5. Software-Design

In diesem Kapitel werden die Anforderungen aus dem vorletzten Kapitel in einem SysML Anforderungsdiagramm dargestellt. Daraufhin wird ein Software-Design in UML entworfen, das den Anforderungen gerecht wird.

5.1. SysML Anforderungsdiagramm

Mit SysML lassen sich Systeme, bestehend aus Hard- und Software, modellieren. Da in der vorliegenden Arbeit eine Lösung entstanden ist, die ausschließlich aus Software besteht, wurde nur das Anforderungsdiagramm der SysML verwendet. Alles andere wurde in UML modelliert.

Die Anforderung „beschreibt ein oder mehrere Eigenschaften oder Verhaltensweisen eines Systems, die stets erfüllt sein müssen.“ [Weilkiens 2014, S. 315] Im Anforderungsdiagramm wird jede Anforderung als Box dargestellt, in derem oberen Bereich der Stereptyp «requirement» steht. Die Anforderungen können Beziehungen untereinander haben. Nachfolgend sind die verwendeten Beziehungen dargestellt.

Enthältbeziehung

Die Enthältbeziehung „beschreibt, dass eine Anforderung in einer anderen enthalten ist.“ [Weilkiens 2014, S. 318] Dargestellt wird diese Beziehung durch eine Verbindung zweier Anforderungen, wobei an einem Ende der Verbindung ein Kreis mit einem Kreuz ist. In Abbildung 5.1 ist ein Beispiel dargestellt. Die Anforderungen „Aufgaben als erledigt markieren“ und „Aufgaben erstellen“ sind in der Anforderung „Aufgabenverwaltung“ enthalten.

Erfüllungsbeziehung

Die Erfüllungsbeziehung „beschreibt, dass ein Element eine Anforderung erfüllt“. [Weilkiens 2014, S. 319] Dargestellt wird diese Beziehung durch einen Pfeil, der mit dem Stereotyp «satisfy» beschriftet ist. In Abbildung 5.2 ist ein Beispiel dargestellt. Die Anforderung „Kalender“ erfüllt die Anforderung „Angabe möglicher Termine“.

Die Grafik wird durch eine Schönere ausgetauscht!

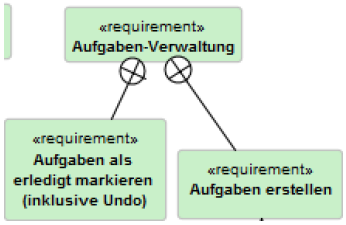

Abbildung 5.1.: SysML Anforderungsdiagramm - Enthältbeziehung

Die Grafik wird durch eine Schönere ausgetauscht!

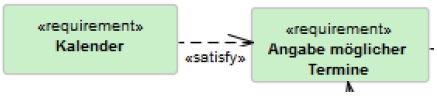

Abbildung 5.2.: SysML Anforderungsdiagramm - Erfüllungsbeziehung

Verfeinerungsbeziehung

Die Verfeinerungsbeziehung „beschreibt, dass ein Modellelement die Eigenschaften einer Anforderung detaillierter darstellt." [Weilkiens 2014, S. 325] Dargestellt wird diese Beziehung durch einen Pfeil, der mit dem Stereotyp «refine» beschriftet ist. In Abbildung 5.3 ist ein Beispiel dargestellt. Die Anforderung „Wiederkehrende Aufgaben erstellen" verfeinert die Anforderung „Aufgabe erstellen".

Die Grafik wird durch eine Schönere ausgetauscht!

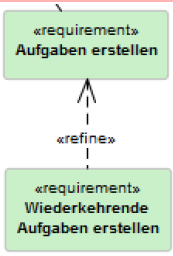

Abbildung 5.3.: SysML Anforderungsdiagramm - Verfeinerungsbeziehung

Anforderungsdiagramm

In Abbilung 5.4 ist das Anforderungsdiagramm mit allen Anforderungen und den Beziehungen untereinander zu sehen.

Es ist deutlich eine Clusterung zu erkennen. Die Anforderungen „Erstellung Dienstplan", „Erstellung Montsplan", „Team-Verwaltung", „Benutzerverwaltung" und „Aufgabenverwaltung" sind die zentralen Anforderungen, die weitere Anforderungen enthalten.

