Architekturdokumentation
nach Arc42

SPAsS
(Semester-Planungs-Anwendung für studierende Studierende)

Version 1.1

Datum: 10.06.2022

Versionshistorie

VERSION	DATUM	VERANTWORTLICHE	WESENTLICHE ÄNDERUNGEN
1.0	03.06.2022	Finn Schindel, Tom Gouthier, Fabio Bertels, Marieke Schmitz	Aufsetzen Arc42-Dokument
1.1	10.06.2022	Finn Schindel, Tom Gouthier, Fabio Bertels, Marieke Schmitz	Bausteinsicht, Sequenzdiagramme
1.2	11.06.2022	Finn Schindel, Tom Gouthier, Fabio Bertels, Marieke Schmitz	Verfeinerung Bausteinsicht, Sequenzdiagramme

Abbildungsverzeichnis

Inhaltsverzeichnis

1. Einführung und Ziele

Die Anwendung „SPAsS“ (Semester-Planungs-Anwendung für studierende Studierende) ist eine interaktive Softwarelösung zur individuellen Studienplanung. Das Ziel der Applikation ist es, Studierenden das eigene Zusammenstellen eines Studienplans zu ermöglichen.
Die Aufgabenstellung, wesentliche funktionale und nicht-funktionale Anforderungen werden in der Anforderungsspezifikation in der Version 1.2 (26.05.2022) erläutert.

