1. Innere Architektur

1.1 Überlegungen zur allgemeinen Architektur

1.1.1 Events

Die Architektur der Gestenerkennungssoftware legt den Schwerpunkt auf eventbasierte Programmierung. Dies hat den Vorteil, dass auf Schleifen und Threading verzichtet werden kann. Das verringert die Komplexität der Anwendung. Zur Statussynchronisation können Event-Argument-Objekte verwendet. Dies fördert die Wart- und Anpassbarkeit des Codes.

1.1.2 Berechnungen

Berechnungen an den 3D-Skeletten sind in eigene Klassen gekapselt, was Korrekturen vereinfacht und Duplicated Code verhindert.

1.1.3 Gestenerkennung

Gesten werden vom Gesture-Checker erkannt. Jener überpfüft, ob die Reihenfolge der zu erfüllenden Gestenteile stimmt. Falls die Erkennung erfolreich war, triggert er die vom API-Benutzer für die erfolgreiche Erkennung hinterlegte Funktion.

1.2 Lösungsansätz für aufgetretene Probleme

1.2.1 Zuweisungsalgorithmus für neue und bestehende Personen

Die Unterscheidung von mehreren Personen hat Schwierigkeiten bereitet. Die Kinect wechselt die Nummerierung der erkannten Skelette ohne erkennbares System. Deshalb muss die Zuweisung der erkannten Skelette an neue oder bestehende Personen per Software erfolgen. Dies erfordert ein Matching zwischen den bisherigen Skeletten der existierenden Personen und den Skeletten die wir jeweils neu von der Kinect bekommen.

Die Ähnlichkeit eines Skelettes zu einem anderen wird lediglich anhand des Skelettgliedes „Hüfte“ bewertet. Diese Bewertung ergibt eine 2D-Matrix mit den Abweichungen als Einträge. Das Problem war nun die Auswertung dieser Ähnlichkeiten. Wie kann man am besten auswerten, welches Skelett welcher Person zuzuweisen ist? Eine Idee war, jeweils das Minimum in der Matrix zu suchen, die Zuweisung zu machen und sowohl Skelett als auch Person aus der Match-Matrix zu löschen. Das ist aber eventuell im Durchschnitt nicht die beste Zuweisung.

Wir entschieden uns vorerst für eine naive Lösung mit Listen, die jedoch gut zu funktionieren scheint:

Es werden drei Fälle unterschieden:

1. Es hat mehr Skelette als schon bestehende Personen, d.h. es kam eine Person ins Bild, sie muss neu erstellt werden. Zudem müssen ihr die benötigten Events registriert werden.
2. Es hat mehr bestehende Personen als neue Skelette, d.h. es ging eine Person aus dem Bild. Sie muss gelöscht/vergessen werden (Sie bleibt in einem dafür vorgesehenen Cache).
3. Es hat gleich viele Personen und Skelette, d.h. Zuweisung muss neu gemacht werden, sonst nichts.

Die Zuweisung erfolgt nun einfach mittels zwei temporärer Listen, aus welchen die gematchten Elemente gelöscht werden. Was übrig bleibt muss nach den drei Fällen (s.o.) beurteilt werden.

1.2.2 Zeitmessung in der .NET-Umgebung

… Ticks, millis, etc. … TODO

1.2.3 Event-Triggers aus Subklassen

Events, aus Subklassen können nicht direkt aufgerufen werden, sonden müssen in der Subklasse von einer protected Funktion gekapselt werden –in folgendem Stil: *fireSubclassEvent()*

