

Design-Dokument

Regalisator Ultimate Edition 1.0

Dokumentenversion: 1.3

Leo Back, Clemens Klein, Julian Sauer, Joshua Barth

Abgabedatum: 16. Juni 2017

Versionshistorie

Nummer	Datum	Uhrzeit	Autor	Änderungen
1.0	14.06.2017	17:00	Alle Beteiligten	Aufbau des Grunddokuments
1.1	14.06.2017	18:30	Leo Back	1.1.2. Einlesevorgang hinzugefügt
1.2	14.06.2017	18:40	Joshua Barth	Abb. 18 + 19 hinzugefügt
1.3	15.06.2017	12:00	Julian Sauer, Clemens Klein	Überarbeitung Abb 11.

Inhaltsverzeichnis

1. Lösungsstrategie	3
1.1. Bausteinsicht	3
1.1.1. Paketdiagramm	3
1.1.2. View: Detailsicht	3
1.1.3. Logik: Detailsicht	4
1.1.4. Persistenz: Detailsicht	5
1.2. Laufzeitsicht	6
1.2.1. Startvorgang	6
1.2.2 Einlesevorgang	7
1.2.3. Paket einfügen	8
1.2.4. Brett einfügen	9
1.2.5. Paketplatz finden	10
1.2.6. Paket verschieben	11
1.3. Detail: Algorithmus "Prüfe Paket"	12
2. Verknüpfung der UI-Skizzen mit den Funktionen	13
2.1. MenuView Aufbau	13
2.2. Regal Konfigurationsmodus	14
2.3. Paket Erstellen Modus	15
2.4. Vorlagen Modus	16
2.5. Suchen Modus	16
3. Querschnittliche Konzepte	17
3.1. Rastersystem	17
4. Qualitätsanforderungen	18
5. Glossar	18
6. Abbildungsverzeichnis	19

1. Lösungsstrategie

1.1. Bausteinsicht

Die Bausteinsicht bietet ein Paketdiagramm zur Gesamtübersicht und zwei Detailsichten, die die jeweiligen Pakete detaillierter zeigen

1.1.1. Paketdiagramm (Abb. 1)

In der ersten Ebene durchleuchten wir die View, Logik und Persistenz. In den drei Pakten sieht man die Klassen des Systems. Auf diese und deren Zusammenspiel wird in den jeweiligen Detailsichten weiter eingegangen.

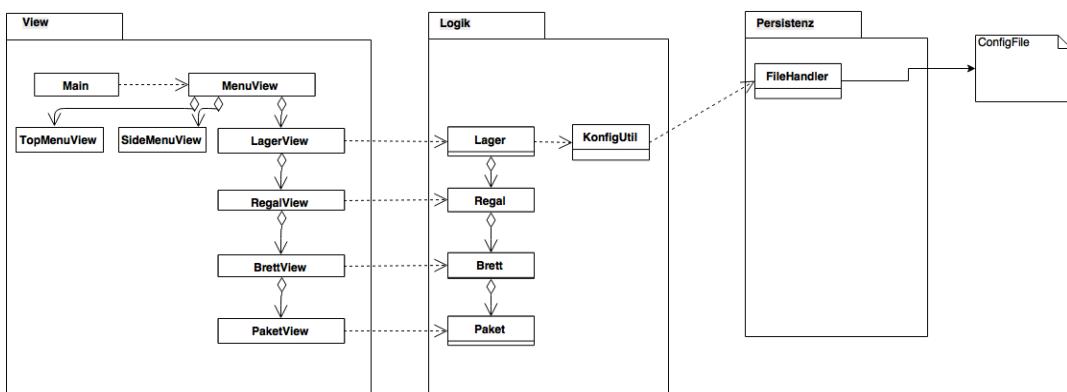


Abbildung 1: Paketdiagramm, UML-Diagramm

1.1.2. View: Detailsicht (Abb. 2)

In der Detailsicht der View wird eine Trennung zwischen Menü-Elementen und den Elementen der Lageransicht ersichtlich. In der MenuView werden LagerView, TopMenu und SideMenuView verwaltet, worauf später noch eingegangen wird.

Das SideMenu bietet dem User Zugriff auf die elementaren Methoden zur Lagerverwaltung, also die Möglichkeit Regale oder Pakete zu erstellen, Vorlagen zu speichern und Pakete zu suchen. Die unterschiedlichen Views der Seitenleiste sind austauschbar. Die SuchenView, benötigt einen Listener zur PaketView (um das aktuell ausgewählte Paket anzuzeigen), die das untere Ende der Aggregationskette unter der Lagerview bildet.

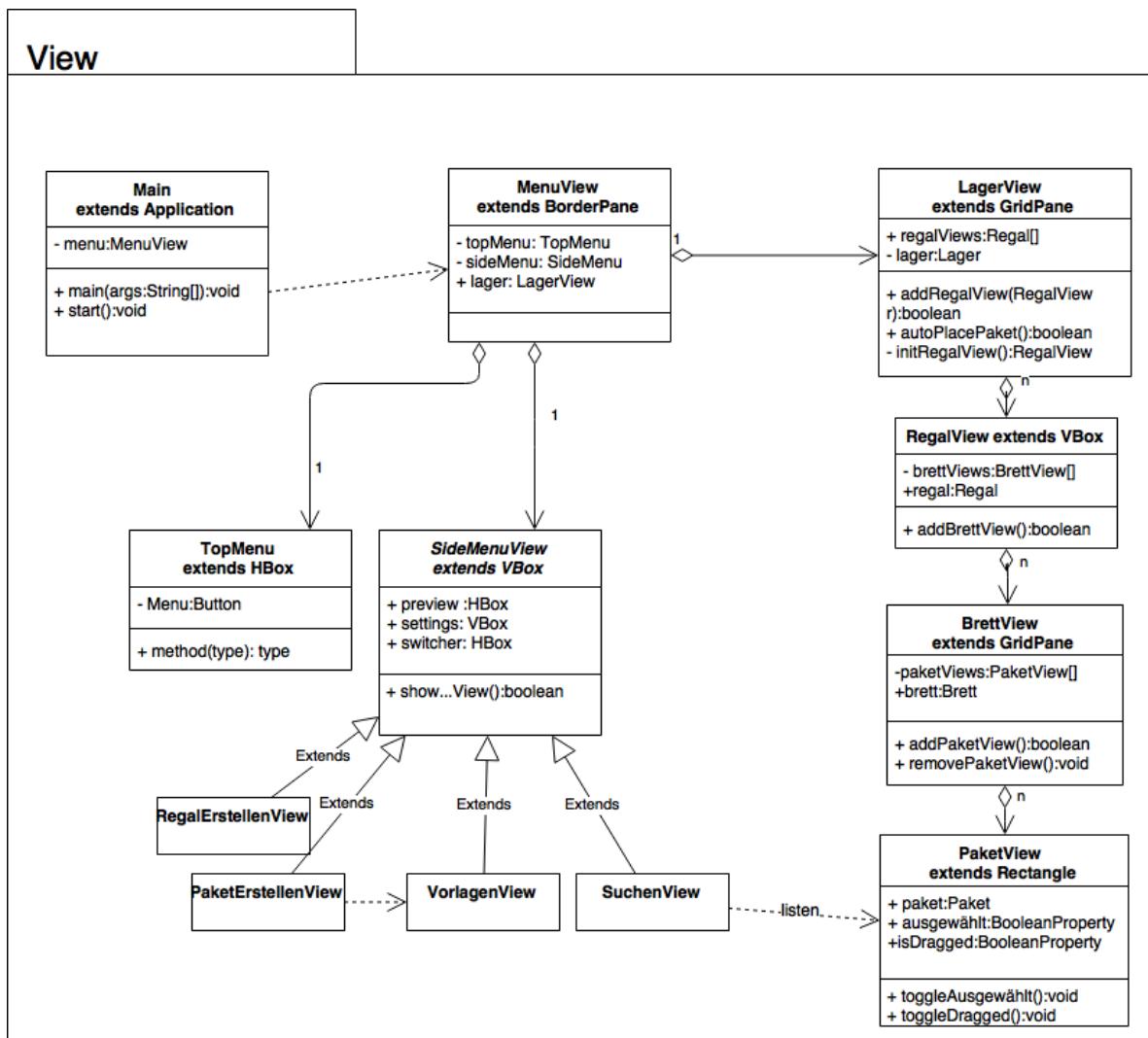


Abbildung 2: View: Detailsicht, UML-Diagramm

1.1.3. Logik: Detailsicht (Abb. 3)

Auch in der Logik haben wir eine Aggregationskette von Lager bis Paket. Ein Lager besteht aus einem, oder mehreren, Regalen, ein Regal aus einem, oder mehreren, Brettern, usw. Im Lager selbst werden alle erstellten Regale in einer Map verwaltet, jedes Regal verwaltet wiederum seine Bretter und jedes Brett seine Pakete. Die Gesamtkonfiguration des Lagers kann dann per KonfigUtil an den FileHandler weitergegeben werden um diese abzuspeichern.