2. Bausteinsicht – Überblick über Gesamtsystem

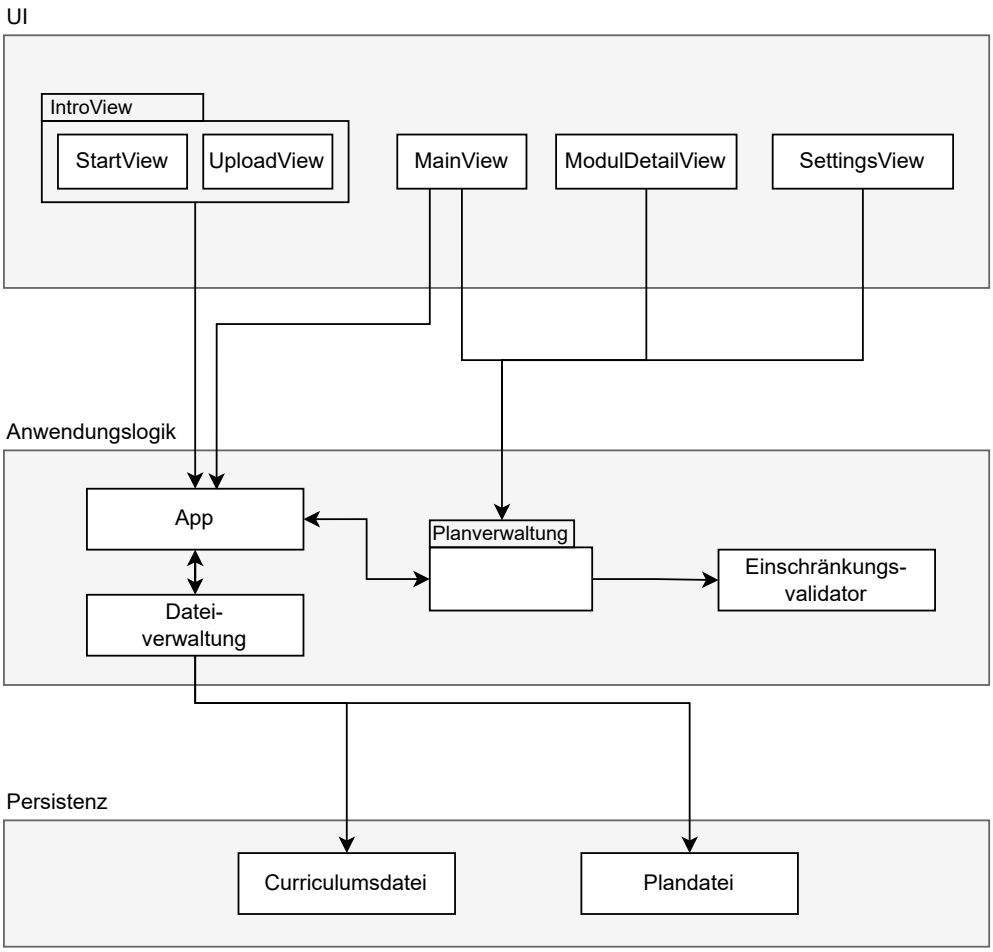

Abbildung 2-1 Paketdiagramm: Übersicht User Interface, Anwendungslogik, Persistenz

In Abbildung 2-1 werden die wesentlichen Komponenten des Gesamtsystems dargestellt. Die Einteilung erfolgt gemäß einer Schichtenarchitektur in die Komponenten des User-Interface (UI), in die Anwendungslogik und die Persistenzschicht. Die Komponenten greifen dabei jeweils nur auf die darunterliegende Schicht zu.

Die UI besteht zunächst aus der IntroViews, die in der Anforderungspezifikation Version 1.2 in Abbildungen 4-1 bis 4-5 in Wireframes dargestellt werden. Hier erfolgt die Wahl bzw. der Upload eines Plans und/oder eines Curriculums. Die IntroViews kommunizieren mit der App aus der Anwendungslogik. Zusätzlich gibt es die MainView (Hauptsicht mit Studienverlaufsplan), die ModulDetailView (vertiefende Informationen zu Modulen) und die SettingsView (Einstellungsfenster).

In der Anwendungslogik bietet die App die wesentliche Schnittstelle für die Views, die Planverwaltung und die Dateiverwaltung. Die Planverwaltung beinhaltet dabei alle Komponenten, die für die Kernlogik des Plans notwendig sind. Sie werden in Abbildung 2-2 weiter aufgeschlüsselt. Die Dateiverwaltung dient als Schnittstelle zur Persistenzschicht und ermöglicht das Laden bzw. Speichern von Dateien, also Curriculums- und Plandateien.

Attribute und Methoden von zentralen Komponenten werden in Abbildung 2-3 dargestellt.

