ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption eines generischen Datenmodells zur Erfassung physischer Verletzungen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Maximilian Grabscheid
maximilian.grabscheid@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Dipl.-Inf. Marc Schickler

2016

Fassung 20. April 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

In einer multikulturellen Zeit wie der unseren, in der Menschen leben und arbeiten können wo sie wollen, trifft man zwangsweise früher oder später auf Sprachbarrieren. Schwierig wird dies insbesondere, wenn eine der beiden Gesprächsparteien des Englischen oder einer ähnlich verbreiteten Sprache nicht mächtig ist. Stellt man dies nun in einen medizinischen Kontext, in welchem ein Mensch Hilfe benötigt, diese aber nicht bekommt, weil er sich nicht verständlich machen kann, kann dies zu ernsthaften Problemen führen. Da diese Problematik durch steigende Einwanderer- und Flüchtlingszahlen ständig wächst, muss hierfür nun eine Lösung geschaffen werden.

Diese Abschlussarbeit bemüht sich um einen Lösungsansatz, um eben jene sprachlichen Probleme aus dem Weg zu schaffen. Dabei soll ein generisches Datenmodell zur Abbildung eines menschlichen Körpers konzipiert und umgesetzt werden. Die Umsetzung dessen resultiert in einer Anwendung, welche auf rein grafischer/symbolischer Form basiert. Diese Anwendung soll Menschen die Möglichkeit bietet sich in einem medizinischen Umfeld verständlich zu machen.

Danksagung

Zuerst möchte ich meinem Betreuer Marc Schickler für seine Unterstützung und Zeit danken, die er aufgewendet hat um Fragen zu beantworten und mich zu beraten. Außerdem möchte ich Jamena Mayer für das Probelesen und Korrigieren meiner Abschlussarbeit danken. Zuletzt möchte ich noch meiner Familie, meinen Freunden und Kommilitonen danken, die mich bei der Bewältigung meiner Bachelorarbeit unterstützt haben.

Inhaltsverzeichnis

1

Einleitung

In der heutigen multikulturellen Gesellschaft gibt es immer wieder Probleme mit der verbalen Verständigung. Dies rührt teils daher, dass Personen sich zwischen Ländern frei bewegen können, allerdings nicht zwangsweise der Sprache dieses Landes mächtig sind. Zwar sind Fremdsprachen wie zum Beispiel Englisch, Französisch oder Spanisch weit verbreitet, dennoch gibt es immer wieder Begriffe, die nicht jeder kennt. Dazu gehören insbesondere auch Vokabeln, zum Beispiel aus der Medizin, die jedoch bei einer Aufnahme in einem Krankenhaus oder einer Arztpraxis zwingend notwendig sind. Zur Zeit werden vor allem in Deutschland sehr viele Flüchtlinge aufgenommen, die auf einer langen Flucht mit hoher Wahrscheinlichkeit Verletzungen davongetragen haben. Dazu zählen Läsionen wie zum Beispiel Frakturen und Schnittwunden. Diese Menschen sind jedoch im Normalfall der deutschen Sprache nicht mächtig. Zudem sprechen viele auch nur ein sehr schlechtes Englisch, wodurch sich die Kommunikation im Allgemeinen als eher schwierig gestaltet. Dies kann dazu führen, dass sich medizinische Vorgänge unnötig in die Länge ziehen und somit medizinisches Personal und Patienten unnötig belasten. Schlimmsten falls kann aber auch die falsche oder gar keine Behandlung veranlasst werden. Dadurch können Patienten in ernsthafte Notlagen gebracht werden. Allerdings können allgemein verständliche Symbole entwickelt werden. Diese stellen dann zum Beispiel physische Verletzungen in einer Art und Weise dar, die den meisten Menschen verständlich ist. Wenn diese nun in Verbindung mit einer Computeranwendung verwendet werden würden, könnten Verletzungen so einfach und verständlich erfasst werden. Diese Anwendung könnte es zum Beispiel einem Patienten ermöglichen einzugeben, wo er welche Schmerzen oder Beschwerden hat. Außerdem könnte dadurch Personal in überfüllten Aufnahmestellen durch eine computergestützte Erfassung

von Verletzungen signifikant entlastet werden. Auf diese Weise könnte so eine Lösung für dieses Problem geschaffen werden.

1.1 Zielsetzung

Diese Abschlussarbeit beschäftigt sich mit einem möglichen Lösungsansatz für Verständigungsprobleme in einem medizinischem Kontext. Dazu soll ein generisches Datenmodell entwickelt werden, welches einen menschlichen Körper abbildet. Weiterhin soll es möglich sein einem bestimmten Bereich auf dem Körper eine Verletzung zuweisen zu können. Anschließend soll eine Anwendung implementiert werden, welches auf Basis des Datenmodells eine Möglichkeit zur Angabe verletzter Körperteile bietet. Dabei soll wo es geht auf typografische Elemente auf der grafischen Benutzeroberfläche so weit wie möglich verzichtet werden.
Am Ende soll es möglich sein mit dem erstellten Datenmodell einen Körper abzubilden und diesen in der implementierten Anwendung mit einer grafische Repräsentation eines menschlichen Körpers zu verbinden. Auf diesem Körpermodell sollen vom Benutzer Bereiche definiert werden können, welchen dann eine Verletzung zugewiesen werden können soll. Die definierten Bereiche und zugewiesenen Verletzungen sollen dann dem Benutzer angezeigt werden.