Die Grafik wird durch eine Schönere ausgetauscht! Die Monatsnavigation wird noch als Anforderung ergänzt.

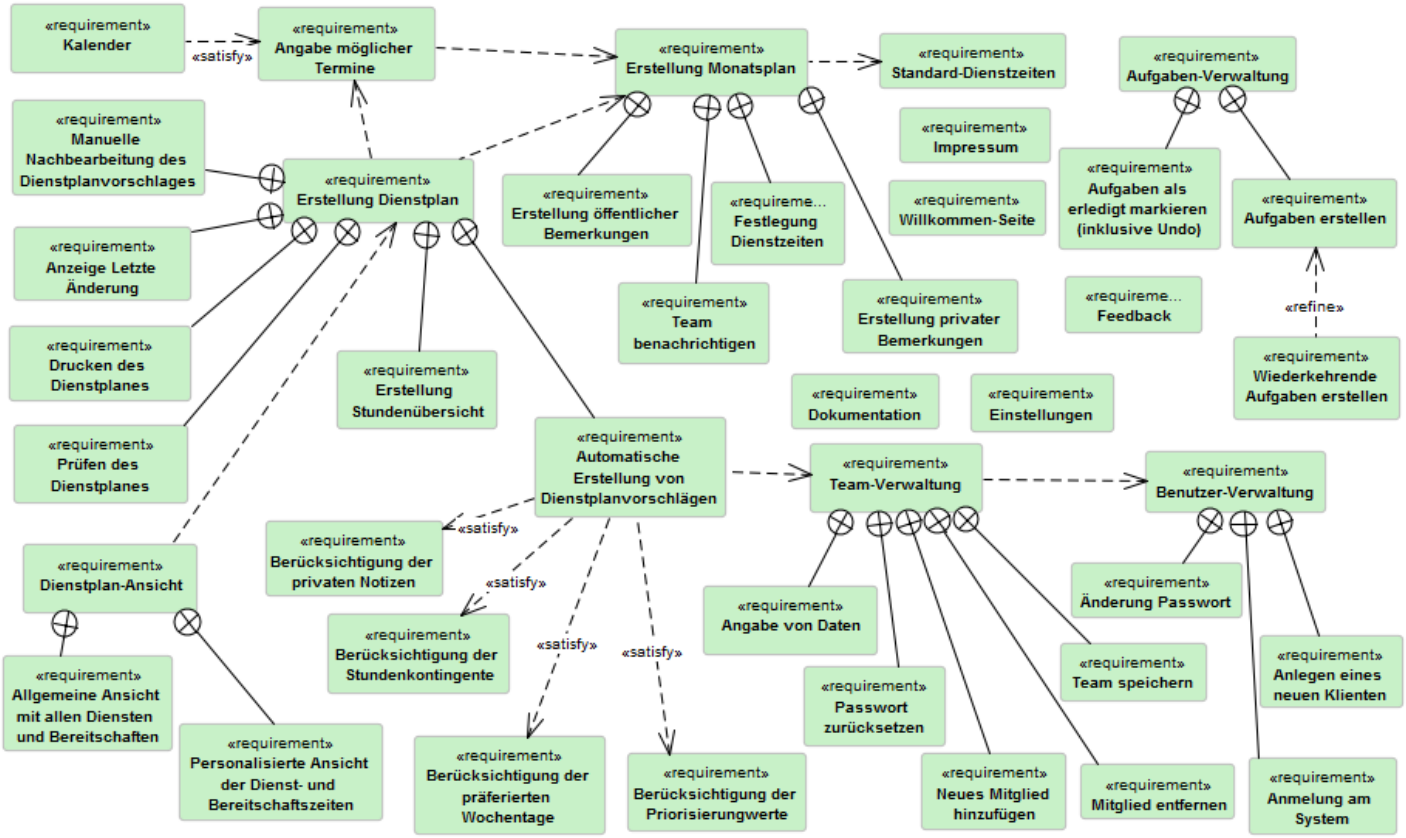

Abbildung 5.4.: SysML Anforderungsdiagramm

5.2. UML Komponentendiagramm

Mit dem Komponentendiagramm der UML kann man eine Grobarchitektur der Software darstellen. Die Diagrammform ist hilfreich um sich einen Überblick zu verschaffen. Nutzt eine Komponente Funktionalität einer anderen Komponente, so kann man dies durch Interfaces[11] darstellen.