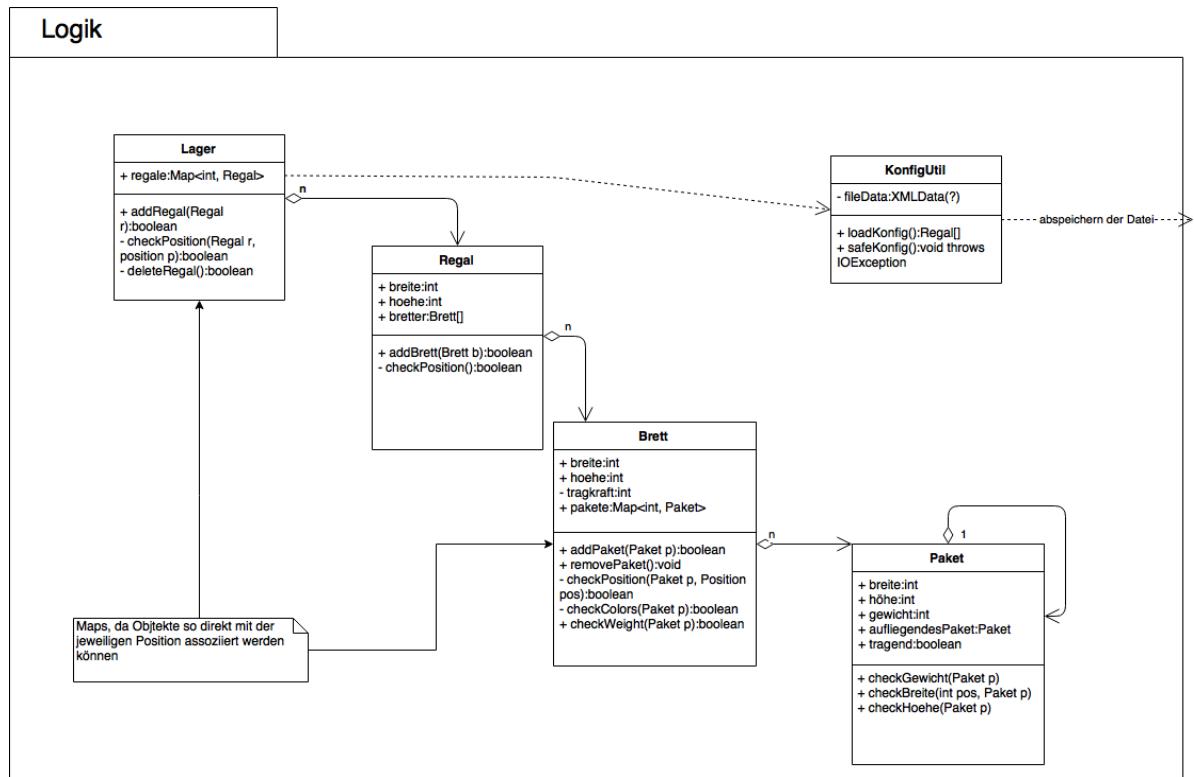


Abbildung 3: Logik: Detailsicht, UML-Diagramm

1.1.4. Persistenz: Detailsicht (Abb. 4)

In der Persistenzschicht befindet sich ein FileHandler, der in der Lage ist XML-Files per saveFile-Methode abzuspeichern, oder per loadFile-Methode einzulesen.

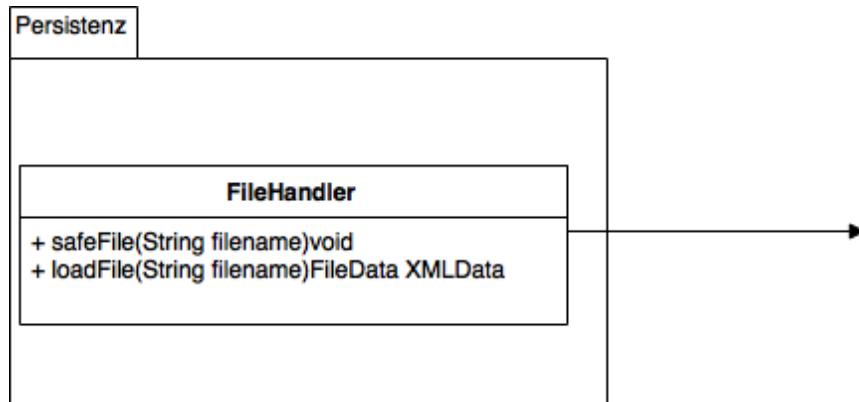


Abbildung 4: Persistenz: Detailsicht, UML-Diagramm

1.2. Laufzeitsicht

1.2.1. Startvorgang (Abb. 5)

Während des Startvorgangs werden durch die **Main**-Klasse **LagerView** und **Lager** initialisiert. Das **Lager** lädt in seinem Konstruktor eine (eventuell) gespeicherte Konfiguration mithilfe des **KonfigUtil** (siehe hierzu 1.2.2 "Einlesevorgang").