1.2 Struktur der Arbeit

Diese Arbeit gliedert sich in sieben Kapitel. Im *Grundlagenkapitel* (siehe Kapitel 2) werden allgemeine Grundlagen zu in der Anwendung verwendeten Technologien und Methoden vermittelt. In der *Anforderungsanalyse* (siehe Kapitel 3) werden die Anforderungen an die Anwendung und das Datenmodell spezifiziert. Im Kapitel *Konzeption* (siehe Kapitel 4) werden verschiedene Ansätze zur Umsetzung einzelner Komponenten diskutiert und erklärt. Das Kapitel über *Implementierung* (siehe Kapitel 5) befasst sich mit der Erstellung der Anwendung und erläutert einige Eigenheiten derer. Im *Anforderungsabgleich* (siehe Kapitel 6) werden schließlich die Anforderungen aus der Anforderungs-

analyse mit den tatsächlich umgesetzten Anforderungen verglichen. Schlussendlich wird in der *Zusammenfassung* (siehe Kapitel 7) das Ergebnis der Arbeit zusammengefasst, ein Ausblick auf mögliche Erweiterungen gegeben und ein Fazit gezogen.

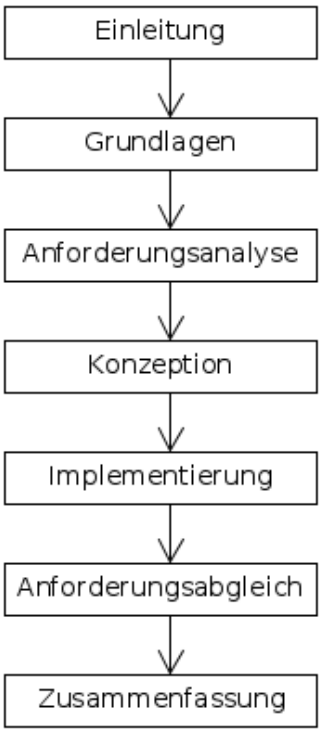

Abbildung 1.1: Gliederung der Abschlussarbeit

2

Grundlagen

In diesem Kapitel sollen unter anderem einige Begriffe eingeführt und erläutert werden (siehe Kapitel 2.1). Des Weiteren werden einige bereits bestehende Anwendungen (siehe Kapitel 2.2) vorgestellt, die eine ähnliche oder verwandte Funktion bieten. Außerdem werden einige Grundlagen zu verwendeten Technologien vermittelt werden, welche Bestandteil der zu entwickelnden Anwendung sein werden (siehe Kapitel 5). Dabei soll auch auf einige grundlegende Funktionen und Bestandteile von SVG 1.1 (siehe Kapitel 2.3) und JSON (siehe Kapitel 2.4) eingegangen werden. Auch sollen Grundzüge der Java Library Apache Batik (siehe Kapitel 2.5) erklärt und ein kurzer Exkurs zu Java's `AffineTransform` (siehe Kapitel 2.6) gegeben werden.

2.1 Begriffe

In diesem Abschnitt sollen einige Begriffe eingeführt werden, welche in der Ausarbeitung Verwendung finden. Diese beziehen sich hauptsächlich auf das Datenmodell und dienen der Differenzierung zwischen einzelnen Komponenten.

- *Level*: Mit Level ist die Ebene eines Knotens in der Baumstruktur gemeint. Level 0 stellt die Wurzel beziehungsweise den gesamten Körper dar. Level 1 entspricht zum Beispiel einem Arm oder Bein. Alle tieferen Ebenen sind vom Benutzer erstellte Körperregionen.
- *Körperteil*: Körperteile sind Elemente des Körpers wie Arme und Beine. Diese wurden nicht vom Benutzer definiert. Sie befinden sich in den oberen beiden Ebenen der Baumstruktur (Level 0 und 1; vgl. Abb. 4.1).

- *Körperregion*: Körperregionen sind die vom Benutzer definierten Bereiche im Modell und dienen der genaueren Beschreibung einer Position auf dem Körpermodell. Diese befinden sich in den Ebenen ab Level 2 der Baumstruktur und können die gleichen Klassen instantiieren wie die Körperteile (vgl. Abb. 4.1). Wenn allgemeine Gegebenheiten oder Abläufe erläutert werden, sind die Begriffe Körperteil und -region gleichgestellt.
- *Kind- und Vaterknoten*: Das Definieren neuer Körperregionen lässt eine Baumstruktur entstehen. Ein Vaterknoten hat n Kindknoten, welche als Körperregion innerhalb des Vaterknotens interpretiert werden. Der Vaterknoten kann auch als der aktuell betrachtete Knoten bezeichnet werden.
- *Interaktionsbereich*: Mit Interaktionsbereich ist der Bereich der grafischen Benutzeroberfläche gemeint, in dem das Körpermodell angezeigt wird und auf dem der Benutzer seine Eingaben macht. Dazu zählt nicht die Buttonleiste am unteren Rand.
- *Hitbox*: Die Hitbox bezeichnet den Auswahlbereich einer Körperregion. Diese wird dem Benutzer durch ein farbiges Rechteck angezeigt.

2.2 Bereits bestehende Anwendungen

Um physische Verletzungen über eine grafische Repräsentation eines Körpermodells zu erfassen, existieren bisher keinerlei Anwendungen. Lediglich einige wenige Anwendungen haben eine Funktion die, in einem weiter gefassten Rahmen, in eine ähnliche Richtung geht. Davon sollen nun zwei vorgestellt werden. Die erste Anwendung wird sich mit Verletzungen an speziellen Körperteilen (siehe Kapitel 2.2.1), die zweite mit Sprachbarrieren (siehe Kapitel 2.2.2) beschäftigen.

