgang funktioniert wird erhält der Nutzer ein positives Feedback. In dem Fall, dass der Vorgang fehlschlägt, werden die neuen Rechte wieder entfernt und der Nutzer erhält ein negatives Feedback bzw. eine Fehlermeldung.

3.3.) Rechte ändern



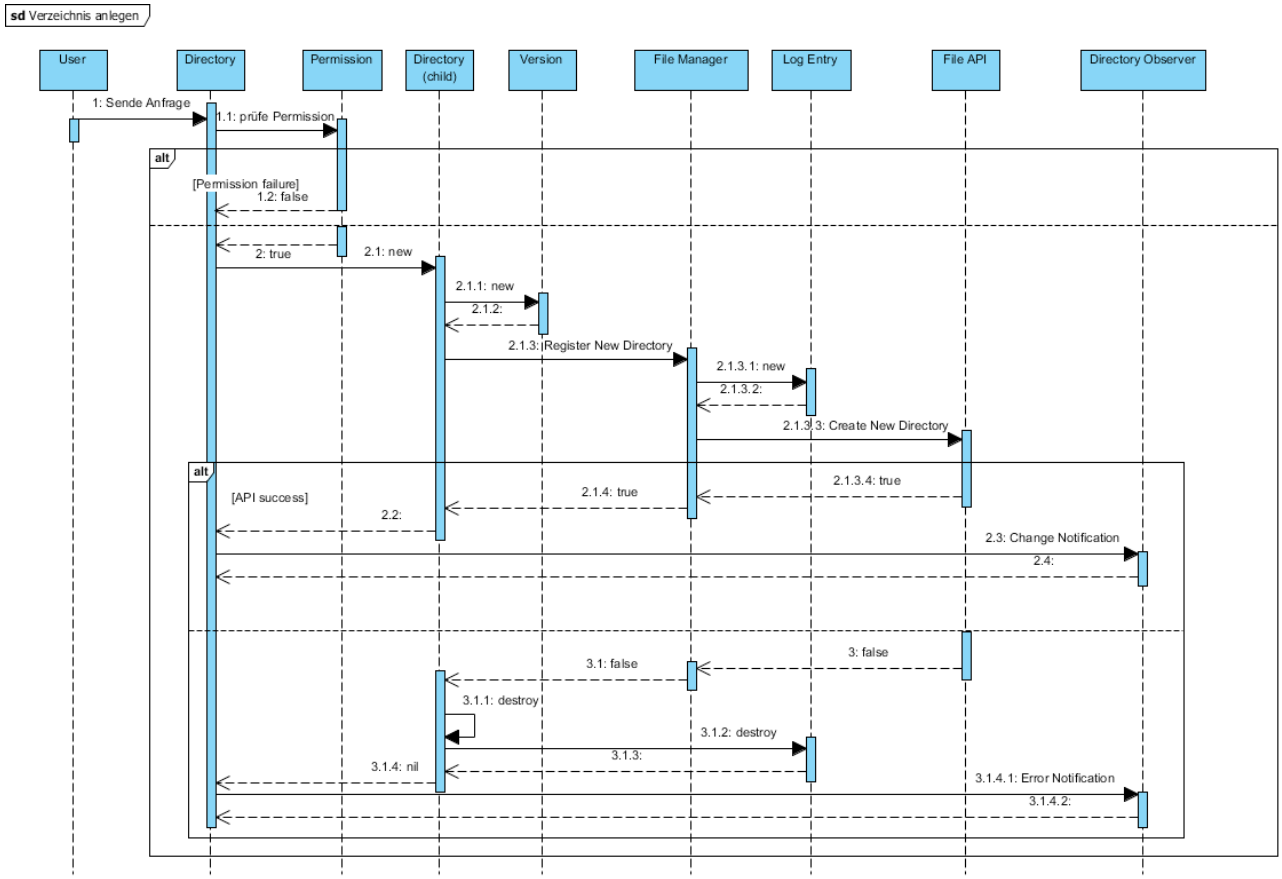
Dieses Diagramm beschreibt wie die Rechte einer Datei verändert werden. Hier sendet der Nutzer zunächst eine Anfrage, dass er bei einer Datei die Rechte ändern will. Daraufhin werden seine Zugriffsrechte überprüft. Wenn seine Rechte nicht ausreichen wird der gesamte Vorgang abgebrochen. In dem Fall, dass der Nutzer die notwendigen Rechte besitzt wird der Prozess fortgesetzt. Zunächst werden die Rechte geändert, im Falle eines Ordners rekursiv für alle Unterordner und Dateien. Daraufhin wird der FileManager von den Änderungen benachrichtigt und die Rechte werden auf dem Server mittels der FileAPI auf den neusten Stand gebracht. Wenn dieser Vorgang erfolgreich abgeschlossen wurde erhält der Nutzer ein positives Feedback. Falls der Vorgang fehlschlägt, werden die neuen Rechte wieder entfernt und der Nutzer erhält ein negatives Feedback.