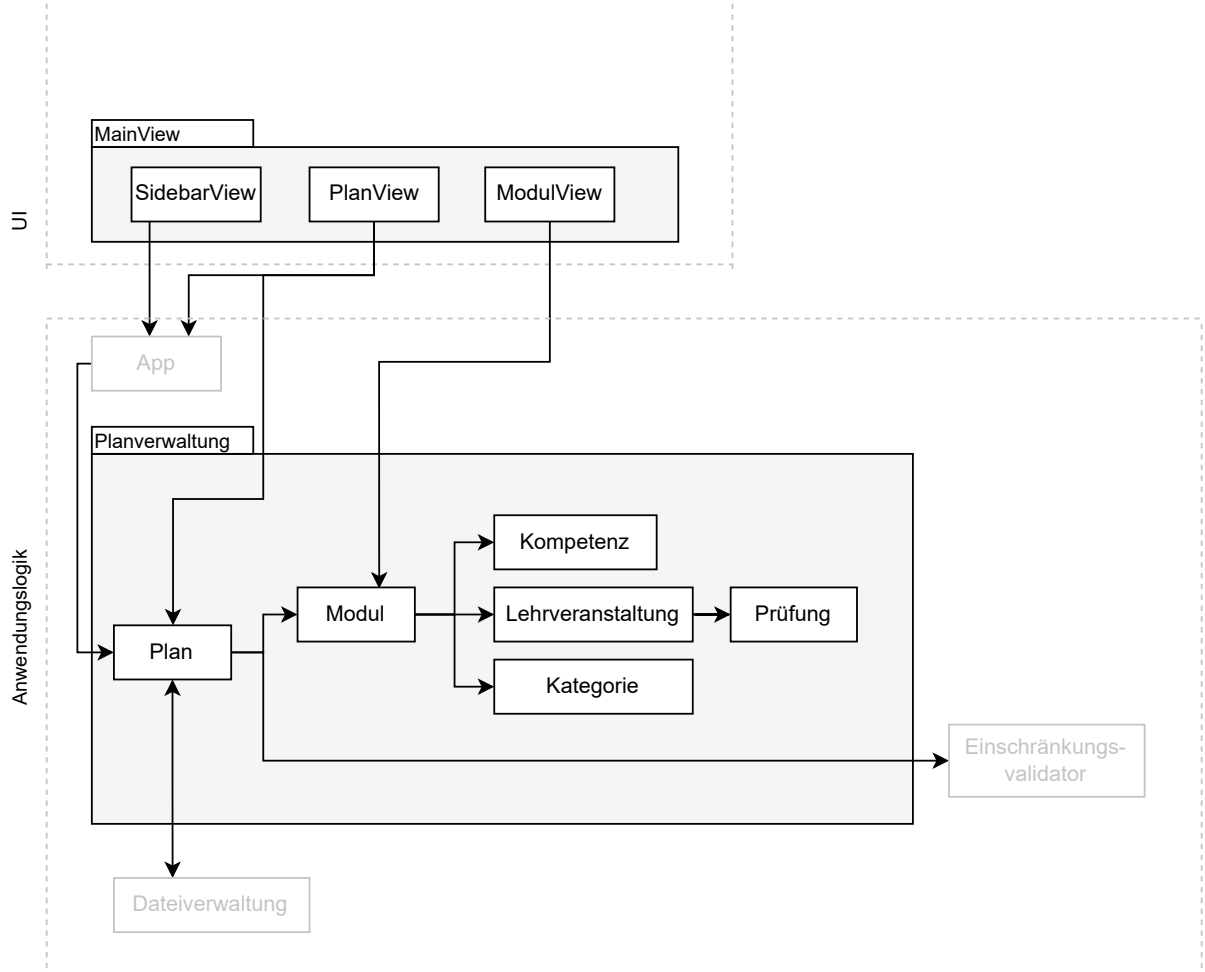

Abbildung 2-2 Paketdiagramm: Vertiefung PlanView/Planverwaltung

In einer Vertiefung in Abbildung 2-2 wird die MainView als Hauptansicht und die Planverwaltung als Kernanwendungslogik weiter aufgeschlüsselt.

Die MainView ist aufgesplittet in die SidebarView (Seitenleiste), die PlanView (Plandarstellung) und die ModulViews (einzelne Module innerhalb des Plans). Diese kommunizieren entweder über die App mit der Anwendungslogik oder aber direkt mit dem Plan/den Modulen innerhalb der Planverwaltung. Die Planverwaltung setzt die Kernfunktion der Anwendung um: den Umgang mit dem individuellen Studienverlaufsplan. Angelehnt an das Domänendiagramm (siehe Anforderungsspezifikation Version 1.2, Abbildung 3-1) beinhaltet der Plan eine Liste an Modulen, die wiederum Kompetenzen und Kategorien zugeordnet sind und aus Lehrveranstaltungen mit Prüfungen bestehen. Ein Plan hat zusätzlich eine Liste von Validatoren, die Einschränkungen (z.B. Fortschrittsregelungen, eigene CP-Begrenzungen) prüfen. Diese Validatoren implementieren das Interface „Validator“.

