Allgemeine Design-Entscheidungen

Wir haben uns dazu entschieden Datenbankobjekte von den Businessobjekten abzukapseln, da die Datenbankobjekte automatisch generiert werden. Außerdem kann so objektbezogene Businesslogik in diesen Objekten untergebracht werden.

Wir unterscheiden DAO's, POJO's und BusinessObjects. DAO's beinhalten nur Methoden für den Datenbankzugriff und liefern POJO's, die nur mit gettern und settern ausgestattet sind. Diese POJO's werden später in BusinessObjects gekapselt.

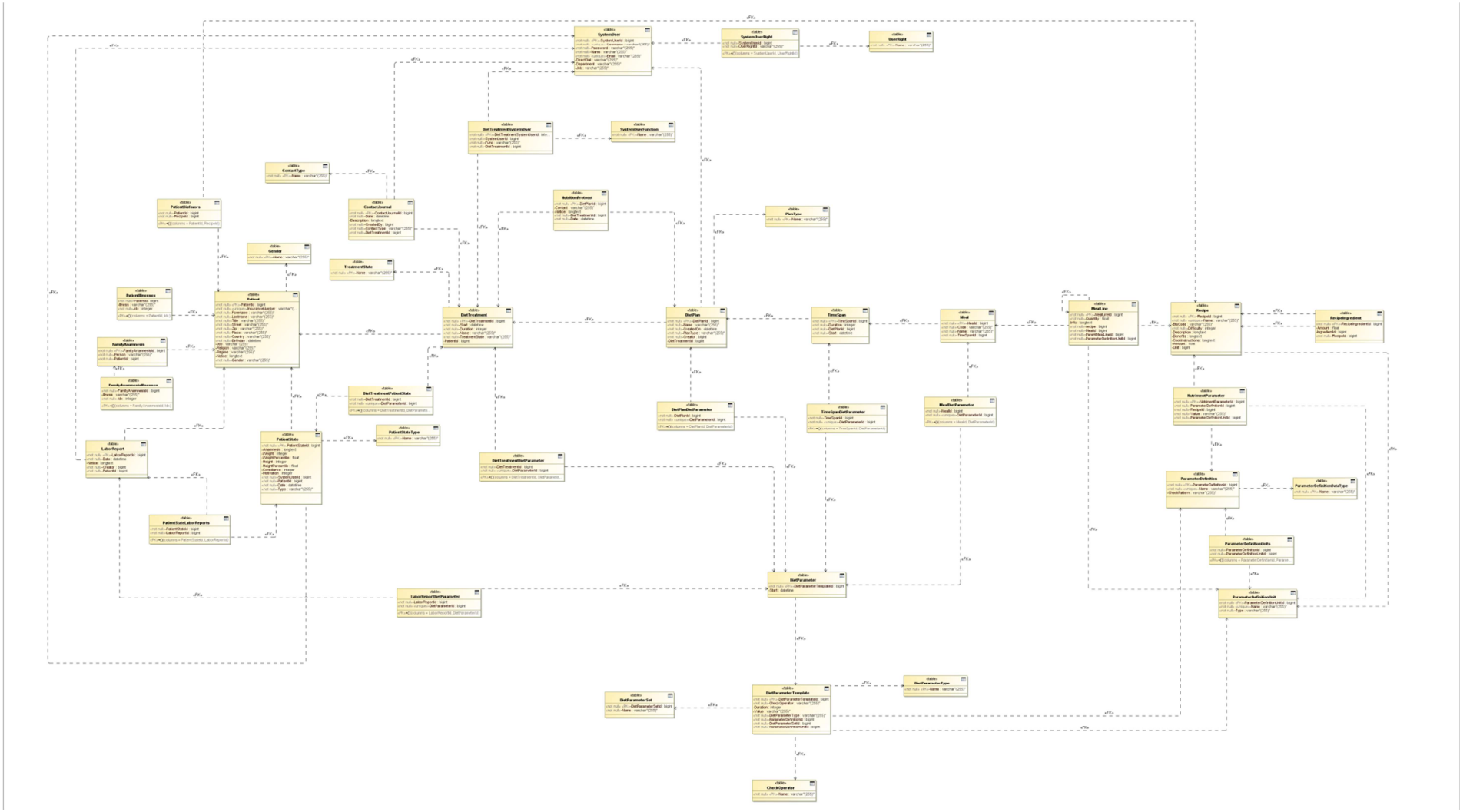
Für das GUI haben wir uns dazu entschieden die Businessobjekte mit einem Interface zu versehen, damit für das GUI nur die benötigten getter sichtbar sind.