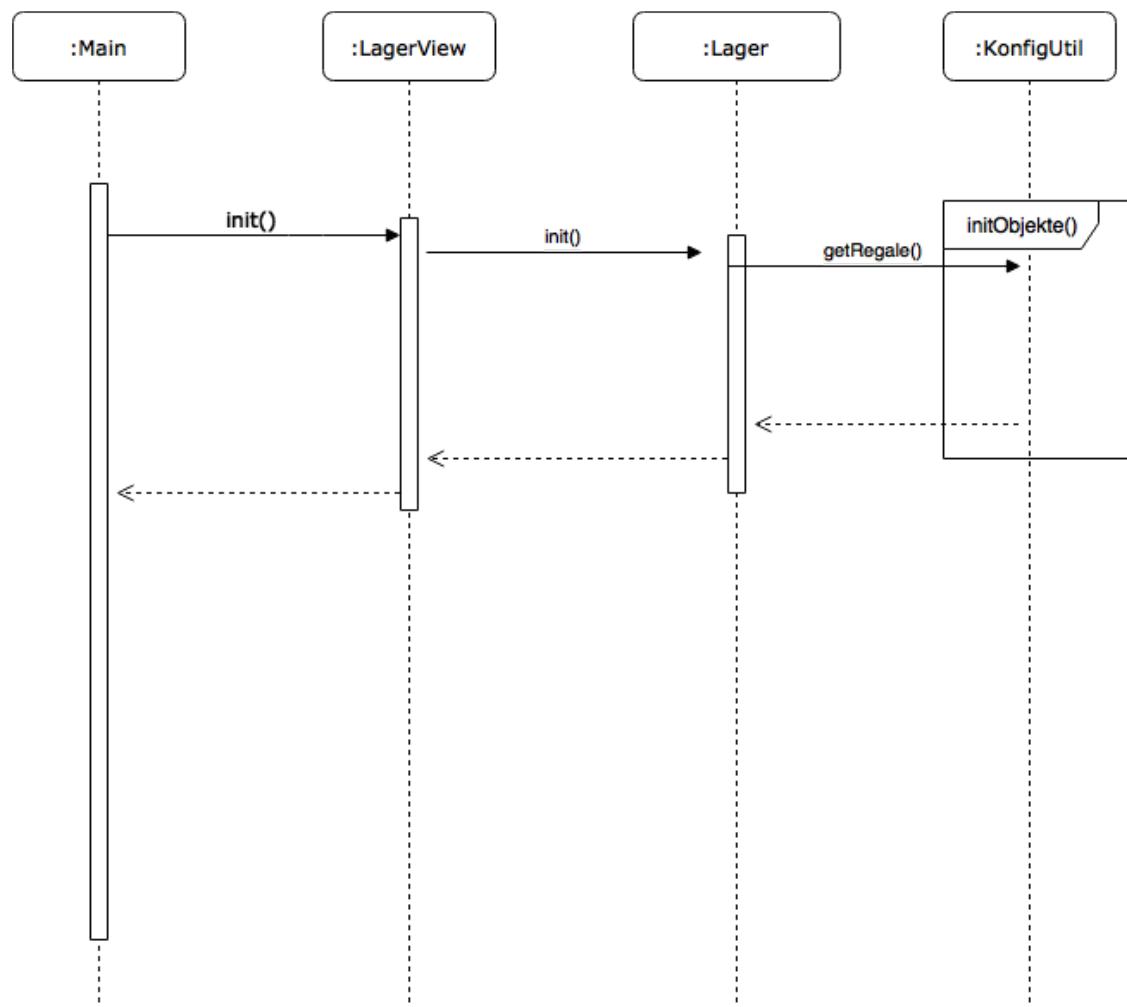


Abbildung 5: Laufzeitsicht: Startvorgang, UML-Diagramm

1.2.2 Einlesevorgang (Abb. 6, 7)

Das KonfigUtil erhält vom FileHandler ein iterierbares Datenobjekt. Dieses basiert auf einer XML-Darstellung. In dieser sind Regale, deren Bretter und deren Pakete verschachtelt. So können alle Regale vollständig eingelesen werden mithilfe einer dreifachen Schleife (siehe Abb. 6).

Pseudocode

```
def lese_ein(regalData):
    regale = []
    for regalData in data:
        bretter = []
        for brettData in regalData:
            pakete = []
            for paketData in RegalData:
                # parse paketdaten und erstelle paket...
                pakete.add(paket)
            # parse brettdaten und erstelle Bett...
            brett.add(pakete)
            bretter.add(brett)
        # parse Regaldaten und erstelle Regal...
        regal.add(bretter)
        regale.add(regal)
    return regale
```

Abbildung 6: Pseudocode: Startvorgang

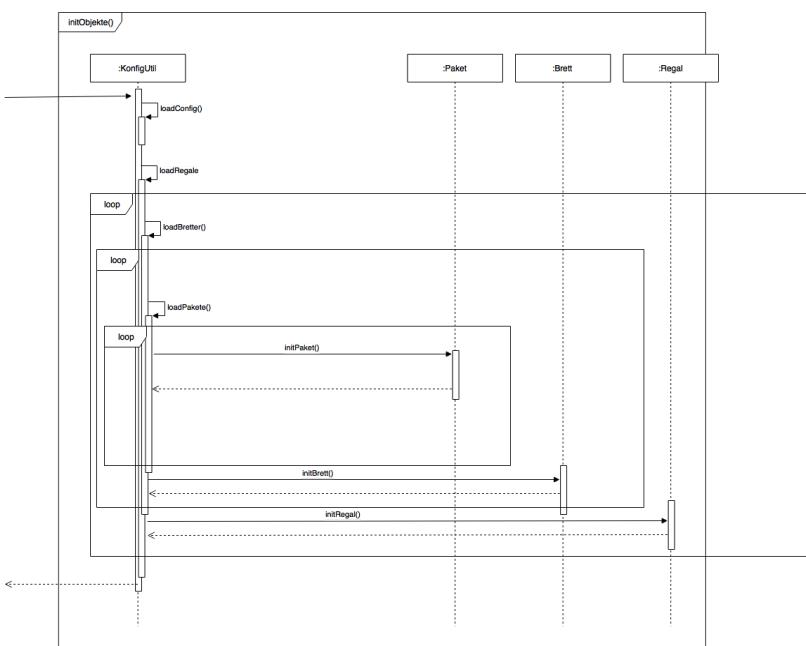


Abbildung 7: Laufzeitsicht: Einlesevorgang, UML-Diagramm

1.2.3. Paket einfügen (Abb. 8)

Um ein Paket einzufügen wird zuerst die addPaketView-Methode in der BrettView aufgerufen. Die BrettView holt sich daraufhin das Datenobjekt der hinzuzufügenden PaketView und versucht das Paket seinem eigenen Datenobjekt "Brett" hinzuzufügen. Das Brett selbst überprüft nun ob das Paket mit seinen Maßen Platz an der gewünschten Position findet, ob die Farbe des Pakets mit den umliegenden Paketen harmoniert und ob es das Gewicht des Pakets noch tragen kann. Die Eigenschaften des Pakets bekommt es zuvor per Getter-Methoden vom Paket selbst. Falls die Überprüfung positiv verlaufen ist, kann das Paket nun hinzugefügt werden und die BrettView benachrichtigt werden, damit diese ihren Inhalt aktualisiert.