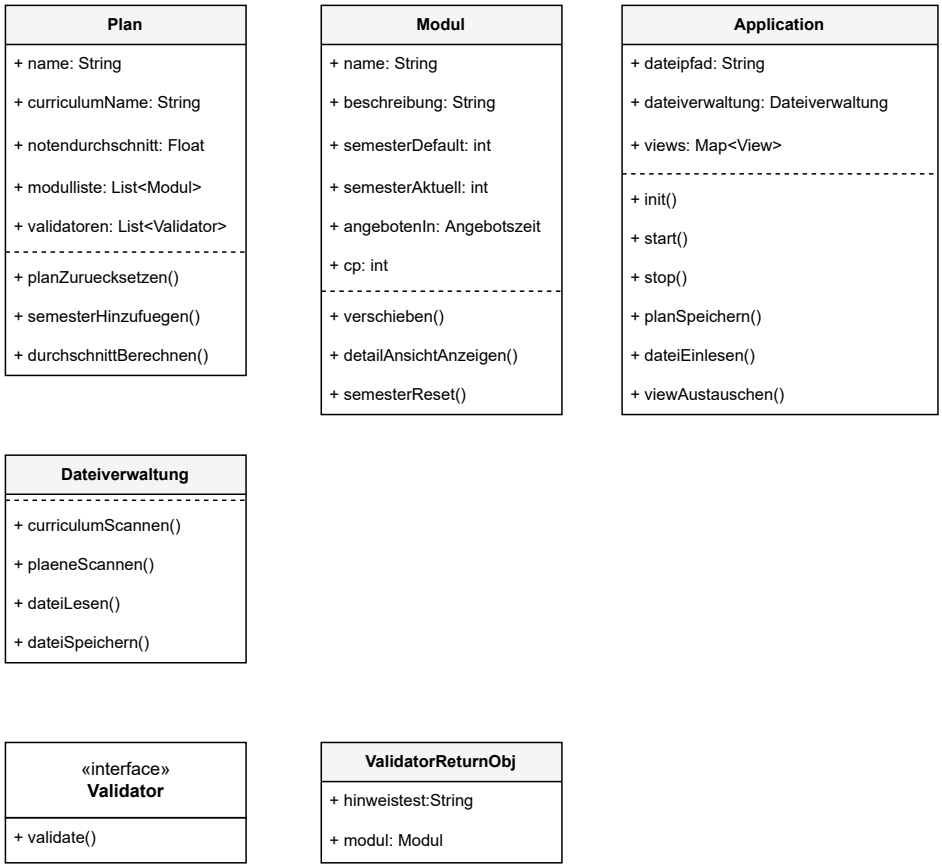

Abbildung 2-3 Übersicht Klassen Plan/Modul/Application/Dateiverwaltung/Validator

Aus den vorangegangenen Abbildungen zur Paketstruktur der Anwendung geht hervor, dass der Plan, die Module, die Application (App), die Dateiverwaltung und die Validatoren besonders relevant sind. In Abbildung 2-3 werden die Attribute und Methoden dieser Klassen daher dargestellt.

3. Laufzeitsicht

Auf Basis der Klassendiagramme in der Bausteinsicht erläutern die nachfolgenden Sequenzdiagramme, wie einzelne funktionale Anforderungen, wie sie in der Anforderungsspezifikation definiert wurden, umgesetzt werden.