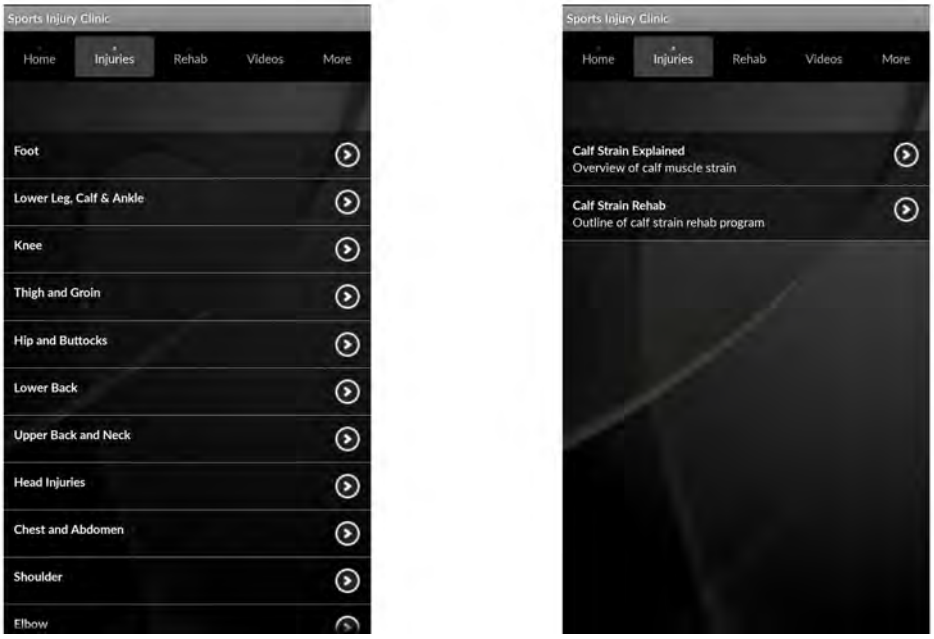

Abbildung 2.1: Screenshot Sports Injury Clinic [1]

2.2.1 Sports Injury Clinic

Die erste mobile Anwendung die Android-Applikation Sports Injury Clinic [1], welche eine textuelle Auswahl eines Körperteils anbietet. Dort lässt sich dann die genauere Region, beispielsweise der betroffene Muskel, auswählen. Wird ein Menüpunkt ausgewählt kann man diese Auswahl teilweise noch verfeinern (vgl. Abb. 2.1). Anschließend wird dann die ausgewählte Verletzung erklärt. Darüber hinaus bietet die Anwendung noch die Möglichkeit, Hintergrundinformationen über Behandlungsmethoden und die jeweiligen Verletzungen zu einzusehen. Für diesen Zweck stehen einige Videos und textuelle Beschreibungen bereit.
Die Ordnung und Kategorisierung der Körperteile und Verletzungen ist dabei aber nur begrenzt vorhanden. Beispielsweise ist die Liste weder alphabetisch noch thematisch (nach Arm, Bein etc.) geordnet. Zudem lassen sich nur manche Bereiche genauer definieren und andere gar nicht. Da die mobile Anwendung rein textuell gestaltet ist und sich hauptsächlich auf Sportverletzungen bezieht, ist sie nur entfernt mit dem Anwendungszweck verwandt.

2.2.2 SmallTalk Intensive Care

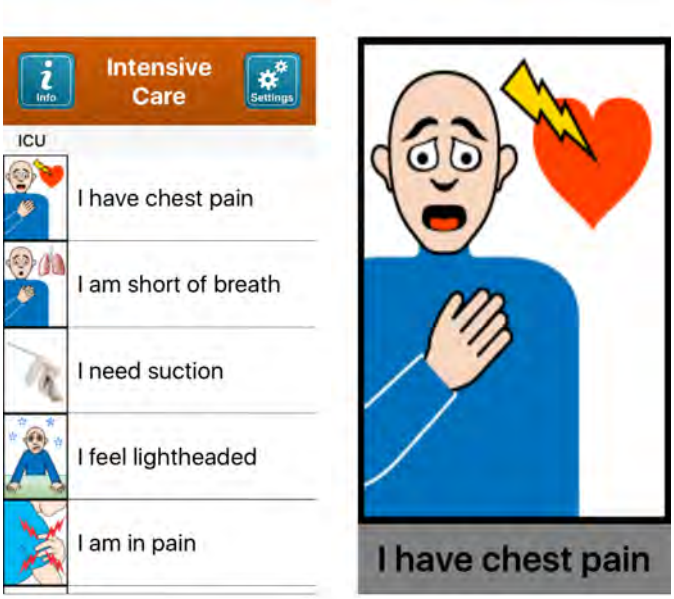

Abbildung 2.2: Screenshot SmallTalk Intensive Care [2]

Die zweite mobile Anwendung SmallTalk Intensive Care [2] ist an Kinder gerichtet, die sich nur schwer oder gar nicht verständlich machen können. In der mobile Anwendung ist eine Liste mit Icons und Beschreibungen dieser Icons vorhanden, durch welche sich bestimmte Mitteilungen auswählen lassen. Diese Mitteilungen können Bedürfnisse sein, wie zum Beispiel ”Ich muss auf die Toilette", oder aber Beschwerden, wie zum Beispiel ”Ich habe Brustschmerzen". Wird eine solche Mitteilung ausgewählt, erscheint das Icon des Menüeintrags in groß und die Beschreibung dessen (vgl. Abb. 2.2). Außerdem wird die Beschreibung des Menüpunkts vorgelesen. Da es sich bei der mobile Anwendung allerdings nur um eine Verständigungshilfe handelt und sich Verletzungen nur sehr begrenzt und allgemein ausdrücken lassen, lässt sich diese mobile Anwendung auch nicht als Referenz verwenden.