3.4.) Datei anlegen



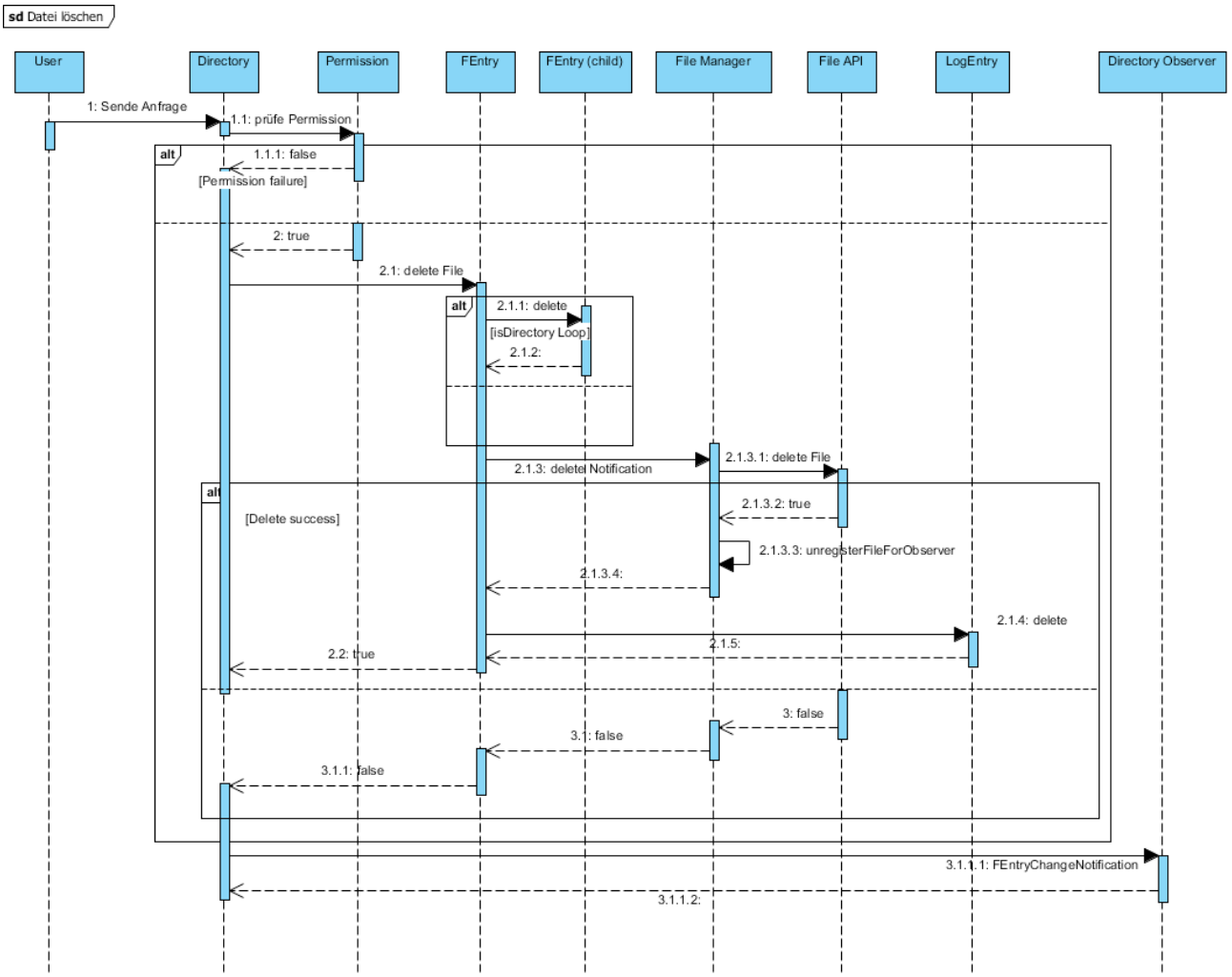
Bei dem Anlegen der Datei stellt der User zuerst die Anfrage, Wonach das dazugehörige gegebene Directory die Zugriffsbefugnisse des Users überprüft. Sollten diese nicht zureichend sein, wird der Anlegeprozess abgebrochen. Ansonsten wird ein neues File Objekt erstellt, Und die dazugehörigen Log-Einträge getätigt, inklusive eines neuen Eintrags in der Versionsliste. Danach wird, je nachdem, ob auf der API-Seite der Vorgang funktioniert, entweder ein Erfolgssignal gegeben oder das bereits erarbeitete File zerstört, wonach der Directory Observer informiert wird.