In der GUI werden nur Apache Pivot Collections verwendet, um einerseits eine Trennung zwischen der GUI und Controller Schicht zu schaffen und andererseits um die Möglichkeit des Databindings zu schaffen. Im Databinding werden allerdings nie direkt Objekte der Applikationsschicht verändert sondern nur neue Objekte hinunter geschickt, die anschließend konvertiert werden. Der GUI Controller ist zuständig dafür, die erhaltenen Java Listen/Sets in Pivot Listen/Sets zu konvertieren.

Der BusinessLogicDelegationController ist dafür zuständig, Anfragen vom GUIController an den richtigen UseCase Controller weiterzuleitet, der alle weiteren Arbeiten übernimmt.

Alle Controller sind Singletons, dass wir von überall auf dieselbe Instanz zugreifen können.

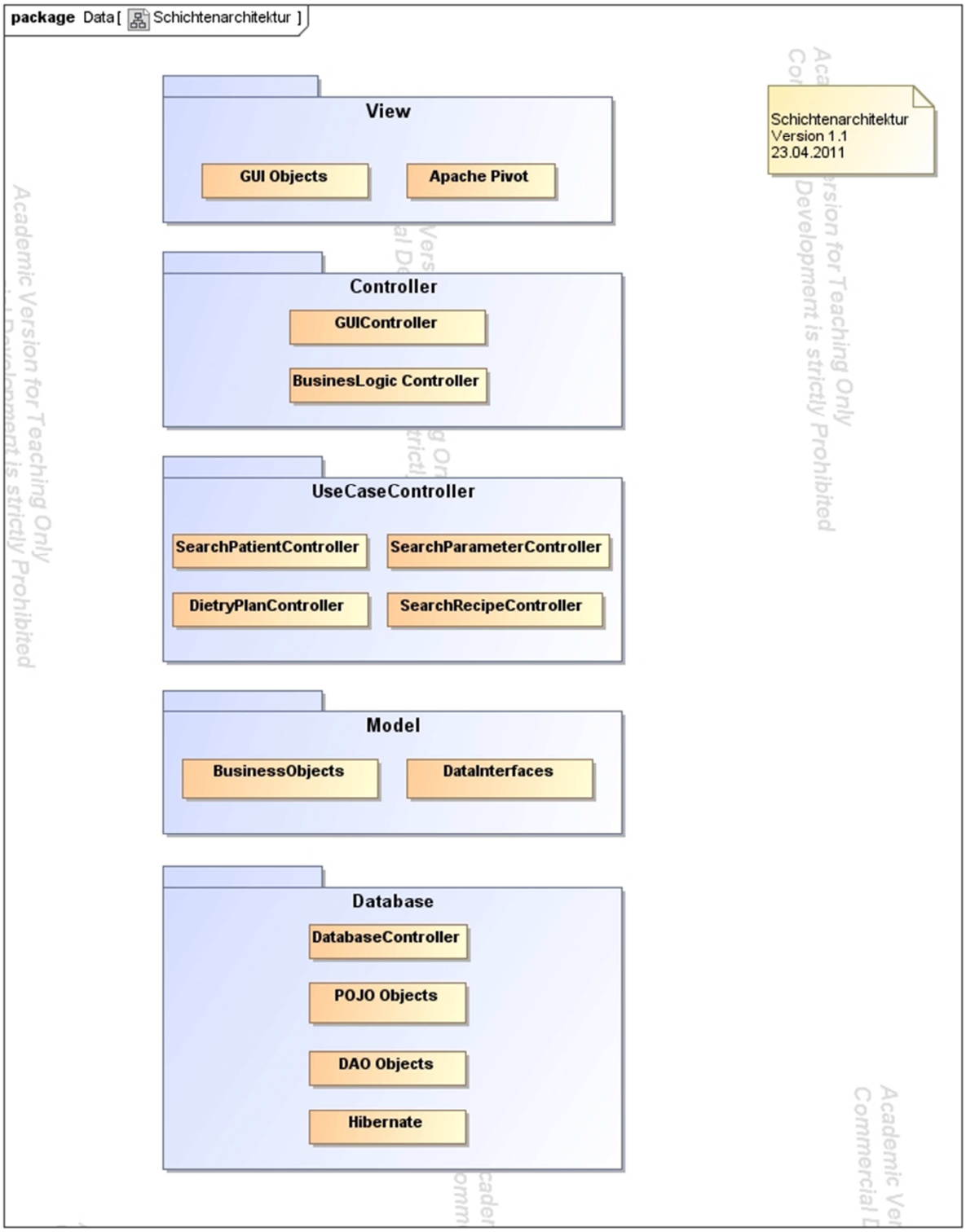
Datenbankmodell



Klassendiagramm

(siehe Anhang Klassendiagramm.jpg)

Schichtendiagramm



Apache Pivot (http://pivot.apache.org/)

Wir haben und dazu entschieden das Open Source GUI Framework Apache Pivot zu verwenden. Pivot ermöglicht es, plattformübergreifende Java GUI's über XML Dokumente zu erstellen. Wir haben uns für Apache Pivot entschieden, da dieses Framework eine Trennung nach dem MVC Prinzip anbietet und so einfach zu verwenden und zu erweitern ist.

Hibernate (http://www.hibernate.org/)

Hibernate ermöglicht es uns auf sehr einfache Weise unsere Objekte zu speichern. Des Weiteren bietet es die Möglichkeit unabhängig von der darunterliegenden Datenbank zu programmieren, und diese auch sehr einfach auszutauschen. Ein weiterer Vorteil ist, dass wir bereits Objekte geliefert bekommen und diese sofort weiterverarbeiten können.