Das Anforderungsdiagramm ist in Deutsch gehalten, da es noch unabhängig von einer Programmiersprache ist. Software schreibt der Autor ausschließlich englischsprachig. Nachfolgend sind die zentralen Anforderungen und die daraus abgeleiteten Pakatnamen dargestellt:

- Erstellung Dienstplan ⇒ Roster
- Erstellung Monatsplan ⇒ MonthOrganisation

[11] Der Autor bevorzugt den englischen Begriff Interface (deutsch: Zwischenstück), weil er ihn treffender findet als das häufig verwendete Wort „Schnittstelle“.

- Team-Verwaltung und Benutzer-Verwaltung ⇒ TeamOrganisation
- Aufgaben-Verwaltung ⇒ ToDoManager

Das Komponentendiagramm ist in Abbilung 5.5 zu sehen.

Grafik wird noch überarbeitet.

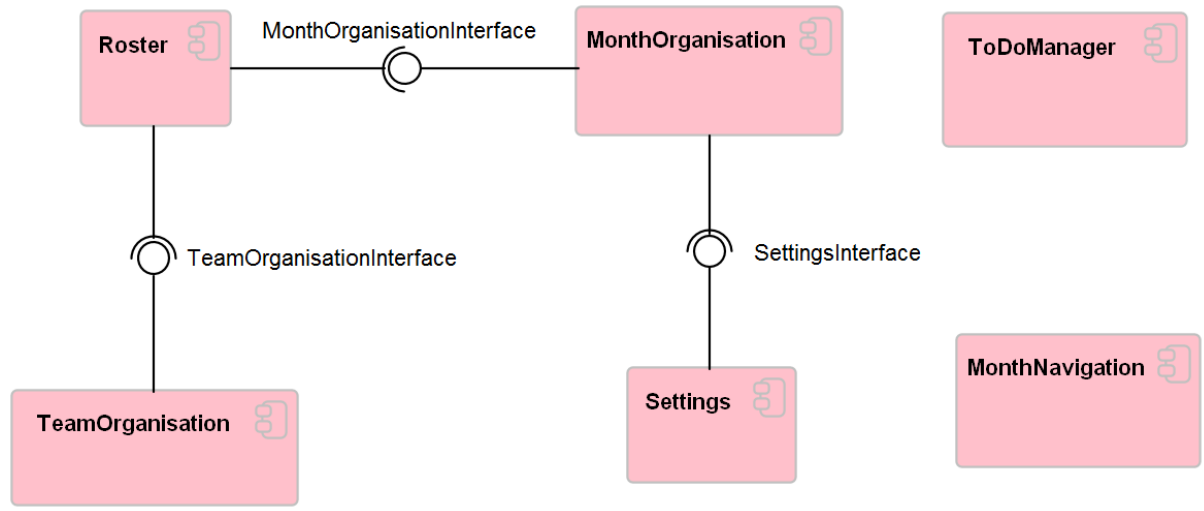

Abbildung 5.5.: UML Komponentendiagramm

Ziel der Modellierung war es, die Komponenten voneinander so unabhängig wie möglich zu machen. Zwei Komponenten (`ToDoManager` und `MonthNavigation`) sind völlig autark. Die restlichen vier Komponenten sind durch schlanke Interfaces verbunden. Beim `MonthOrganisationInterface` und `SettingsInterface` ist nur eine Methode zu implementieren. Beim `TeamOrganisationInterface` sind fünf Methoden zu implementieren.

5.3. UML Klassendiagramme

In diesem Abschnitt werden die Komponenten aus Abbildung 5.5 näher beleuchtet und die darin liegenden Klassen gezeigt.

Komponente `MonthNavigation`

Die Komponente `MonthNavigation` (siehe Abbildung 5.6) enthält nur die Klasse `MonthNavigation`.