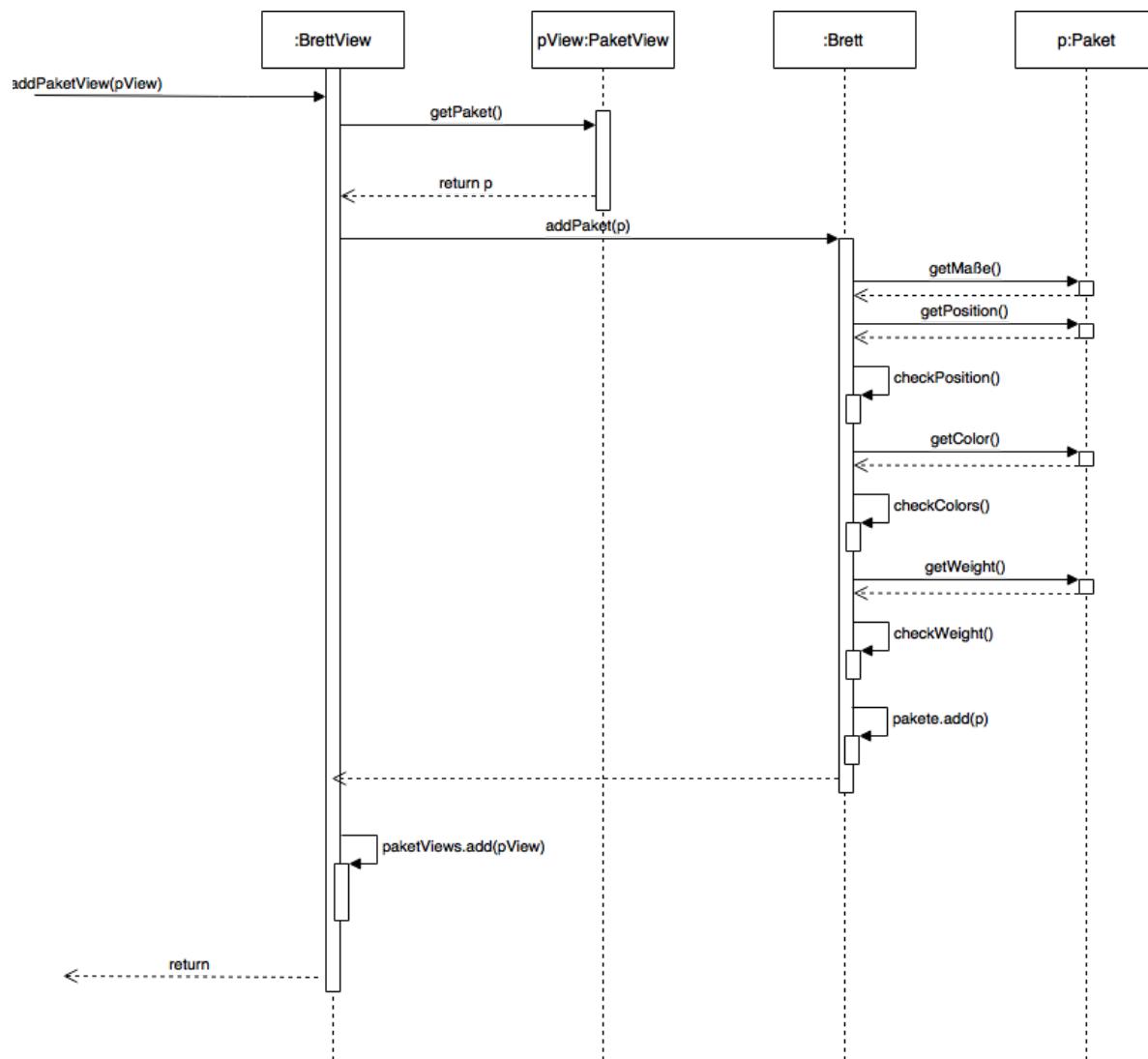


Abbildung 8: Laufzeitsicht: Paket einfügen, UML-Diagramm

1.2.4. Brett einfügen (Abb. 9)

Soll ein Brett eingefügt werden, so wird die addBrettView-Methode der RegalView aufgerufen. Diese holt sich die Informationen des einzufügenden Brettes per Getter-Methode aus der BrettView. Anschließend ruft sie die addBrett-Methode der Regal-Klasse auf um das Brett einzufügen. Die Regal-Klasse nutzt Getter-Methoden der Brett-Klasse um Höhe, Breite und Tragkraft des Brettes zu erhalten und mit Hilfe dieser zu prüfen, ob das Brett an dieser Stelle eingefügt werden kann. Liefert die Prüfmethode checkPosition() true zurück, so wird das Brett per add-Methode der Brett-Klasse eingefügt.

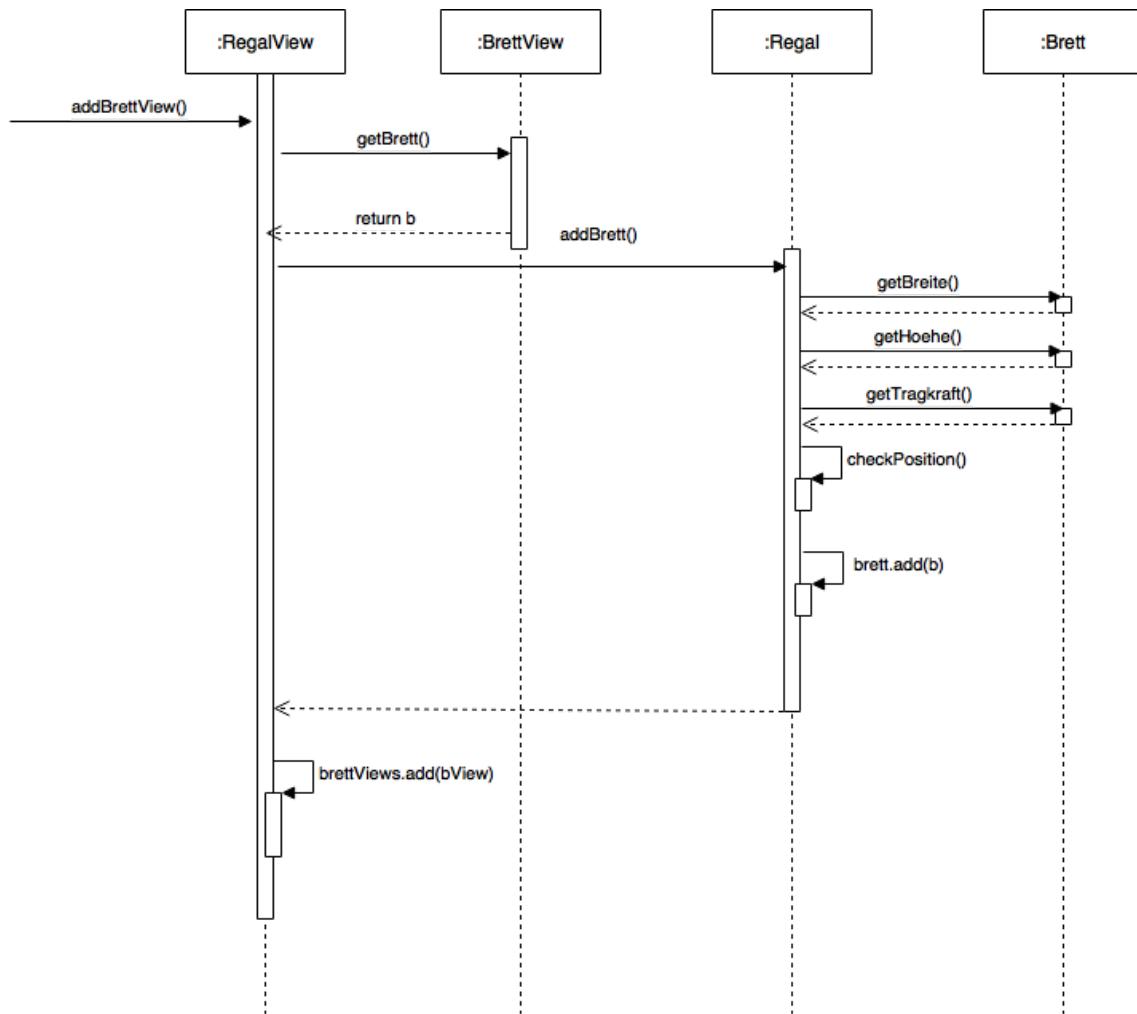


Abbildung 9: Laufzeitsicht: Brett einfügen, UML-Diagramm

1.2.5. Paketplatz (automatisch) finden (Abb. 10)

Ziel ist es, den ersten Platz (egal welcher) für ein Paket zu finden, der alle erforderlichen Kriterien (Farbe, Gewicht, Platz) erfüllt. Hierbei iteriert das Programm durch alle verfügbaren Bretter und versucht das Paket an jeder Position des Rasters (vgl. Kapitel "RasterSystem" 3.1) einzufügen. Da die Funktion **addPaketView()** einen Boolean zurückliefert lässt sich leicht nachvollziehen, ob die Operation erfolgreich war und die Schleife beendet werden kann.