Plan speichern

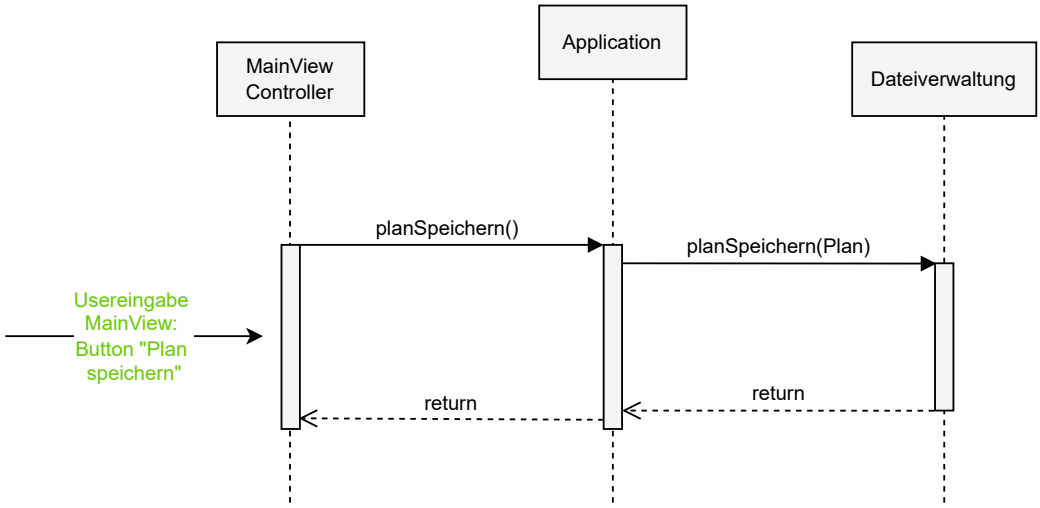

Abbildung 3-1 Sequenzdiagramm: Plan speichern

Nach der Individuellen Anpassung / Bearbeitung des Plans möchte der User ggfs. seine Datei in seinem Verzeichnis abspeichern. Um dies zu tun, betätigt der User in der MainView den „Speichern“-Button, welche die „planSpeichern“-Methode der Application aufruft. Diese Methode ruft die gleichnamige Methode in der Dateiverwaltung mit dem aktuellen Plan auf. Die Dateiverwaltung als Schnittstelle zur Persistenzschicht verarbeitet den Auftrag und speichert das Plan-Objekt in eine dafür angelegte Datei, die im Dateisystem abgelegt wird.

Datei einlesen

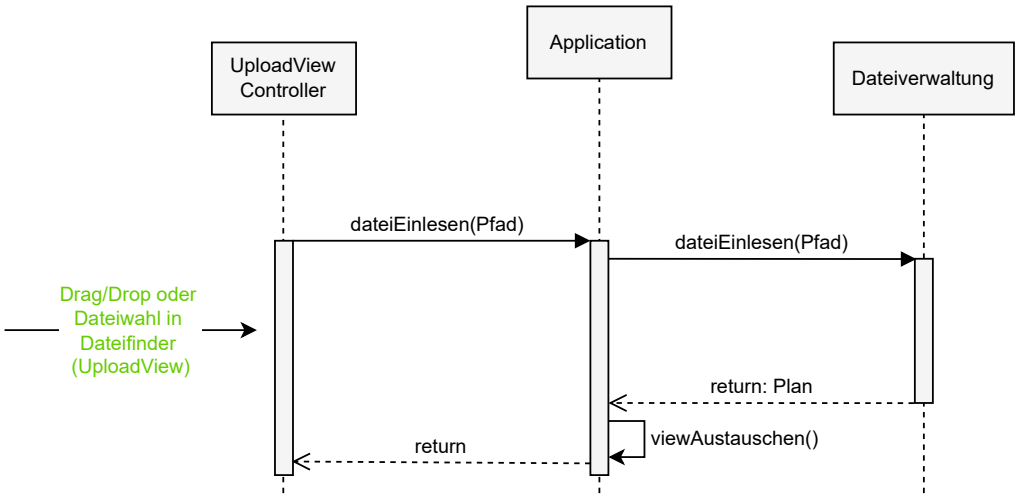

Abbildung 3-2 Sequenzdiagramm: Datei einlesen

Zu Beginn der Anwendung möchte der Benutzer ggfs. eine neue Datei (Curriculum mit Default-Plan und eigenen Plan) hochladen. In der UploadView legt er über Drag/Drop oder einen Dateifinder den Dateipfad fest. Diesen Pfad gibt der zugehörige Controller über die „dateiEinlesen“-Methode an die zentrale Application weiter. Diese wiederum ist die Schnittstelle zur Persistenzschicht über den Zugang zur Dateiverwaltung. Die Application gibt den Befehl nun also an die Dateiverwaltung weiter. Diese parst die Datei, erstellt daraus eine Planinstanz und gibt diese an die Application zurück. Die Application zeigt den neuen Plan an, indem er in die MainView wechselt.