3.5.) Verzeichnis anlegen



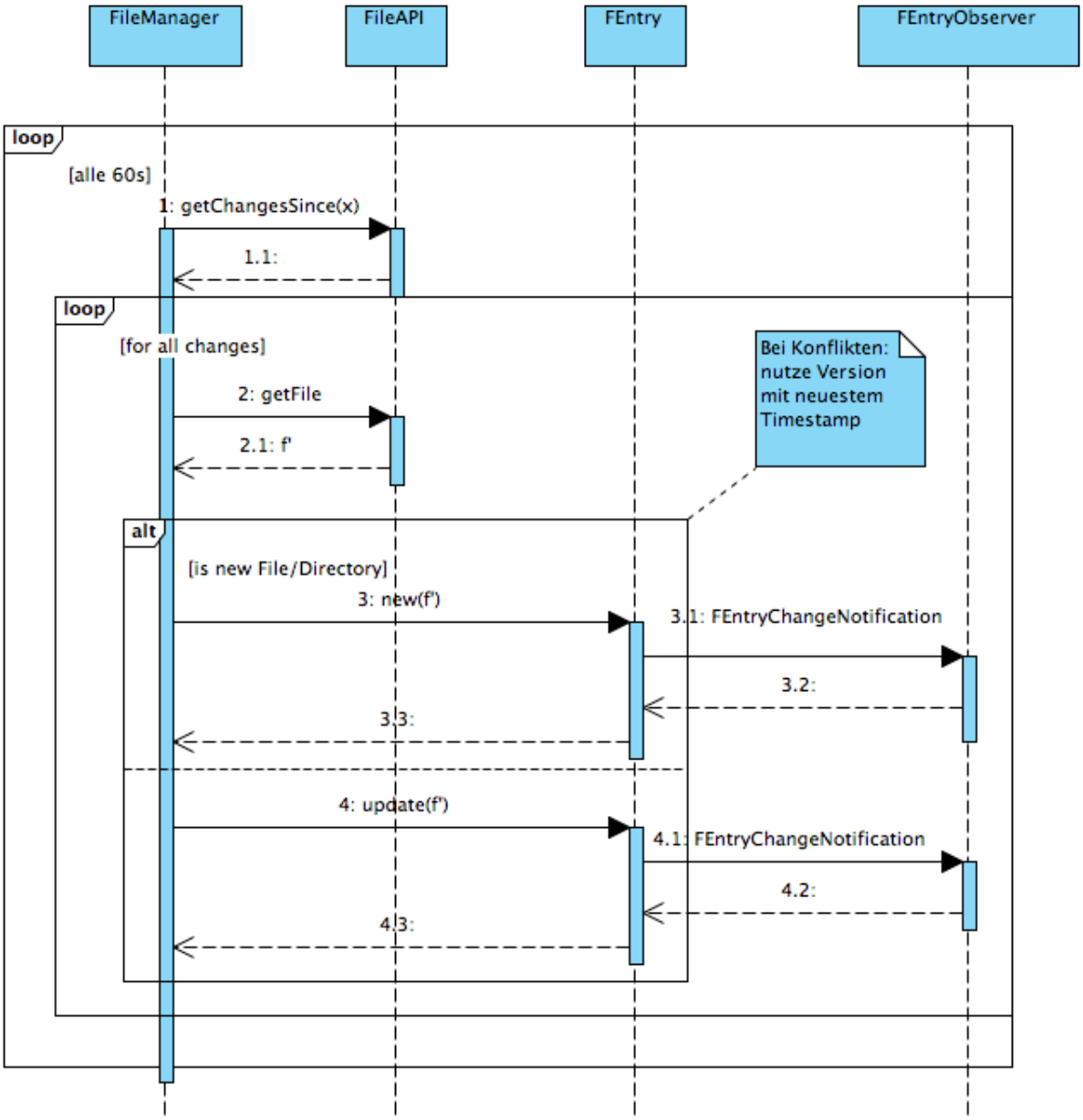
Ein Verzeichnis Anzulegen ist identisch mit dem Anlegen einer Datei, bis auf das in allen Fällen ein Directory anstelle eines Files versucht wird zu erstellen.