2.3 SVG 1.1

Für die zu erstellende Anwendung bietet das auf XML basierende SVG-Bildformat (`Scalable Vector Graphics`) einige sehr nützliche Eigenschaften. Da SVG auf XML basiert, werden in SVG auch dieselbe Syntax und alle XML-Konventionen verwendet [3]. Hier sollen einige Grundlagen zu SVG vermittelt werden. Unter anderem aus welchen Bestandteilen eine SVG besteht (siehe Kapitel 2.3.1). Außerdem soll auf den Begriff der Viewbox (siehe Kapitel 2.3.2) und die Möglichkeiten zur bereits vorhandenen Benutzerinteraktion mittels eingebetteter JavaScript Skripte innerhalb des Dokuments [4] (siehe Kapitel 2.3.3) eingegangen werden.

2.3.1 Zeichnen von Formen

Bei SVG handelt es sich um Bilddateien, bei denen nicht einzelne Pixel gespeichert werden, sondern die Beschreibung der Pfade, die letztendlich die Figur ergeben [3] [5]. Durch diese Eigenschaft kann die SVG beliebig groß oder klein skaliert werden ohne an Qualität (Aliasing etc.) zu verlieren. Diese Pfade werden in XML-Notation in Tags (`<g>`) angegeben. SVG bietet hierbei verschiedene vorgefertigte Formen, wie zum Beispiel Rechtecke (`<rect>`) oder Kreise (`<circle>`) an. Diese Formen benötigen lediglich Attribute, die die Höhe und Breite des Objekts angeben. Zusätzlich können auch Attribute wie Breite der Kontur und Farben angegeben werden. Wenn nun spezielle Formen erzeugt werden sollen, bietet SVG noch eine Funktion mit der sich Pfade (`<path>`) quasi per Hand zeichnen lassen [5]. Diese Pfade werden durch einzelne Punkte oder Längen zwischen zwei Punkten angegeben. Die Form der Pfade wird durch Befehle wie `m` (move to, bewegt den Cursor zum angegebenen Punkt) oder `l` (line to, zieht eine Linie zum angegebenen Punkt) und viele mehr umgesetzt [5]. Groß- und Kleinschreibung machen hier insofern einen Unterschied, dass kleingeschriebene Befehle relative und großgeschriebene Befehle absolute Pixelangaben sind. Da die manuelle Programmierung einer SVG aber sehr mühselig und unübersichtlich ist, wird

hier meist auf etablierte Programme wie zum Beispiel Inkscape oder Adobe Illustrator zurückgegriffen.

2.3.2 Viewbox

Die Viewbox [6] gibt den aktuell angezeigten Ausschnitt der SVG an. Dafür werden vier Parameter benötigt. Diese sind die Höhe und Breite der Viewbox sowie die Ursprungskoordinaten auf der SVG. Höhe und Breite müssen dabei nicht zwangsweise der der SVG entsprechen, da die Viewbox eher als Fenster zur Betrachtung der SVG angesehen werden kann. Mit einer Viewbox ist es so also möglich die SVG zu skalieren oder zu verschieben. Um die SVG zu skalieren, wird hierbei einfach Höhe und Breite der Viewbox vergrößert beziehungsweise verkleinert. Beim Verschieben hingegen werden einfach die Ursprungskoordinaten in eine beliebige Richtung verschoben. Die Viewbox der SVG wird später von der JSVGCanvas-Klasse der Apache Batik Library verwendet, um eben jene Operationen durchzuführen (siehe Kapitel 5.1.4).

2.3.3 Interaktivität

In SVG-Dokumenten ist es möglich Mausevents zu registrieren und zu verarbeiten [4]. Dazu zählen unter anderem `mouseOver` und `click`. Diese werden einfach einem `<g>`-Element als Attribut gesetzt und einer Aktion zugewiesen. Da es auch wie zum Beispiel in HTML-Dokumenten, möglich ist durch `<script>`-Tags JavaScript Skripte einzubetten, können diese Mausevents einfach überschrieben werden. Dadurch ist es möglich eigene Funktionen zu verwenden. Das Standard-Mausevent von JavaScript enthält jedoch nur einige Informationen über den Mausklick, unter anderem x- und y-Position des Klicks sowie die verwendete Maustaste. Durch ein eingebettetes Skript können allerdings beliebige Werte an das Skript übergeben und verarbeitet werden. So kann unter anderem auch ein String übergeben werden. Dieser kann dann wiederum als Bezeichner für das vom Mausevent betroffene `<g>`-Element genutzt werden. In der Funktion selbst kann so zum Beispiel ein String aus mehreren Informationen zusammengesetzt und als Alert ausgegeben werden. Dies wird später unter Verwendung der

Apache Batik Library noch wichtig sein (siehe Kapitel 5.1.3 und 2.5).

```
[...]
<g id="Head" >
        [...] onclick="buttonClicked(event, 'head')"/>
</g>
[...]
<script>
  function buttonClicked(event, location) {
    var eevent = event ? event : window.event;
      if(eevent.button != 2) {
        alert(eevent.clientX + ":"
          + eevent.clientY + ":"
          + location);
      }
  }
</script>
[...]
```

Listing 2.1: Ausschnitt aus der SVG mit dem verwendeten JavaScript Mouselistener

2.4 JSON

JSON (`JavaScript Object Notation`) ist ein leichtgewichtiges Datenaustauschformat, welches für Menschen einfach zu lesen ist. In JSON sind einige Konventionen aus gängigen Programmiersprachen wie C/C++ und Java übernommen. JSON selbst basiert teilweise auf JavaScript, genauer gesagt auf dem JavaScript Standard ECMA-262 3rd Edition von 1999 [7]. Diese Eigenschaften machen JSON zu einem sehr umgänglichen und universal verwendbaren Format zum Austausch von Daten.