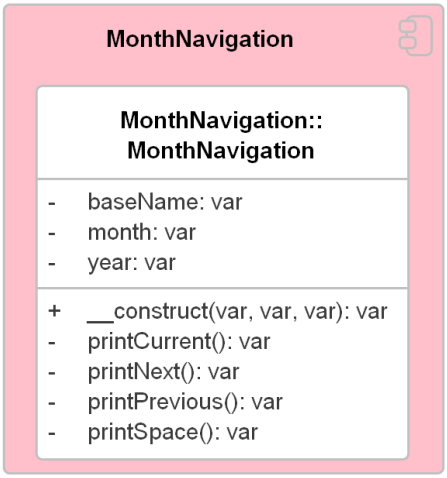

Abbildung 5.6.: Klassendiagramm der Komponente `MonthNavigation`

Die Klasse `MonthNavigation` ist für die Erstellung der Monats-Navigation (Implementierung siehe Abschnitt 6.6) zuständig.

Komponente `MonthOrganisation`

Die Komponente `MonthOrganisation` (siehe Abbildung 5.7) enthält die Klassen `MonthPlan`, `AssistanceInput`, `WorkingTimes` und `Day`.

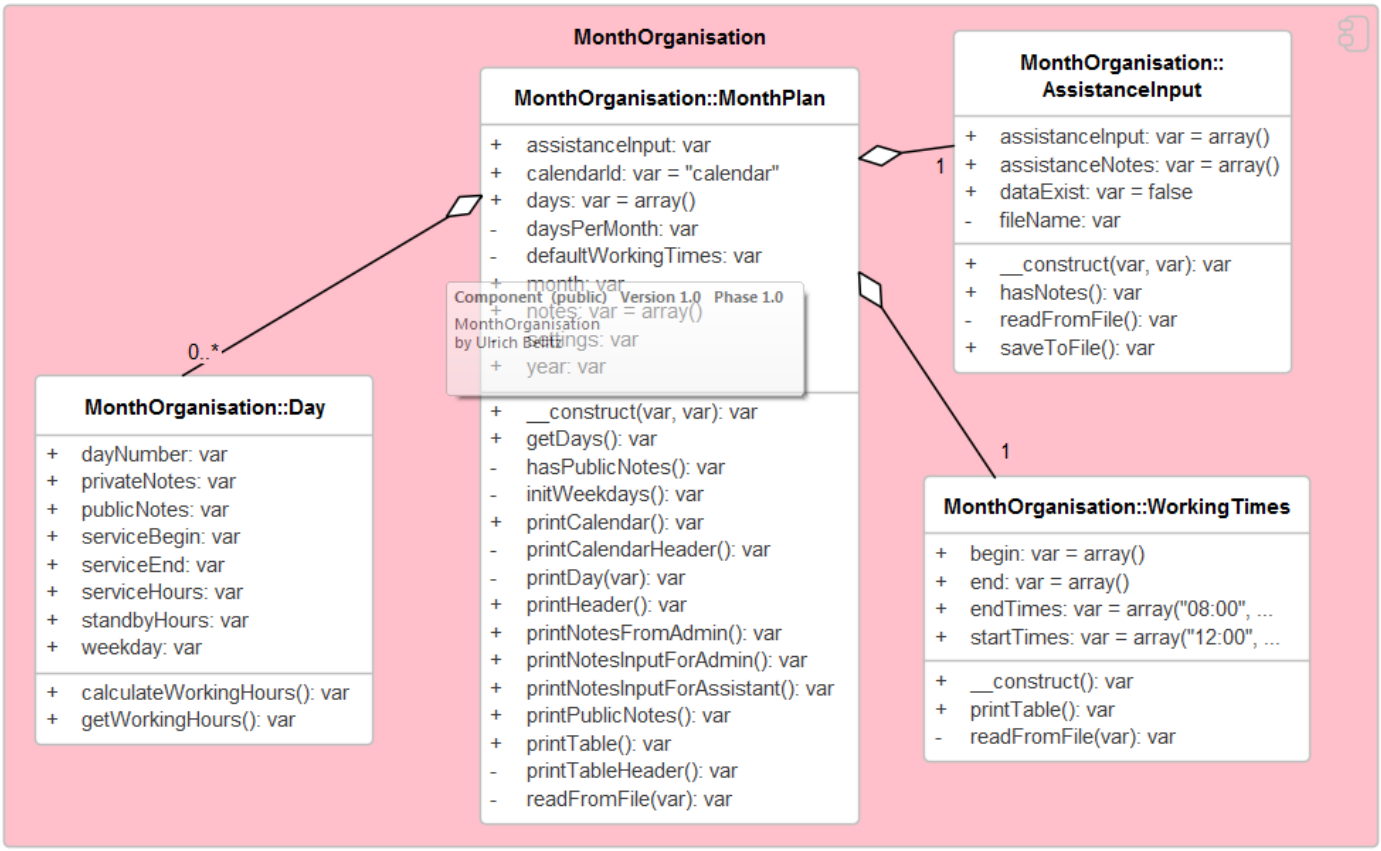

Abbildung 5.7.: Klassendiagramm der Komponente `MonthOrganisation`

Zentrale Klasse ist `MonthPlan`, die Instanzen von allen weiteren Klassen hat. Mit der Klasse `MonthPlan` werden alle Daten des Monatplans (Implementierung siehe Abschnitt 6.12) verwaltet. Die Kalendereingaben (Implementierung siehe Abschnitt 6.13) der Assistenten werden in der Klasse `AssistanceInput` hinterlegt. Die Standardarbeitszeiten (Implementierung siehe Abschnitt 6.10) werden von der Klasse `WorkingTimes` bereitgestellt. In der Klasse `Day` werden alle tagesbezogenen Daten gebündelt.