3.6.) Datei löschen



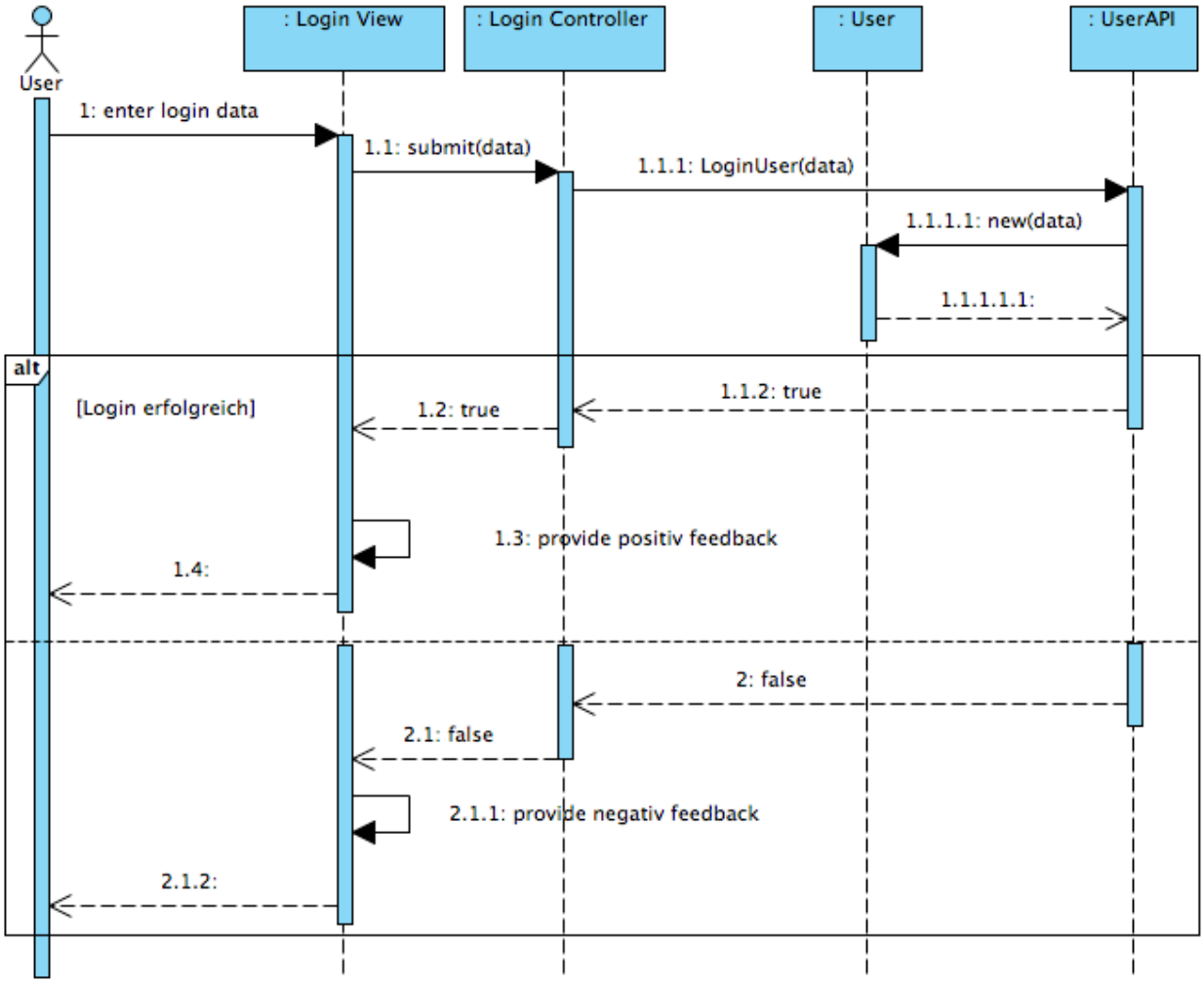
Das Löschen einer Datei Verläuft ähnlich dem Anlegen, jedoch wird nach erfolgreicher Überprüfung der Befugnisse der Delete-Befehl an alle Unterdateien und -Verzeichnisse gesendet, sollte der zu löschende Ursprungseintrag ein Verzeichnis selbst sein. Daraufhin wird das Objekt auch bei dem File Observer entfernt, oder der gesamte Prozess, wieder - bei fehlgeschlagener API Operation - abgebrochen.