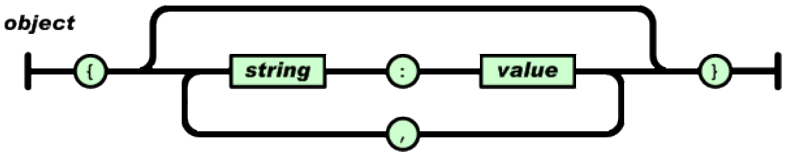

Abbildung 2.3: Syntax eines JSON-Objekts [8]

Die Struktur von JSON-Objekten [8] basiert auf `key:value`-Paaren. Der Schlüssel ist hier ein String, gekennzeichnet durch , wobei der Wert die Form eines Strings, einer Zahl, eines anderen Objekts, eines Arrays, `true/false` oder `null` annehmen kann. Ein Objekt ist hier eine Menge aus Schlüsseln (`key`) und Werten (`value`), bestehenden Wertepaaren, welche durch Kommas getrennt und durch ein Paar geschweifter Klammern zusammengefasst werden. Es ist auch möglich, dass ein `key:value`-Paar aus einem Objekt oder einem Array besteht. Ein Array [9] wird durch ein Paar eckiger Klammern gekennzeichnet und kann eine Menge von Werten zusammenfassen, welche dann keinen eigenen Schlüssel besitzen.

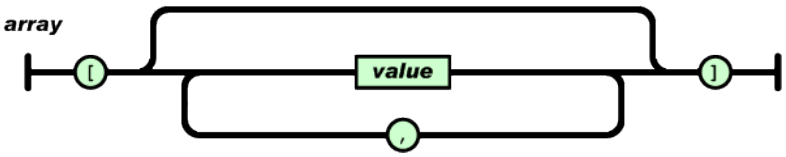

Abbildung 2.4: Syntax eines JSON-Arrays [9]

2.5 Apache Batik

Durch die Verwendung von SVGs war es nun nötig eine externe Library zu Verarbeitung dieser zu verwenden. Die Apache Batik Library [10, 11] liefert eine große Menge an Möglichkeiten SVG-Dokumente in einer Java-Anwendung zu verarbeiten. Libraries wie zum Beispiel SVGSalamander [12] und JFreeSVG [13] beschränken sich hauptsächlich auf das Anzeigen und Erstellen von SVGs. Möchte man allerdings spezielle Aspekte wie unter anderem die zuvor genannte Interaktivität (siehe Kapitel 2.3.3) von SVG nutzen muss man sich anderweitig umsehen. Hier bietet sich dann die Verwendung von einer sehr umfangreichen Library wie Apache Batik an, deren Funktionsumfang den von

anderen Libraries bei weitem übersteigt.
Ein Aspekt der Apache Batik unter anderem so nützlich macht, ist das SVGUserAgent-Interface. Dieses Interface [11, 14] ermöglicht eine Interaktion mit der SVG auf einer tieferen Ebene, zum Beispiel mit Mouselistenern innerhalb der SVG. So können Events auf der SVG registriert und mit eigenen Funktionen verarbeitet werden. Hinzukommt, dass es relativ einfach ist durch die JSVGCanvas die Viewbox der SVG zu manipulieren, wozu unter anderem das Skalieren und Verschieben dieser zählt (siehe Kapitel 2.3.2). Diese Funktionen werden mit Hilfe von Javas `AffineTransform`-Klasse realisiert (siehe Kapitel 2.6). So ist es zum Beispiel möglich, die SVG an der Position eines Klicks zu skalieren und dort zu zentrieren. Außerdem ist es möglich durch die Overlay-Klasse [15, 11] von Apache Batik grafische Elemente wie Bilder oder einfache geometrische Formen über die SVG zu zeichnen. Zudem besitzt JSVGCanvas auch so genannte Interactors. Mit diesen lassen sich Operationen wie Rotieren oder Skalieren einfach umsetzen. Diese überlagern allerdings die `showDialog()`-Operation und lassen so nicht mehr auf in der SVG implementierte Mouselistener zugreifen. Overlays und Interactors werden als Listen gespeichert, welche sich beliebig erweitern lassen.

2.6 AffineTransform

Zur Manipulation des Bildausschnitts wird von Apache Batik dabei die sogenannte `AffineTransform` Klasse von Java verwendet. Die `AffineTransform` Klasse [16] wird benutzt, um zweidimensionale Koordinaten auf andere zweidimensionale Koordinaten anhand von Matrizen abzubilden. Die Klasse besitzt eine Reihe von Operationen von denen allerdings für diesen Zweck nur zwei wirklich interessant sind:

- `scale(double sx, double sy)` [16]: Skalierung des Bildes. Hierbei wird eine Matrix-Repräsentation des Bildes mit einer 3x3-Matrix der Form

$$\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

multipliziert, wobei *sx* den Vergrößerungsfaktor der x-Koordinate und *sy* den der y-Koordinate repräsentiert. Dabei wird das Bild bei einem Wert kleiner 1 vergrößert beziehungsweise bei einem Wert größer 1 verkleinert. Jedoch wird die Viewbox nicht mit verschoben, was darin resultiert, dass die SVG beim vergrößern nach rechts unten beziehungsweise beim verkleinern nach links oben verschoben wird.