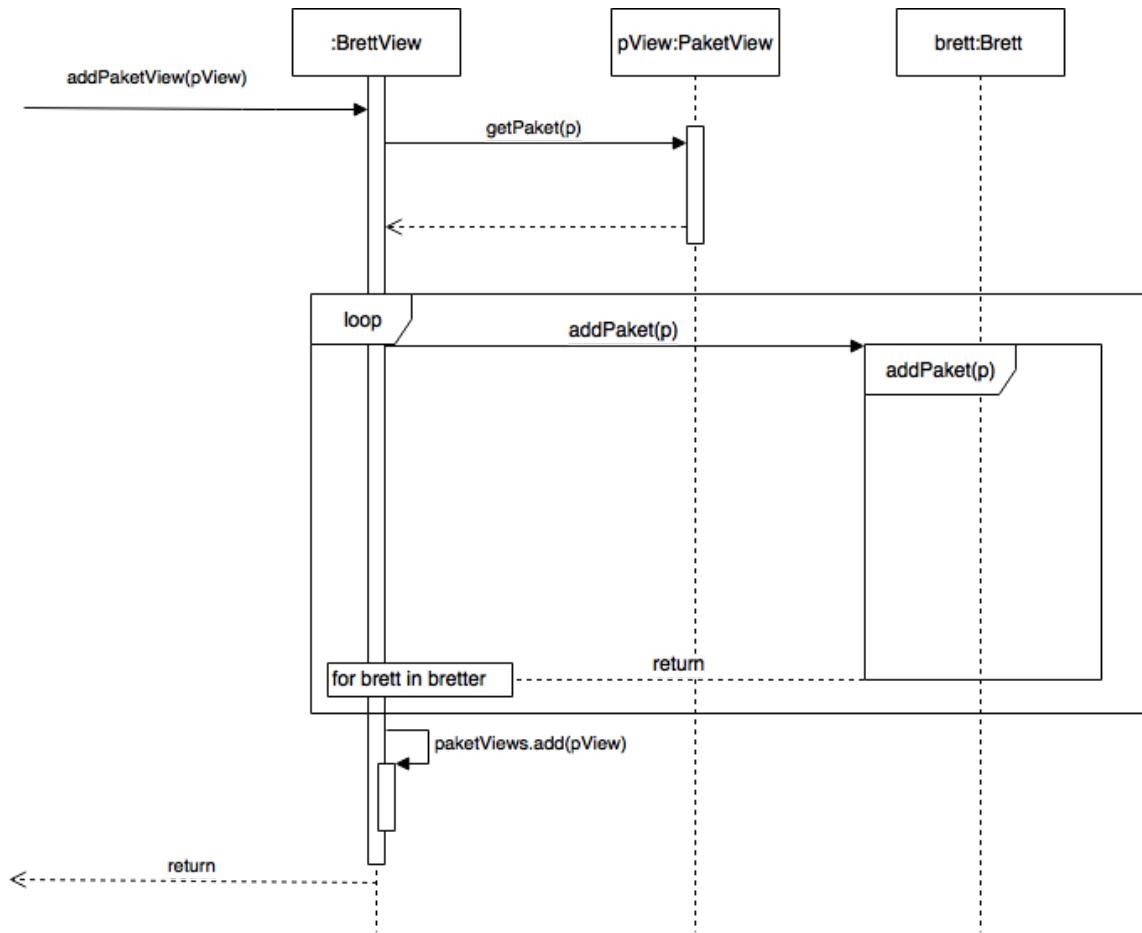


Abbildung 10: Laufzeitsicht: Paket finden, UML-Diagramm

1.2.6. Paket verschieben (Abb. 10)

Das Verschieben von Paketen erfolgt vom Nutzer durch Drag and Drop. Die PaketView hat einen Listener, der bei erkannten Drag-Vorgang die BooleanProperty **dragged** auf true stellt. Die BrettView, die dieses Paket enthält, hat einen Listener auf **dragged**. Wird dieses true, so entfernt die BrettView das entsprechende Paket aus seinem Datenobjekt und seiner Darstellung.

BrettViews haben einen Listener, der bei erkanntem Drop-Vorgang das gedroppte Paket in sich selbst einfügt (vgl. "Paket einfügen" 1.2.3.).

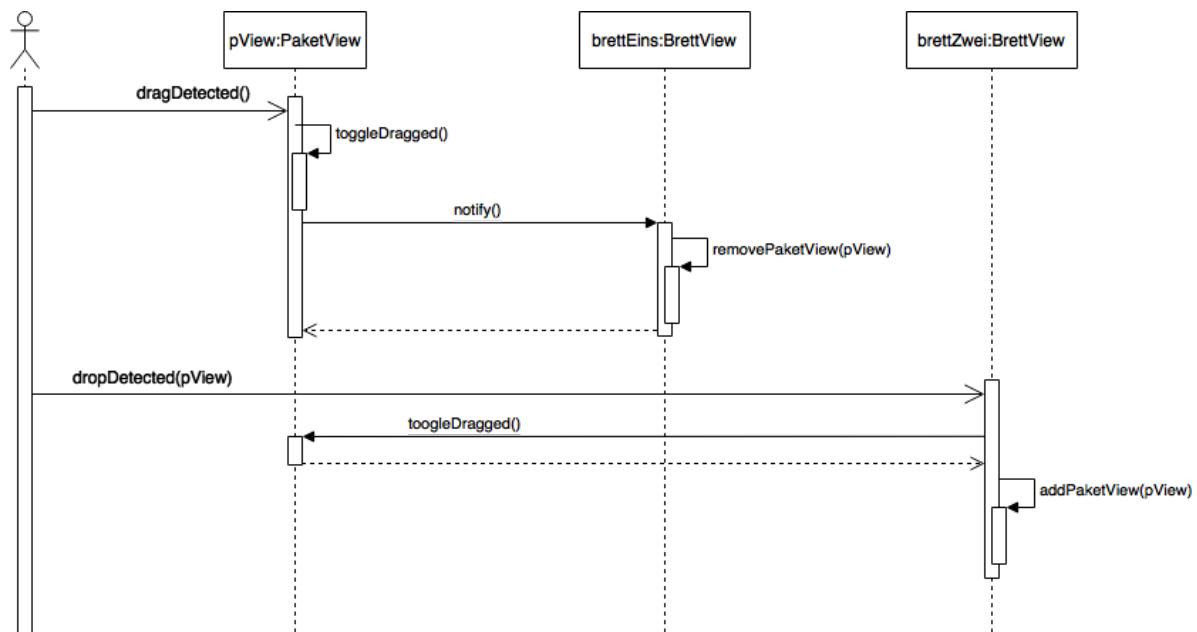


Abbildung 11: Laufzeitsicht: Paket verschieben, UML-Diagramm

1.3. Detail: Algorithmus "Prüfe Paket"

Der Algorithmus zur Prüfung der Positionierung eines Pakets stellt das Herzstück des Programms dar. Er muss sowohl Breite und Höhe des einzufügenden Paketes als auch, nach Bestätigung der Einfügeposition, die verfügbare Tragkraft aller Pakete reduziert werden. Hier bietet sich Rekursion als elegante Lösung an. Der Algorithmus prüft zunächst im Brett, ob alle Kriterien erfüllt sind. Anschließend geht er rekursiv alle aufliegenden Pakete durch. Es werden nun die Kriterien "Tragkraft" und "Breite" geprüft. Hat er das oberste erreicht, die verfügbare Höhe geprüft und ein neues Paket darauf platziert, reduziert er in allen darunter liegenden entsprechend die Tragkraft.