Modul verschieben

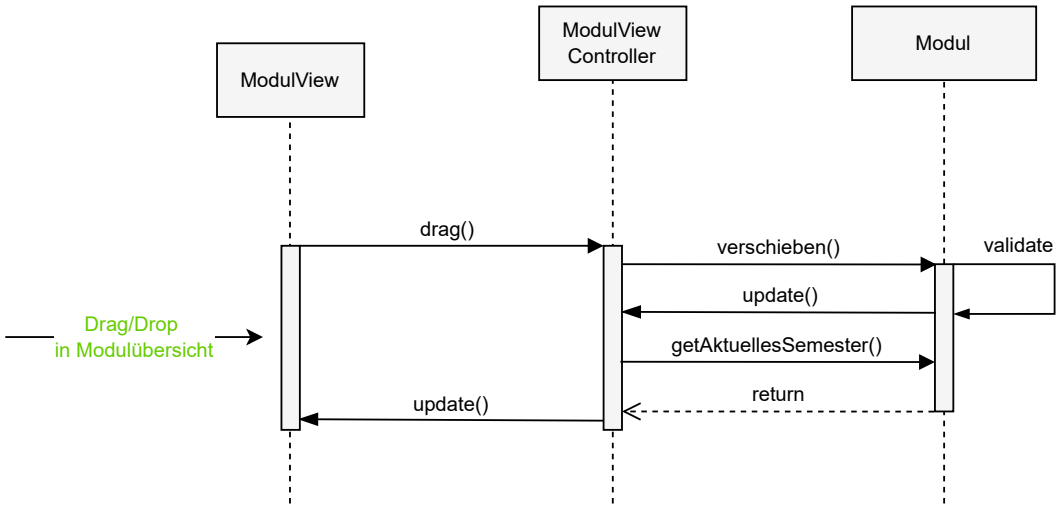

Abbildung 3-3 Sequenzdiagramm: Modul verschieben

Um Module in andere Semester verschieben zu können, soll der Benutzer die Module (ModulView) per Drag & Drop bewegen können. Hierzu wird der Drag über den ModulViewController ausgeführt. Der Controller benachrichtigt daraufhin das eigentliche Modulobjekt, dass es verschoben werden soll (verschieben()). Hier wird dann zunächst geprüft, ob die getätigte Verschiebung möglich bzw. valide ist. Gemäß des Observer Patterns meldet das Modulobjekt (Model) dem Observer (Controller) nach Tätigen der Änderung dann, dass eine Änderung stattgefunden hat. Dieser holt sich daraufhin die neue Position/das neue Semester für sein Modul und aktualisiert dementsprechend die View.