3.7.) Änderungen vom Server/Dateisystem abfragen



Dieses Diagram beschreibt das Abfragen der Daten vom Server mithilfe der FileAPI. Diese liefert eine Liste an Änderungen an den Dateien des Nutzers seit dem übergebenen Zeitpunkt der letzten Abfrage. Auf Basis dieser Änderungsliste werden nacheinander alle Details zu den einzelnen Änderungen abgefragt und diese Änderungen an den lokalen Dateien vorgenommen.
Bei der Abfrage vom Dateisystem läuft dieser Vorgang analog ab, wobei jedes Mal alle Dateien bzw. Ordner auf eine Veränderung überprüft werden müssen - vorzugsweise anhand des vom System verwalteten Änderungsdatums der Dateien und Ordner.

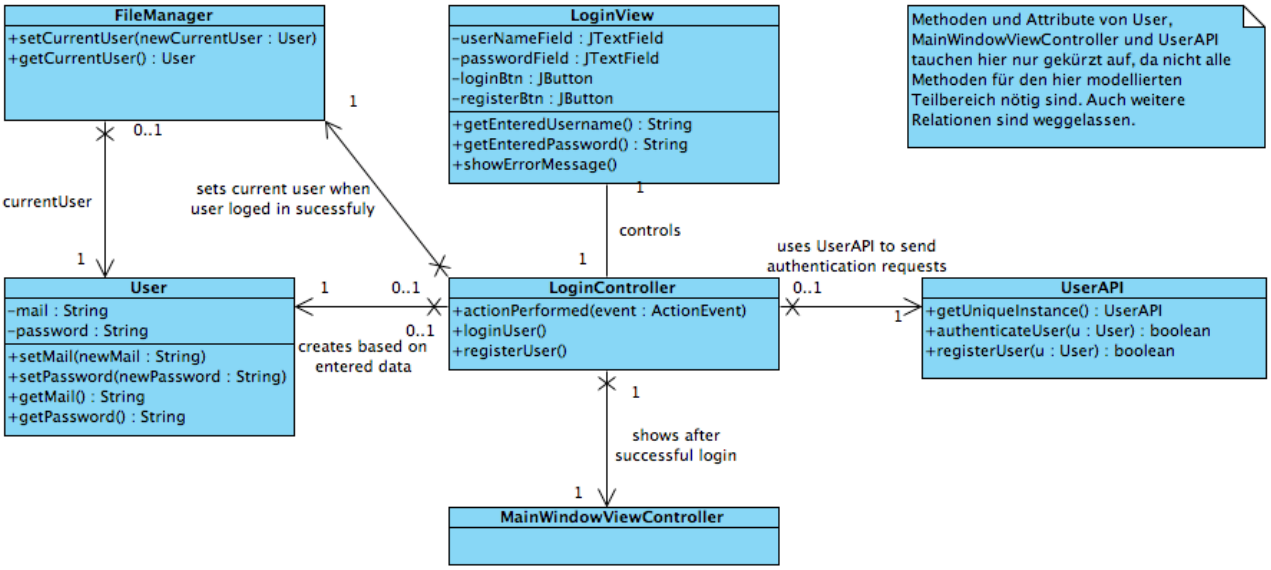
3.8.) Login



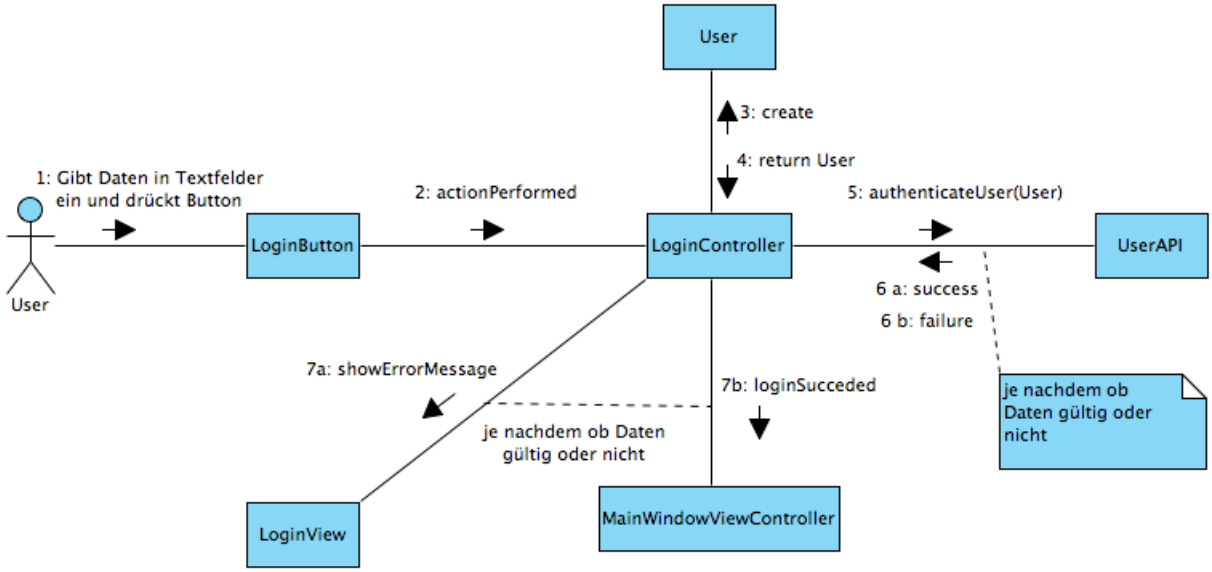
Dieses Diagram beschreibt den Loginvorgang. Alle Aktionen die eine Veränderung oder Abfrage der Nutzerdaten benötigen nutzen dafür die UserAPI. Dieses Diagram steht daher beispielhaft für jede solche Interaktion.
Der Nutzer gibt auf der Oberfläche etwas ein, der Controller erkennt diese Veränderungen bzw. wird davon benachrichtigt und leitet die Anfrage an die UserAPI weiter, welche die Kommunikation mit dem Server übernimmt und dann Rückmeldung gibt ob die Anfrage erfolgreich war oder nicht.