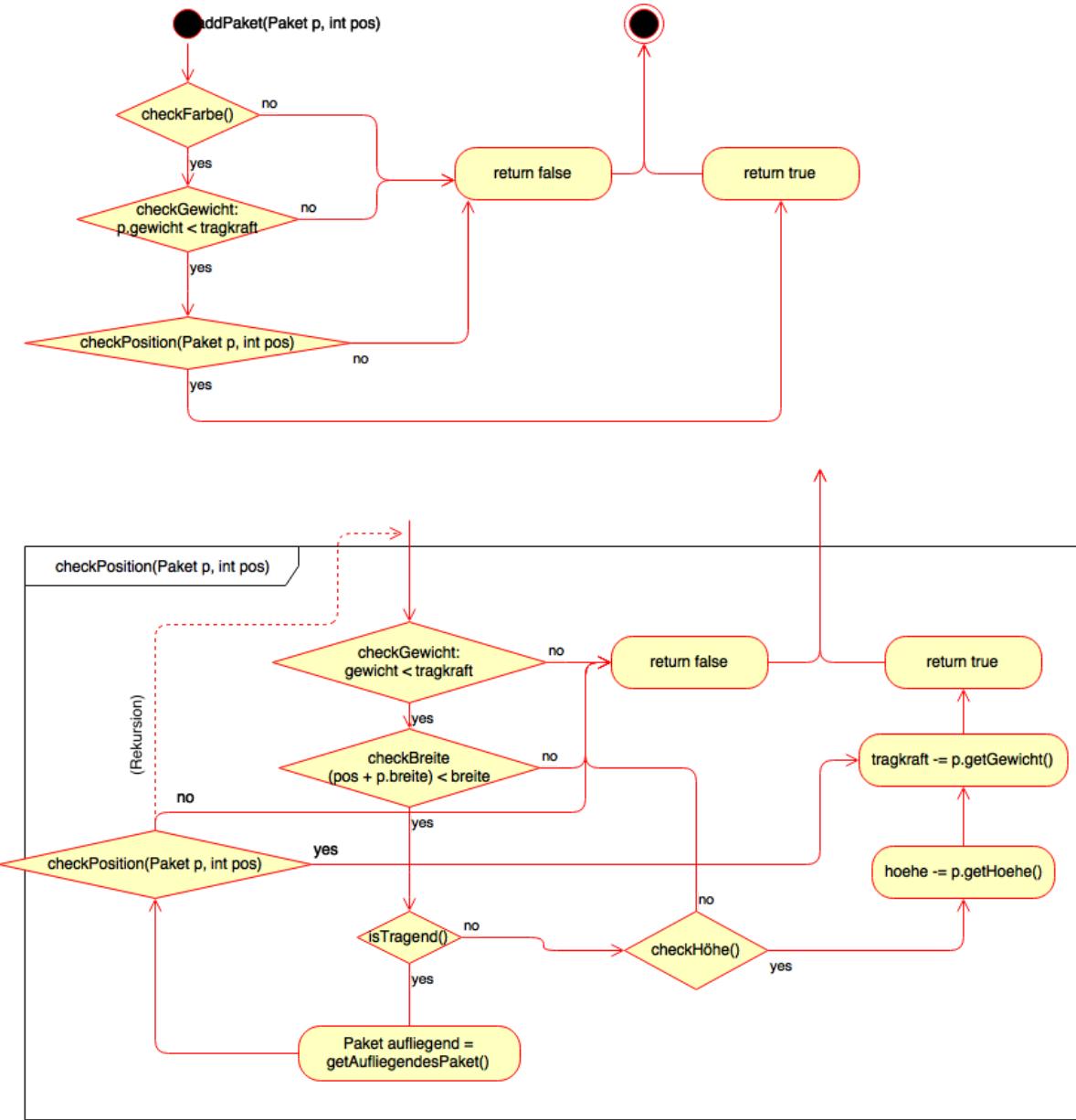


Abbildung 12: "Prüfe Paket"-Algorithmus

2. Verknüpfung der UI-Skizzen mit den Funktionen

Im folgendem Abschnitt werden die UI-Wireframes mit den vorgestellten Funktionen verknüpft und der Aufbau kurz erläutert. Die Grafiken sind dieselben, die auch in der Spezifikation zu finden sind. Alle nachfolgenden GUI Elemente, deren Beziehungen und wichtige Methoden sind in der *View-Detaillsicht* (1.1.3 und Abb. 2) zu finden.

Die Beschreibung der einzelnen Wireframes findet man in der Spezifikation unter (*4. Benutzungsschnittstelle*).

2.1. MenuView Aufbau

Nachfolgend (Abb. 13) ist der grobe Aufbau der *MenuView* dargestellt.

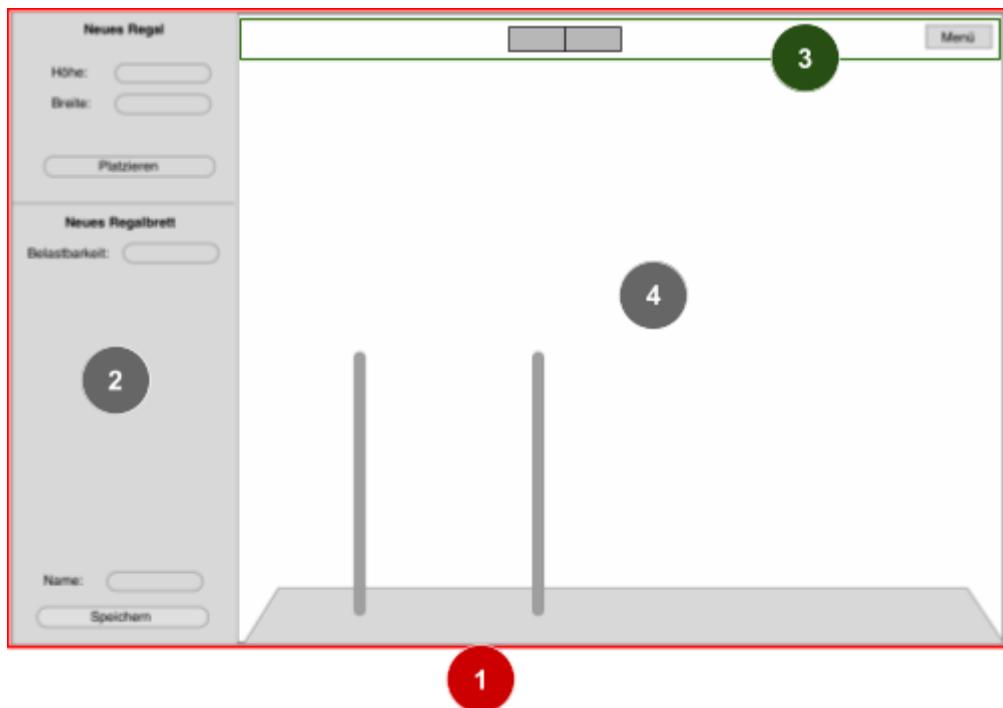


Abbildung 13: Funktionsverknüpfung: MenuView

1. **MenuView:** Das zentrale Layout Element, welches alle Unterelemente zusammenfasst (roter Rahmen) als *JavaFX BorderPane*.
2. **SideMenuItemView:** Die Seitenleiste, die sich an das aktuelle Geschehen anpasst, als *JavaFX VBox*. Um die Seitenleiste unterschiedlich darstellen zu können, wird für jedes Menü eine eigene View erstellt, die von *SideMenu* erben (siehe Abb. 2).
3. **TopMenuItemView:** Die Obere Leiste, die Vereinzelte Buttons und Menüelemente angezeigt als *JavaFX HBox*.
4. **LagerView:** Der dynamische Bereich in dem das Lager mit Regalen oder zusätzlich mit Inhalt angezeigt wird, als *JavaFX GridPane*.

2.2. Regal Konfigurationsmodus

Nachfolgend (Abb. 14) ist der “Regal Konfigurationsmodus” und die vorhandenen Methodenverknüpfungen dargestellt.

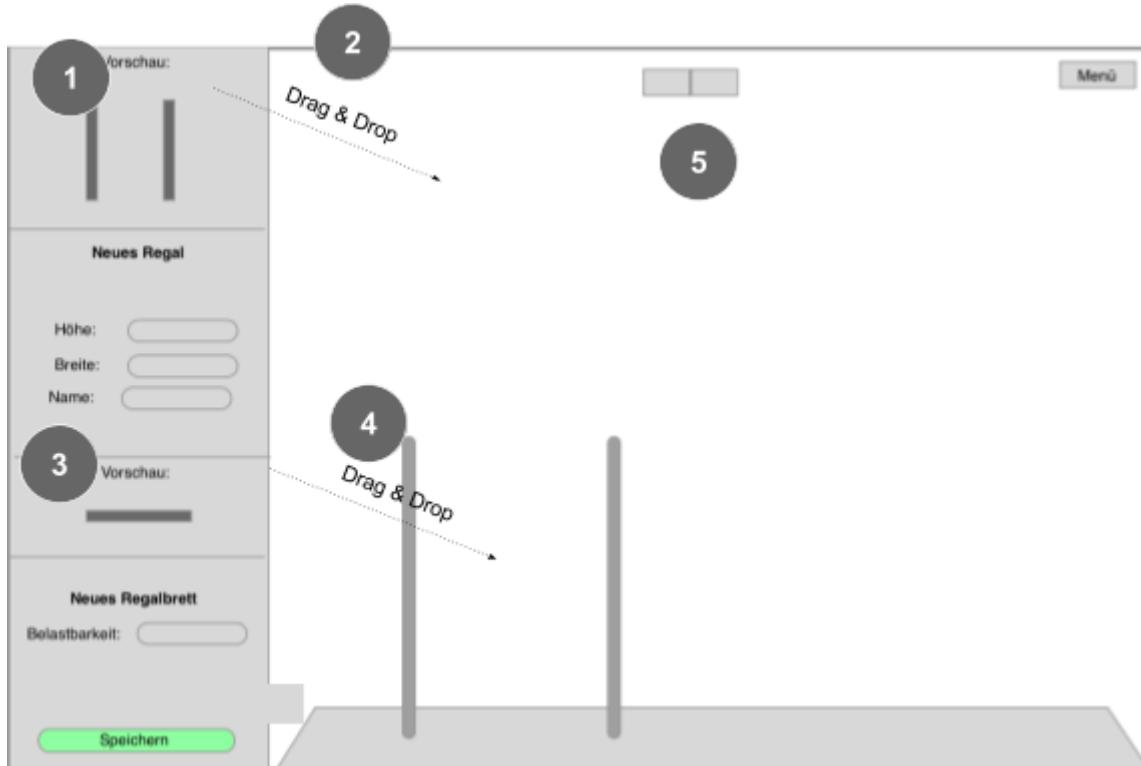


Abbildung 14: Funktionsverknüpfung: Regal Konfigurationsmodus

1. Inhalt der Textboxen (Höhe, Breite, ...) ist an die jeweiligen Properties des aktuellen RegalViews gebindet (kommt von “Binding”). Die RegalView wird beim öffnen des Menüs automatisch erstellt (mit Standardwerten).
2. *addRegalView()* - **LagerView**
3. Inhalt der TextBox (Belastbarkeit) ist an die jeweilige Property des Datenobjekts der aktuellen BrettViews gebindet (*Vorschaubereich mitte Abb. 14*). Die BrettView wird beim öffnen des Menüs automatisch erstellt (mit Standardwerten).
4. *addBrettView()* - **RegalView**: die RegalView, auf der das Brett “gedropped” wurde, erkennt dies und fügt es entsprechend an der Position hinzu.
5. *showPaketErstellenView()* - **SideMenu**: Dieser Button lässt zwischen dem “RegalModus” und dem “PaketModus” wechseln, falls man als Admin angemeldet ist (*Siehe Spezifikation Kapitel “Anwendungsfälle”*). Hier wird sowohl der Inhalt der Seitenleiste ausgetauscht, als auch der Inhalt der *LagerView* ergänzt.