Plan zurücksetzen

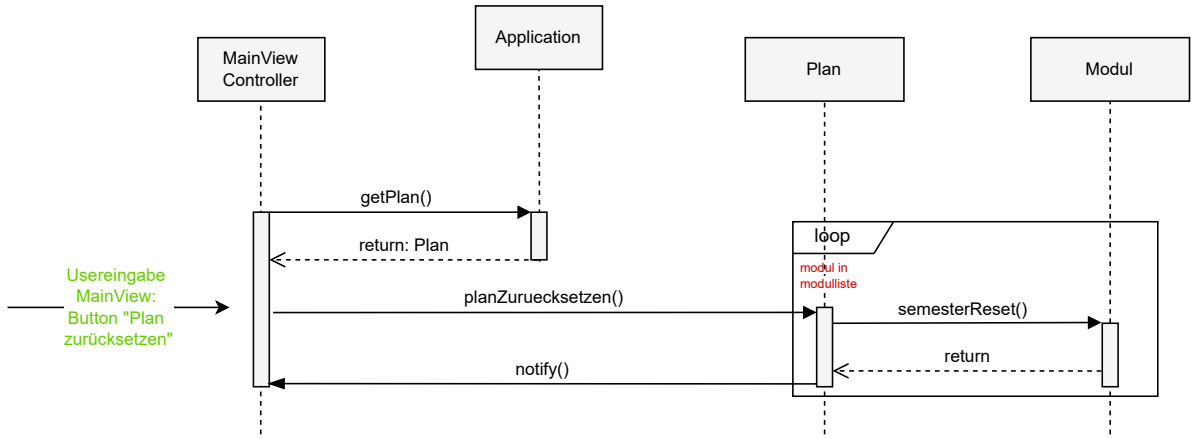

Abbildung 3-4 Sequenzdiagramm: Plan zurücksetzen

Möchte der User seinen Plan auf die ursprüngliche Darstellung zurücksetzen, so klickt er auf den „Plan zurücksetzen“ Button. Der MainView-Controller reagiert auf den User-Input, holt sich über getPlan() den in der Application hinterlegten Plan und ruft die planZurueecksetzen()-Methode des Plans auf. Diese setzt das Semester aller Module mit einer Schleife durch seine Modulliste durch den Methodenaufruf semesterReset() der einzelnen Module zurück. Der MainViewController wird nun vom Plan benachrichtigt, dass sich er geändert hat, woraufhin der MainViewController die Darstellung in der MainView aktualisiert.

Einschränkungen prüfen

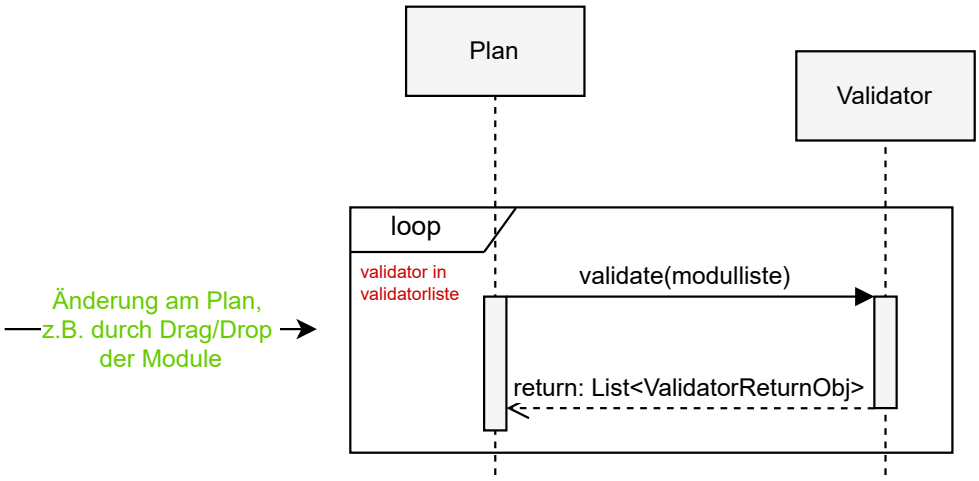

Abbildung 3-5 Sequenzdiagramm: Einschränkungen prüfen

Ändert der User z.B. durch Drag/Drop eines Moduls seinen Plan, so wird das Überprüfen der Einschränkungen des Plans eingeleitet. Der Plan hat eine Liste von Validatoren, die jeweils eine eigene validate()-methode haben. Im Plan wird in einer Schleife über die Validationen iteriert und diese Methode aufgerufen. Jeder Validator überprüft in der Methode den Plan entsprechend nach seinen Einschränkungen und gibt eine Liste von ValidatorReturnObj an den Plan zurück, die die Ergebnisse der Validierung repräsentieren.