1.2.4 GestureChecker-Statemachine: Unterscheidung zwischen triggered und success

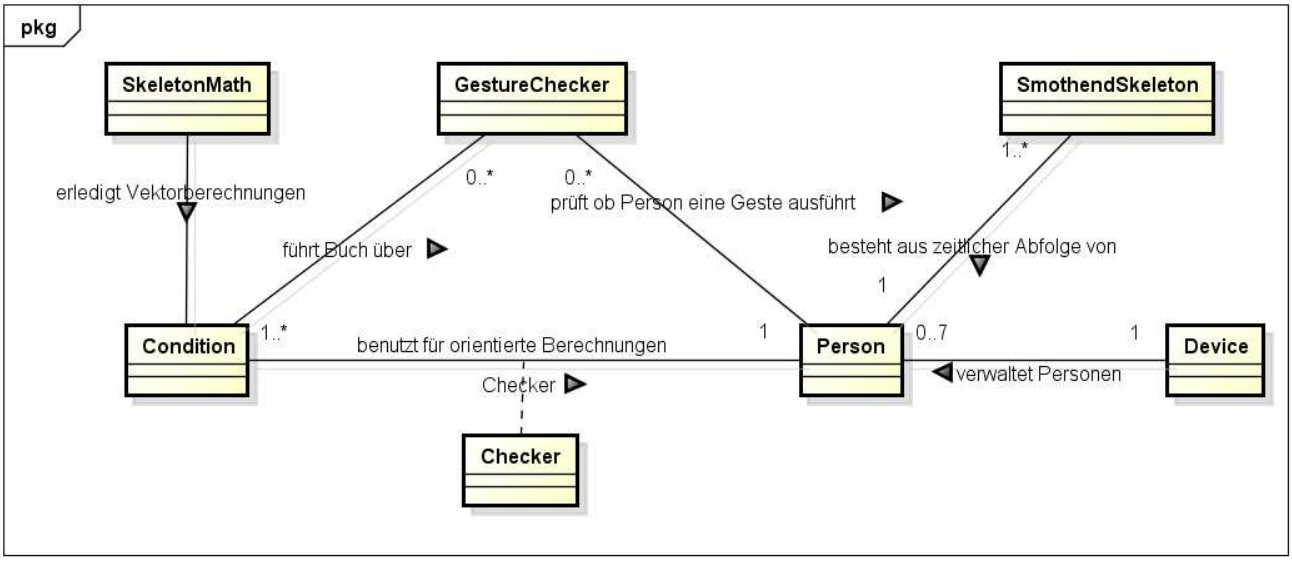
Lösung: DynamicCondition erbt von Condition

1.2.5 …

1.3 Domain

Die folgende Domainanalyse ist stark vereinfacht und soll einen groben Überblick über die Gestenerkennungssoftware ermöglichen.

Objekt	Beschreibung
Checker	Der Checker benutzt die von der *Person* abgespeicherten Skelettdaten um verschiedene zeitabhängige Berechnungen durchführen zu können, z.B: absolute Geschwindigkeit, relative Geschwindigkeit, etc.
Condition	Eine *Condition* ist der eigentliche Gestenteil. Sie kann mit *check* auf Gültigkeit überprüft werden. Zudem beinhaltet sie die Events *OnSuccess* und *OnFail*.
Device	Das *Device* kapselt das physikalische Gerät. Es gibt *Personen* per Event zurück wenn sie aktiv werden und meldet neue *Skelette*.
GestureChecker	Der *GestureChecker* führt Buch über eine komplette Geste. Er speichert wie weit fortgeschritten die Erkennung einer Geste ist und feuert Events bei der erfolgreichen Beendigung oder beim Abbruch.
SkeletonMath	Klasse für Vektorberechnungen auf Skelettdaten.
SmothendSkeleton	Geglättete *Skelette* sind die Datenquelle für alle Berechnungen und damit für die Gestenerkennung. Sie werden in der *Person* gespeichert und vom *Checker* verarbeitet.



1.4 Beispielsequenz einer Geste

Der Ablauf einer Gestenerkennung lässt sich in drei Phasen aufteilen:

1. Initialisierung:

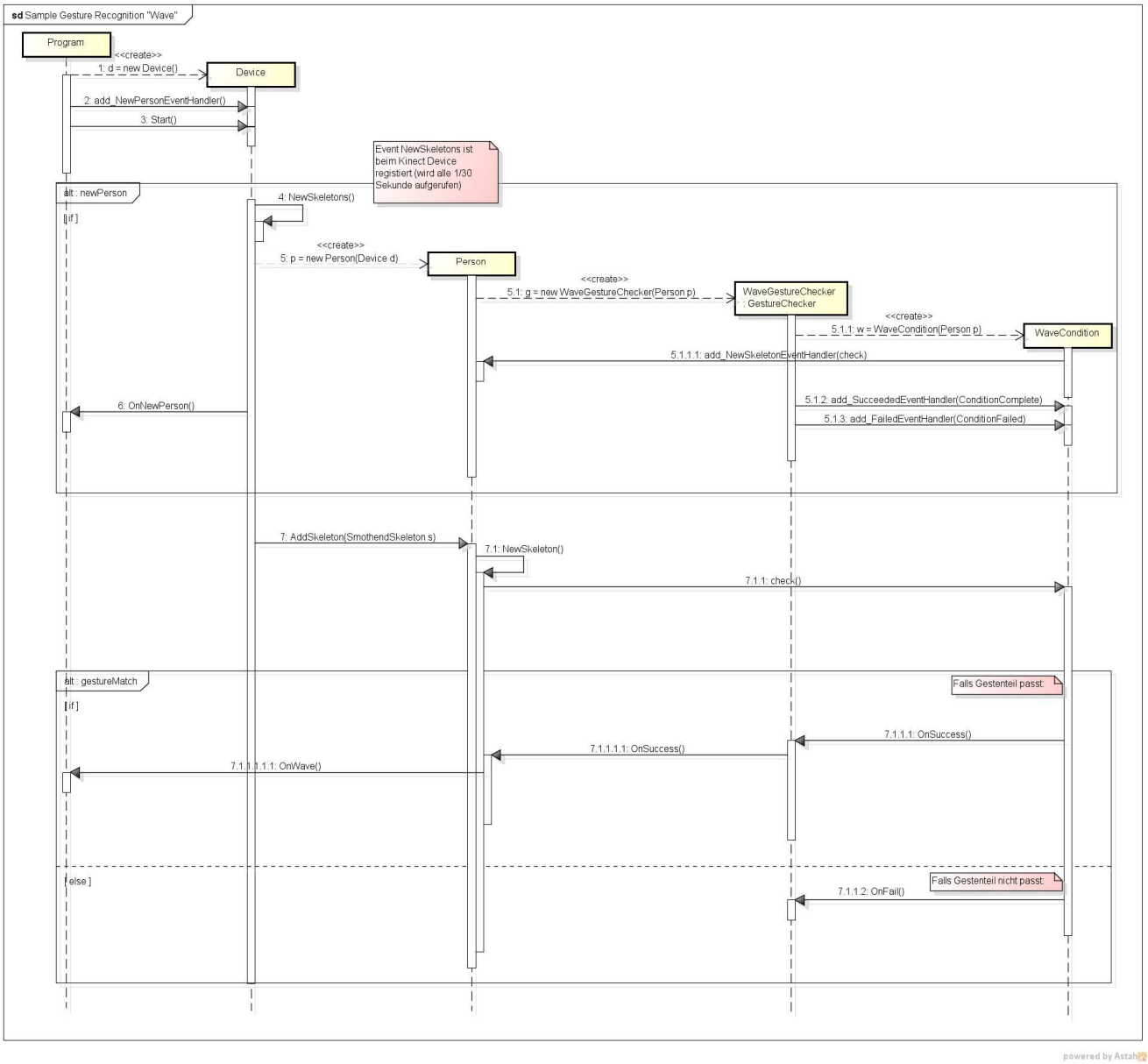
   In jedem Framezyklus werden alle *Personen* vom *Device* eindeutig identifiziert. Die identifizierten *Personen* bekommen vom *Device* ihr aktuelles *Skelett* zugewiesen. Jede *Person* erhält so mit der Zeit einen Skelettcache von 10 zeitlich geordneten *Skeletten*, welche für dynamische Berechnungen verwendet werden können.

2. Prüfen:

   Nach der Zuweisung vom neuen *Skelett* wird von den der *Person* zugeordneten *GestureCheckern* geprüft ob das aktuelle *Skelett* einen gültigen Kontext für die registrierten *Conditions* bildet.

3. Feedback:

   Der GestureChecker führt Buch über die Reihenfolge der Gestenteile (Conditions). Fall eine Geste komplett fertig durchlaufen wird, ist sie erfolgreich und wird mit einem von der User-API registrierten Event quittiert.



2. Äussere Architektur (API)

2.1 Allgemeine Überlegungen

2.1.1 Multi-Layer

Die User-API wird in verschiedenen Layern aufgebaut. Der Benutzer des Gestenerkennungsframework kann entscheiden, welchen Layer und damit auch welche Komplexität er benutzen will. Das hohe Layer bietet einen eingeschränkten Funktionsumfang, den man sehr einfach einbinden kann. Das tiefe Layer bietet Möglichkeit eigene Gesten zu definieren oder auf Low-Level Eigenschaften zuzugreifen. Beide Layer lassen sich kombiniert einsetzen. Beispielsweise muss sich der Benutzer der API nicht um die Aktivierung der Personen kümmern, kann aber dennoch eigene Gesten definieren – oder umgekehrt.

2.2 Schnittstellendefinition – Hoher Layer

Klasse	Beschreibung
Device	Von einer *Device*-Instanz bekommt man alle neuen *Personen* indem man sich beim *NewPerson*-Event registriert. Nach dem Aufruf von *Start()* initialisiert das *Device* die Kinect und beginnt mit der Erkennung von *Personen*. Die bei *NewPerson* registrierten Funktionen werden jetzt mit dem *NewPersonEventArgs*-Parameter aufgerufen. Jener enthält jeweils eine neue Person. Der *PersonActive*-Event wird gefeuert, wenn sich eine *Person* einloggt. Im *ActivePersonEventArgs*-Parameter wird diese *Person* mitgegeben.
Person	Bei einer *Person*-Instanz können Gestenreaktionen registriert werden. Momentant sind dies die folgenden: - *OnZoom*: übermittelt den Zoomfaktor - *OnSwipe*: signalisiert eine Wisch-Geste - *OnWave*: signalisiert Winken

2.3 Schnittstellendefinition – Tiefer Layer

…