4.) Plattformspezifische Modelle

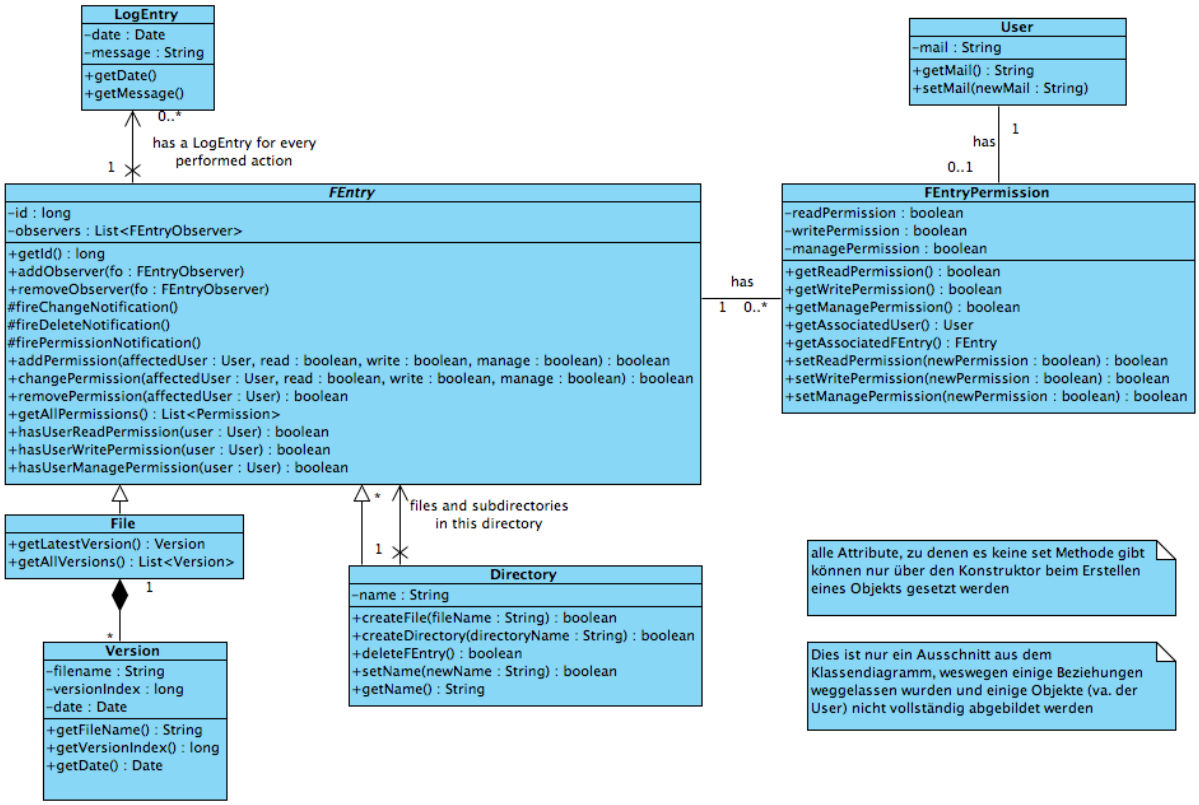
4.1.) Login Klassenstruktur



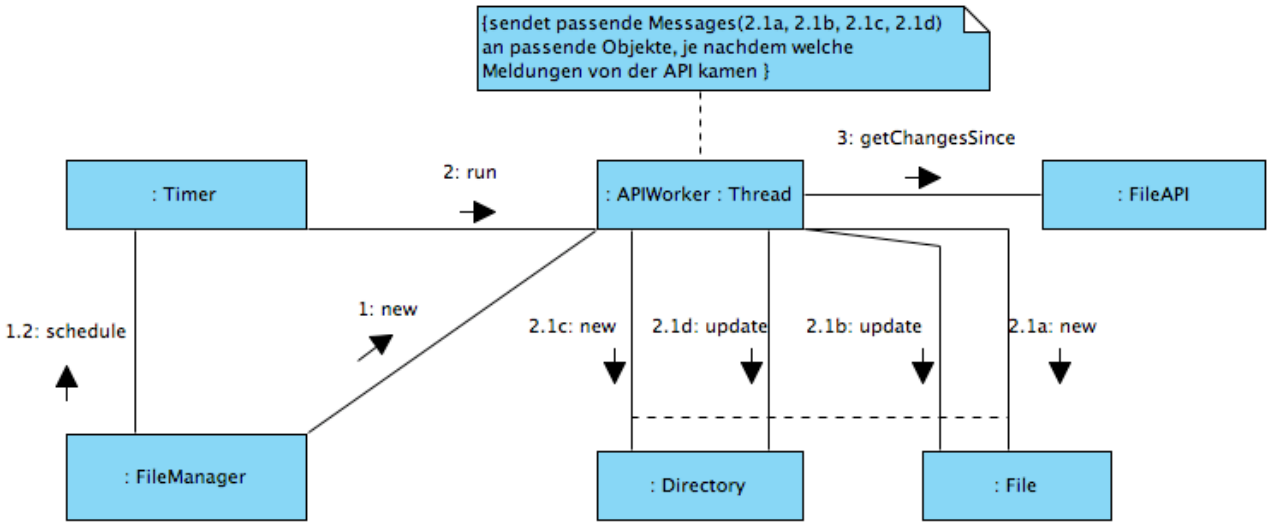
4.2.) Login Kommunikationsverhalten



4.3.) Aufbau der Klassen zur Dateiverwaltung (Model-Ebene)



4.4.) Änderungen vom Server abfragen



4.5.) Package Diagram