2.3. Paket Erstellen Modus

Nachfolgend (Abb. 15) ist der “Paket Erstellen Modus” und die vorhandenen Methodenverknüpfungen dargestellt.

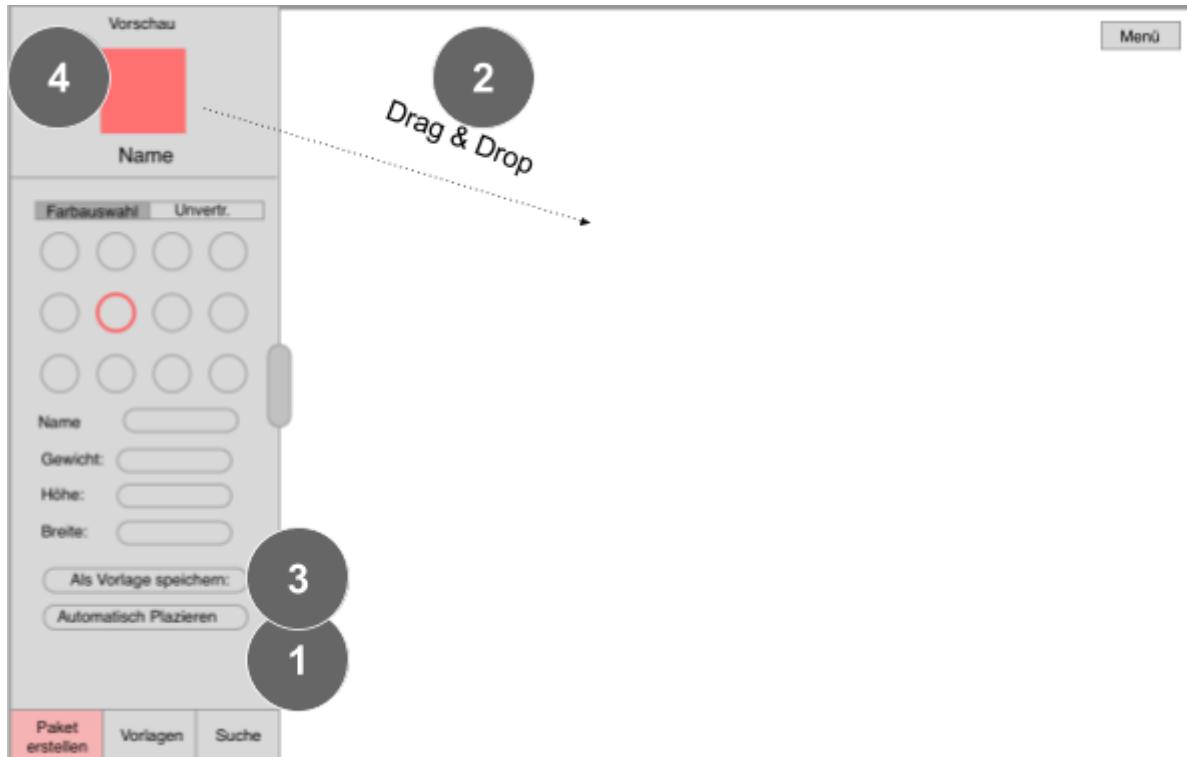


Abbildung 15: Funktionsverknüpfung: Paket Erstellen Modus

1. *autoPlacePaket()* - **LagerView**: die LagerView leitet den Aufruf weiter an das Lager (Logikobjekt), welches einen freien Platz zum Paket platzieren zurückgibt. Der weitere Ablauf ist im entsprechendem Laufzeidiagramm (1.2.4) zu finden.
2. *addPaketView()* - **BrettView**: ein Regal erkennt per Listener, ob ein PaketView abgelegt wurde und agiert damit (siehe 1.2.3.).
3. *saveTemplate()* - **VorlagenView**: Templates werden in einer Liste in der VorlagenView gespeichert und beim Startvorgang aus einer Konfigurationsdatei eingelesen
4. Inhalt der Textboxen (Name, Gewicht, ...) ist an die jeweiligen Properties des aktuellen PaketViews gebindet (*Vorschaubereich oben in Abb. 15*). Die PaketView wird beim öffnen des Menüs automatisch erstellt (mit Standardwerten).

2.4. Vorlagen Modus

Nachfolgend (Abb. 16) ist der “Vorlagen Modus” und die vorhandenen Methodenverknüpfungen dargestellt.

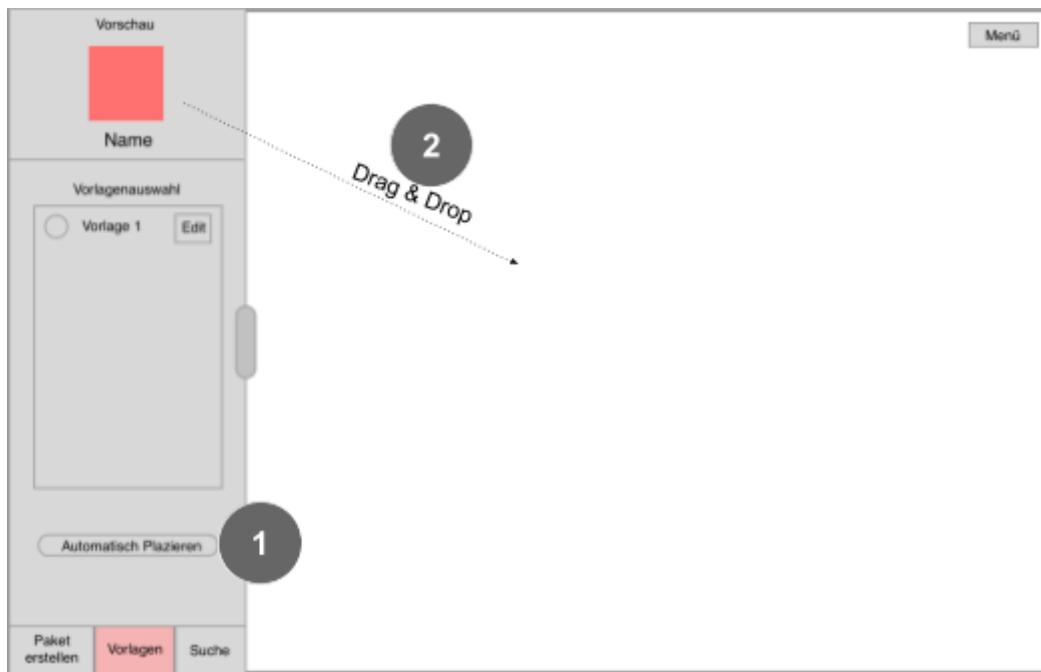


Abbildung 16: Funktionsverknüpfung: Regal Konfigurationsmodus

1. `autoPlacePaket()` - **LagerView**: (siehe 2.3. “Paket erstellen”)
2. `addPaketView()` - **BrettView**: (siehe 2.3. “Paket erstellen”)

2.5. Suchen Modus

Nachfolgend (Abb. 17) ist der “Suchen Modus” und die vorhandenen Methodenverknüpfungen dargestellt.