Komponente `Roster`

Die Komponente `Roster` (siehe Abbildung 5.8) enthält nur die Klasse `Roster`.

Abbildung 5.8.: Klassendiagramm der Komponente `Roster`

Die Klasse `Roster` wird bei der Dienstplanerstellung (Implementierung siehe Abschnitt 6.14) verwendet.

Komponente `Settings`

Die Komponente `Settings` (siehe Abbildung 5.9) enthält die Klassen `Passwords` und `Settings`.

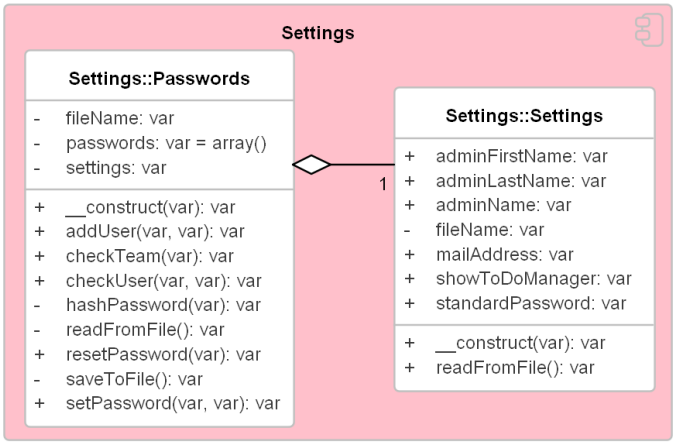

Abbildung 5.9.: Klassendiagramm der Komponente `Settings`

Mit der Klasse `Settings` werden die Einstellungen (Implementierung siehe Abschnitt 6.9) des Assistenzplaners verwaltet. In der Klasse `Passwords` sind alle Funktionen vorhanden, die für die Benutzerverwaltung (Implementierung siehe Abschnitt 6.3) benötigt werden.

Komponente `TeamOrganisation`

Die Komponente `TeamOrganisation` (siehe Abbildung 5.10) enthält die Klassen `Team` und `TeamMember`.