- `translate(double tx, double ty)` [16]: Verschiebt die Viewbox der SVG ebenfalls anhand einer 3x3-Matrix. Diese hat allerdings folgende Gestalt:

$$\begin{pmatrix} 0 & 0 & tx \\ 0 & 0 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

tx ist hier der Wert um welchen der Bildausschnitt in x-Richtung und *ty* der, der den Bildausschnitt in y-Richtung verschiebt. Dies wird dazu benötigt, um die Verschiebung durch `scale` auszugleichen. In diesem Fall wird die SVG dann an der Position des Mausklicks zentriert.

Beide Funktionen basieren auf der `concatenate(AffineTransform at)` Funktion, welche eine `AffineTransform this` mit einer anderen `AffineTransform at` in der Form `this = this x at` multipliziert.
Das JSVGCanvas-Widget von Apache Batik nutzt zur Berechnung der Viewbox diese Art der Repräsentation. Die `AffineTransform` wird im Attribut `RenderingTransform` der JSVGCanvas gespeichert. Um die SVG zum Beispiel zu skalieren, wird eine neue `AffineTransform` erstellt und anschließend als neue `RenderingTransform` genutzt (siehe Kapitel 5.1.4).

3

Anforderungsanalyse

In diesem Kapitel sollen die Anforderungen an die zu entwickelnde Anwendung definiert werden. Diese werden in funktionale (siehe Kapitel 3.1) und nicht-funktionale (siehe Kapitel 3.2) Anforderungen unterteilt.

3.1 Funktionale Anforderungen

Funktionale Anforderungen definieren den Funktionsumfang der Anwendung, welche dem Benutzer später zur Verfügung stehen sollen. Diese sind unterteilt in Anforderungen an die grafische Benutzeroberfläche (siehe Kapitel 3.1.1), die Persistenz (siehe Kapitel 3.1.2) und das Datenmodell (siehe Kapitel 3.1.3).

3.1.1 Grafische Benutzeroberfläche

I. *Definieren neuer Körperregionen*:
   Es soll möglich sein durch Benutzereingaben neue Regionen auf dem Körper zu erstellen.

II. *Auswählen von Verletzungen*:
   Körperregionen sollen Verletzungen hinzugefügt werden können.

III. *Navigation durch Baumstruktur*:
   Dem Benutzer soll es möglich sein, durch den Körper, d.h. durch die erstellten Körperregionen zu navigieren.

IV. *Anzeigen bereits erstellter Körperregionen*:
Körperregionen, die bereits definiert wurden, sollen dem Benutzer markiert und angezeigt werden.

V. *Speichern-Knopf*:
Der Benutzer soll die Möglichkeit haben, durch einen Speichern-Knopf den Zustand der Anwendung zu speichern.

VI. *Laden-Knopf*:
Dem Benutzer soll es möglich sein durch einen Laden-Knopf einen vorher erstellten Körper auszuwählen und diesen wiederherzustellen.

VII. *Reset-Knopf*:
Dem Benutzer soll es möglich sein durch einen Reset-Knopf den ursprünglichen Zustand der Anwendung wiederherzustellen.

3.1.2 Persistenz

I. *Speichern des aktuellen Zustands*:
Der aktuelle Zustand der Anwendung soll persistent gespeichert werden.

II. *Laden eines vorherigen Zustands*:
Eine vorherige Instanz eines Körpers soll wiederhergestellt werden können.

3.1.3 Datenmodell

I. *Generisches Erstellen von Körperregionen*:
Das Datenmodell soll es ermöglichen an einer frei wählbaren Stelle auf dem Körpermodell eine neue Region zu definieren.

II. *Spezifizierung der Position*:
Die Position einer erstellten Körperregion soll gespeichert werden.

III. *Zuweisung einer Verletzung*:
Es soll möglich sein einer definierten Region eine Verletzung zuzuweisen.

3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen definieren die Art und Weise wie die Anwendung umgesetzt werden soll und welche Eigenschaften abseits der Funktionalität erfüllt werden sollen.

I. *Technische Anforderungen*:
   Das System muss in Java 1.8 entwickelt werden und soll in einer entsprechenden Java VM lauffähig sein.

II. *Ergonomische Anforderungen*:
   Die Bedienung der Anwendung muss möglichst einfach gehalten werden.
   Außerdem soll auf typografische Elemente möglichst verzichtet werden.

III. *Anforderungen an Dienstqualität*:
   Der Benutzer soll keine zu langen Antwortzeiten der Anwendung abwarten.

IV. *Zuverlässigkeit*:
   Die Anwendung muss Fehleingaben des Benutzers abfangen und diese aussortieren können.
   Die Anwendung darf nicht durch Fehleingaben zum Absturz gebracht werden.

V. *Wart- und Erweiterbarkeit*:
   Die Anwendung muss einen hohen Grad an Erweiter- und Wartbarkeit bieten.

VI. *Verwendung einer Rule-Engine*:
   Es soll eine Rule-Engine verwendet werden.

4

Konzeption

In diesem Kapitel sollen die Konzepte für die Entwicklung des Datenmodells (siehe Kapitel 4.1) sowie des Körpermodells (siehe Kapitel 4.2) erläutert werden. Dabei wird sowohl auf verworfene als auch letztendlich übernommene Konzepte eingegangen. Besondere Beachtung erfährt hier vor allem das generische Datenmodell, die Benutzeroberfläche und die Art der Darstellung der Baumstruktur des Datenmodells.