Abbildung 17: Funktionsverknüpfung: Regal Konfigurationsmodus

1. *toggleAusgewählt()* - **PaketView**: Bei Klick: Listener ruft Methode auf, welche eine BooleanProperty "ausgewählt" auf True setzt. Die Such-Sidebar hat einen Listener auf diese Property (Von der natürlich nur eine True sein darf!) und zeigt das aktuelle Paket an.
2. *removePaket()* - **BrettView**: das ausgewählte Paket wird entfernt
3. wenn *isDragged* - **PaketView** aktiv ist, wird *removePaket()* - **BrettView** aufgerufen. Danach: *addPaketView()* - **BrettView**: (siehe 2.3. "Paket Erstellen Modus")

3. Querschnittliche Konzepte

3.1. Rastersystem

Um die Platzierung von Regalen und Paketen in selbigen wird ein Rastersystem genutzt. Einerseits hat das "Lager"-Datenobjekt ein Raster (bestehend aus Spalten, *siehe Abb. 18*), in denen die Regale platziert werden. Ein Regal besteht immer aus zwei Pfosten, die in dem Raster platziert werden können. Um Regale zu verbinden können nebeneinander stehende Pfosten ineinander gestellt werden.

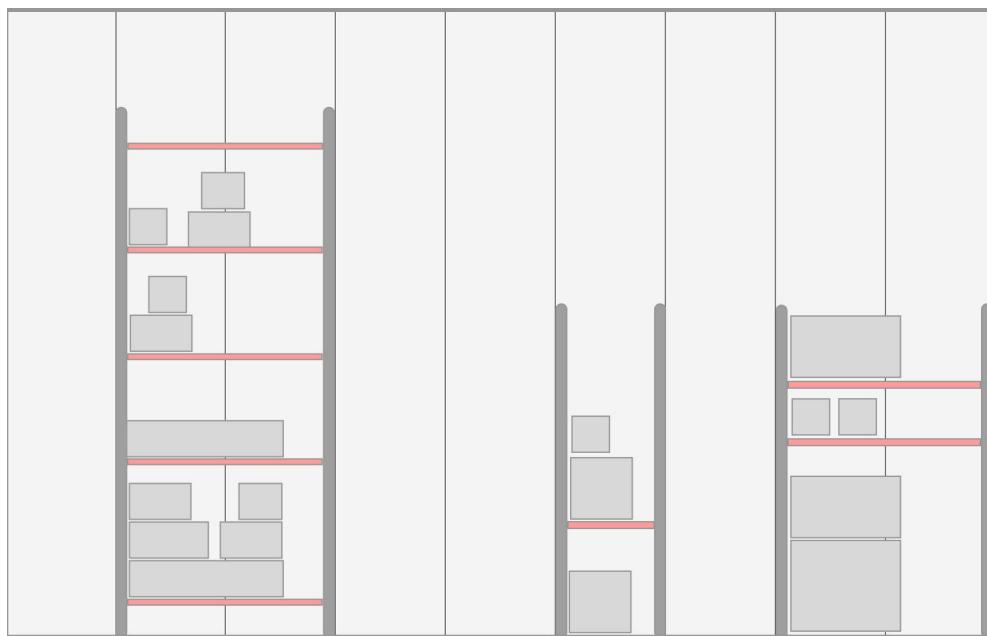


Abbildung 18: Rastersystem Regale

Die Regalbretter haben ein weiteres Raster, um die Platzierung der Pakete logisch abzubilden (*Abb. 19*). Hierbei sind ebenfalls die Spalten des Rasters besonders wichtig, da diese die Platzierung der Pakete bestimmen.

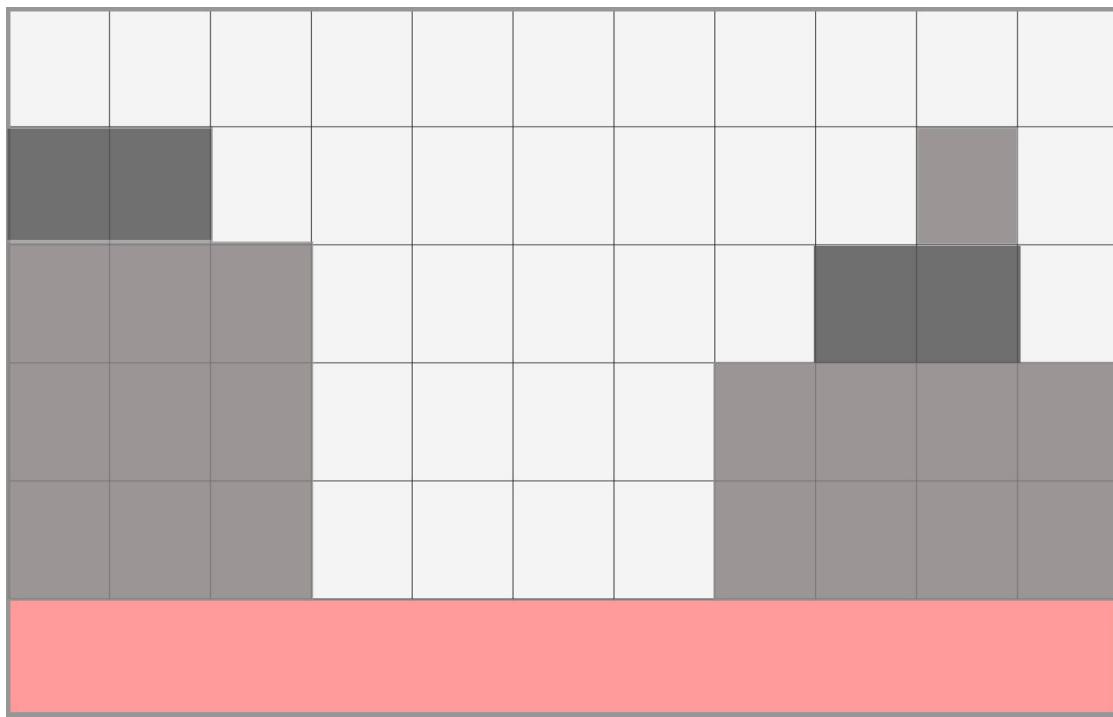


Abbildung 19: Rastersystem Regalbrett, das Brett ist hier in rot dargestellt

Die Größe der Regale muss somit ein Vielfaches der Spaltenbreite des Lager-Rasters sein. Ebenso muss die Größe der Pakete ein Vielfaches der Spalten- und Zeilenbreite/-höhe sein.

4. Qualitätsanforderungen

Die aktuellen Qualitäts- und Betriebssystemanforderungen sind in der Spezifikation unter "5. Nicht funktionale Anforderungen" zu finden.

5. Glossar

Das aktuelle, alphabetisch sortierte Glossar dieses Projekts ist in der aktuellen Spezifikation unter "6. Glossar" zu finden.

6. Abbildungsverzeichnis

Abbildung 1: Paketdiagramm, UML-Diagramm	3
Abbildung 2: View: Detailsicht, UML-Diagramm	4
Abbildung 3: Logik: Detailsicht, UML-Diagramm	5
Abbildung 4: Persistenz: Detailsicht, UML-Diagramm	5
Abbildung 5: Laufzeitsicht: Startvorgang, UML-Diagramm	6
Abbildung 6: Pseudocode: Startvorgang	7
Abbildung 7: Laufzeitsicht: Einlesevorgang, UML-Diagramm	7
Abbildung 8: Laufzeitsicht: Paket einfügen, UML-Diagramm	8
Abbildung 9: Laufzeitsicht: Brett einfügen, UML-Diagramm	9
Abbildung 10: Laufzeitsicht: Paket finden, UML-Diagramm	10
Abbildung 11: Laufzeitsicht: Paket verschieben, UML-Diagramm	11
Abbildung 12: "Prüfe Paket"-Algorithmus	12
Abbildung 13: Funktionsverknüpfung: MenuView	13
Abbildung 14: Funktionsverknüpfung: Regal Konfigurationsmodus	14
Abbildung 15: Funktionsverknüpfung: Paket Erstellen Modus	15
Abbildung 16: Funktionsverknüpfung: Regal Konfigurationsmodus	16
Abbildung 17: Funktionsverknüpfung: Regal Konfigurationsmodus	16
Abbildung 18: Rastersystem Regale	17
Abbildung 19: Rastersystem Regalbrett	18