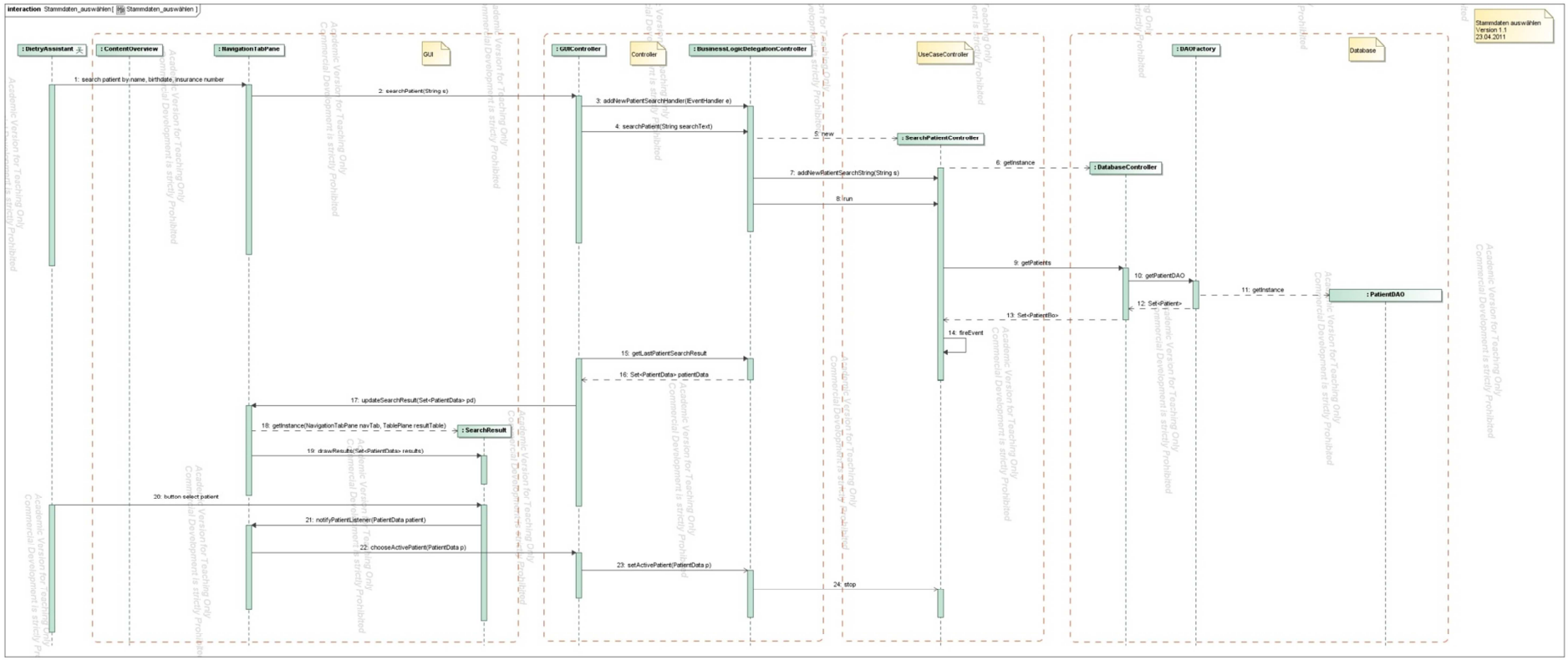
UseCase 1: Patienten suchen)

In unserer Patientensuche ist es möglich nach Vorname, Nachname, SVN und Geburtsdatum (Format: dd.mm.yyyy) zu suchen. Alles wird in einen einzigen String eingegeben der von unserer Logik interpretiert wird und die dazugehörigen Patienten liefert.

Damit das GUI während der Suche nicht blockiert, implementiert der PatientSearchController das Runnable Interface und der BussinessLogicalDelegationController startet einen neuen Thread, welcher laufend nach Patienten sucht. Das GUI wird über Events informiert, wenn der PatientSearchController neue Patienten gefunden hat und aktualisiert entsprechend.

Jede Änderung in dem Textfeld wird im PatientSearchController auf einen Stack gelegt, damit dieser immer die neuste Eingabe analysiert.

Sequenzdiagramm



UseCase 2: Diätplan erstellen)

Wir haben uns dazu entschieden, dass jeder Tag einer Diät genau auf eine Zeitspanne abgebildet wird. Dies ermöglicht uns die Parameter Tag genau zu berechnen, was aber nicht ausschließt dass Berechnungen über einen längeren Zeitraum als ein Tag möglich sind.

Wie haben uns dazu entschieden, dass das GUI nicht 0 indiziert arbeitet wie die unteren Schichten sondern 1 indiziert. Dies erleichtert die Darstellung und die Umrechnung erfolgt nur in den tieferen Schichten.

Der Diätplan wird sobald er leer mit seinen Parametern erstellt wurde einmal gespeichert, so dass ein Programmabsturz nicht zu einem Datenverlust führt. Zusätzlich wird auch nach jeder hinzugefügten Mahlzeit gespeichert.

Bei der Überprüfung der Parameter für alternative Mahlzeiten werden alle Alternativen zusammengerechnet und durch die Anzahl der Alternativen dividiert um einen Mittelwert zu erhalten.

Da es in der Datenbank aktuell keine Möglichkeit gibt, Mahlzeit Codes separat zu speichern (Frühstück, Mittagessen, etc.) haben wir uns dazu entschieden, eine statische Klasse mit diesen Mahlzeit Codes zu erstellen.

Sequenzdiagramm

(siehe Anhang Diätplan_erstellen.jpg)