4.1 Datenmodell

Das Datenmodell (vgl. Abb. 4.2) sollte eine möglichst generische Art der Abbildung eines menschlichen Körpers darstellen, weshalb auf einen hohen Grad der Generalisierung einzelner Körperteile Wert gelegt wurde. Um dies zu erreichen, wird eine abstrakte Klasse definiert, die alle allgemeinen Attribute eines Körperteils enthält. Darüber hinaus wurden einige Methoden implementiert, die für jedes Körperteil benötigt werden. Diese Methoden arbeiten mit der abstrakten Klasse, um diese möglichst allgemein zu halten und sie auch für jede abgeleitete Klasse nutzbar zu machen. Von dieser abstrakten Klasse sollen nun alle anderen Körperteil-Klassen erben. Für jedes Körperteil, das sowohl auf der linken als auch der rechten Seite des Körpers vorhanden ist, wird eine weitere Abstraktionsebene eingefügt, um eventuelle Eigenheiten der linken beziehungsweise rechten Seite des Körpers abbilden zu können. Beispielsweise sind linker und rechter Arm gespiegelt, was falls das Körpermodell enger mit dem Datenmodell verknüpft werden soll, eine Rolle spielen könnte. Außerdem kann so eine genauere Typisierung erfolgen, welche Überprüfungen und Zuordnungen vereinfacht. So kann durch Typisierung sowohl zwischen linken und rechtem Arm wie auch allgemein zwischen Arm

und Bein unterschieden werden. Körperteile, die einem anderen Körperteil zugeordnet werden können, wie etwa Augen und Ohren dem Kopf, werden diesem direkt als Attribut zugeordnet.

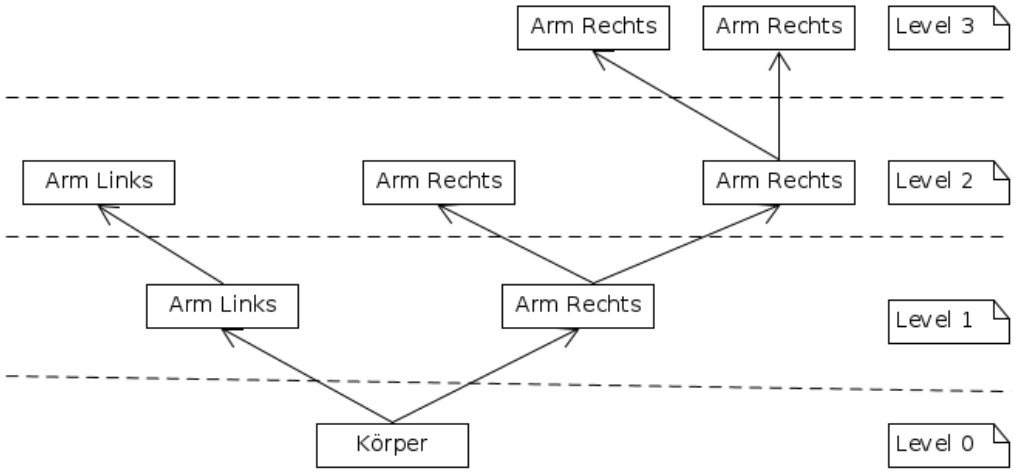

Abbildung 4.1: Skizze der Baumstruktur

Um eine genaue, stufenweise Definition von Körperregionen zu ermöglichen, muss eine Art Baumstruktur geschaffen werden. Dazu hat jedes Körperteil eine Liste an Kindknoten und einen Vaterknoten. Diese Kindknoten werden als innerhalb des Vaterknotens befindliche Subregionen des Körpers interpretiert. Die dadurch entstehende Baumstruktur lässt sich beliebig weit in die Tiefe treiben. Die Wurzel soll hierbei ein Körper bilden, welcher zwei Arme, Beine, Hände und Füße sowie Kopf und Torso besitzen soll. Wie oben bereits erwähnt sollen allerdings Finger, Zehen, Augen und Ohren den jeweiligen Händen, Füßen und dem Kopf zugeordnet werden. Kindknoten dieser Hände, Füße und dem Kopf werden beim Erstellen die zugehörigen Extremitäten des Vaterknotens übergeben. So erhält man auf jedem Levelzum Beispiel, innerhalb einer Hand, Zugriff auf die Fingerkörperteile und kann so durch diesen Unterbaum traversieren.
Da der Körper aus einzelnen Körperteilen besteht, können durch Abändern dessen auch zum Beispiel Tiere auf diese Art und Weise abgebildet werden. Dann besitzt der Körper dann statt Armen und Beinen eben nur Beine. Hierfür muss aber ein eigenes Körpermodell entworfen werden.