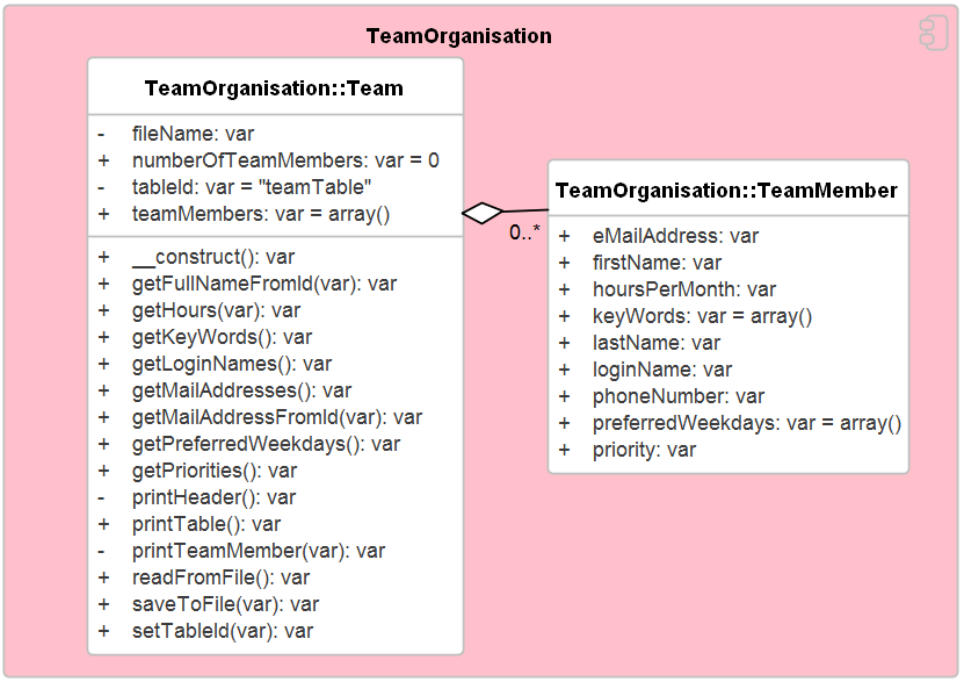

Abbildung 5.10.: Klassendiagramm der Komponente `TeamOrganisation`

In der Klasse `Team` ist die Funktionalität für die Team-Verwaltung (Implementierung siehe Abschnitt 6.11) implementiert. `Team` hat soviele Instanzen von `TeamMember`, wie das Assistenz-Team Mitglieder hat. Die Klasse TeamMember ist eine reine Datenhaltungsklasse[12] ohne Funktionalität.

Komponente `ToDoManager`

Die Komponente `ToDoManager` (siehe Abbildung 5.11) enthält die Klassen `ToDoManager` und `ToDoItem`.

[12] Alle Member sind `public`.

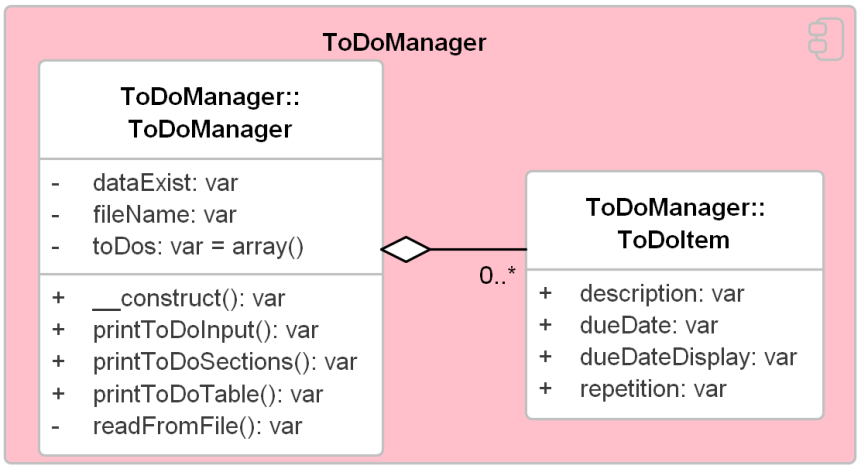

Abbildung 5.11.: Klassendiagramm der Komponente `ToDoManager`

In der Klasse `ToDoManager` ist die Funktionalität für die Aufgaben-Verwaltung (Implementierung siehe Abschnitt 6.16) implementiert. `ToDoManager` hat soviele Instanzen von `ToDoItem`, wie Aufgaben in der Aufabenliste hinterlegt sind. Die Klasse `ToDoItem` ist eine reine Datenhaltungsklasse ohne Funktionalität.