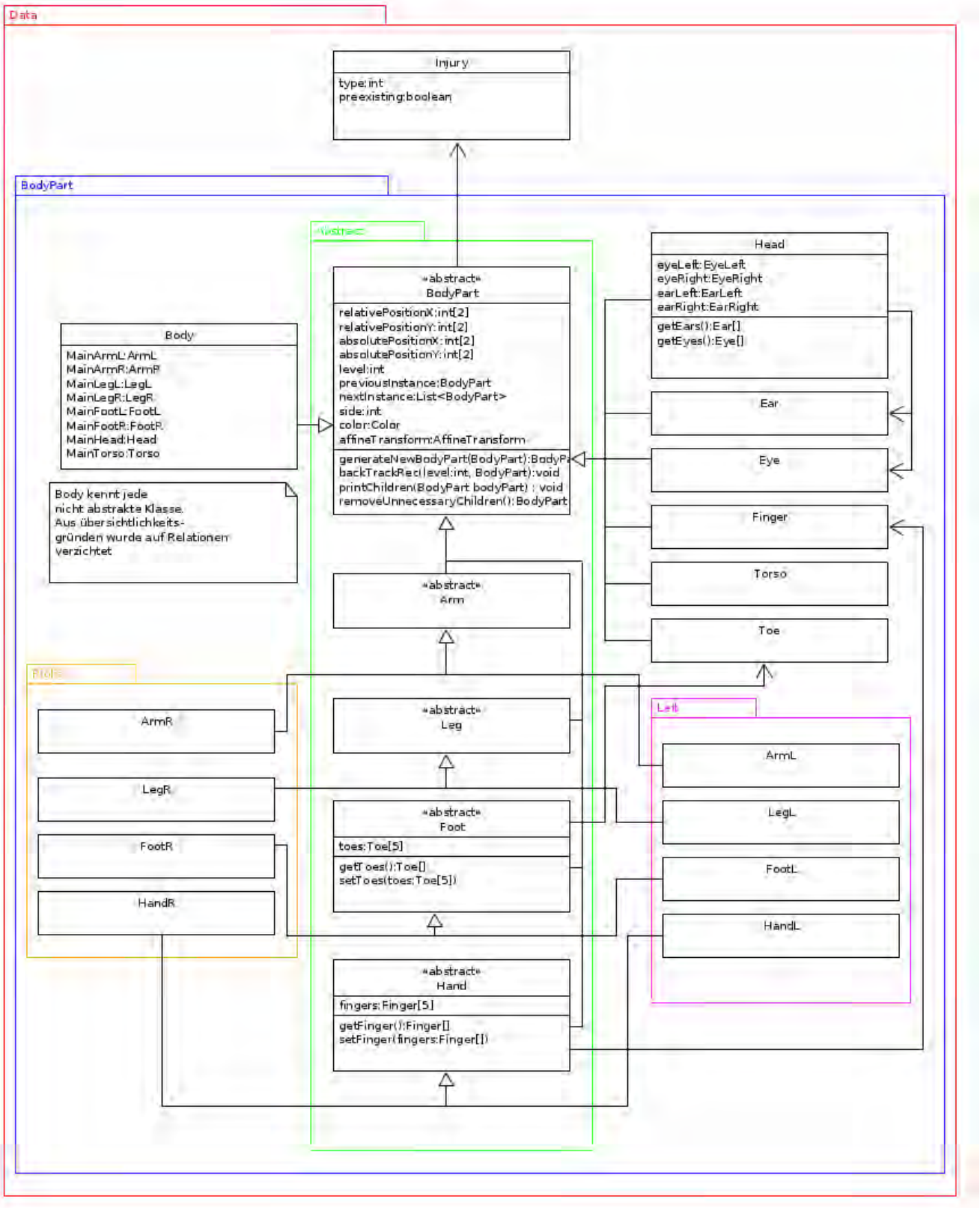

Abbildung 4.2: UML-Diagramm des Datenmodells

4.2 Körpermodell

Die Konzeption des Körpers teilt sich in einige Unterpunkte. Diese betreffen zum einen die Gestaltung des Körpermodells (4.2.1), und zum anderen Anzeigefeatures der Oberfläche. Mit Anzeigefeatures sind hier die Darstellung von Körperregionen (4.2.2) und Verletzungen (4.2.3) und wie welche Eingaben entgegengenommen und verarbeitet werden sollen (4.2.4) gemeint.

4.2.1 Gestaltung des Körpers

Da für diese Art von Anwendung noch keine Modelle entwickelt wurden, musste eine eigene Idee entwickelt und umgesetzt werden. Das Körpermodell soll möglichst allgemein und eindeutig gehalten werden. Der Grundgedanke war eine Art Puppe abzubilden, wie sie zum Beispiel Kinder bei einem Arzt bekommen könnten, um zu zeigen wo sie Schmerzen haben. Eine erste Idee war die Verwendung mehrerer einzelner Bilder die zu einem Körper zusammengesetzt werden sollten. Bei einem Klick auf das entsprechende Körperteil, zum Beispiel den linken Arm, sollte dann der Rest des Körpers ausgeblendet werden und nur noch dieses ausgewählte Körperteil vergrößert dargestellt werden. Auf Grund dessen, dass die Grafik jedoch skalierbar sein soll, um Körperabschnitte möglichst genau definieren zu können, wurde statt diesem pixelbasiertem Ansatz auf einen vektorbasierten Ansatz zurückgegriffen. Diese sind unendlich skalierbar und weisen bei hohem Skalierungsgrad keine Qualitätsverluste auf.

Dazu wurde eine SVG erstellt, die möglichst deutlich Körperteile optisch von einander trennt. So werden die einzelnen Körperteile einfacher zur treffen und eine bessere Orientierung wird ermöglicht. Um dies zu erreichen wurde ein möglichst allgemein gehaltenes Modell eines Menschen erstellt, der alle Gliedmaßen von sich streckt.

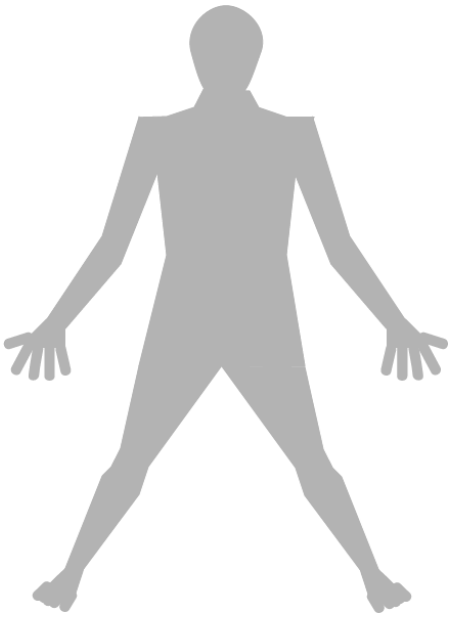

Abbildung 4.3: Abstrakte Abbildung eines menschlichen Körpers

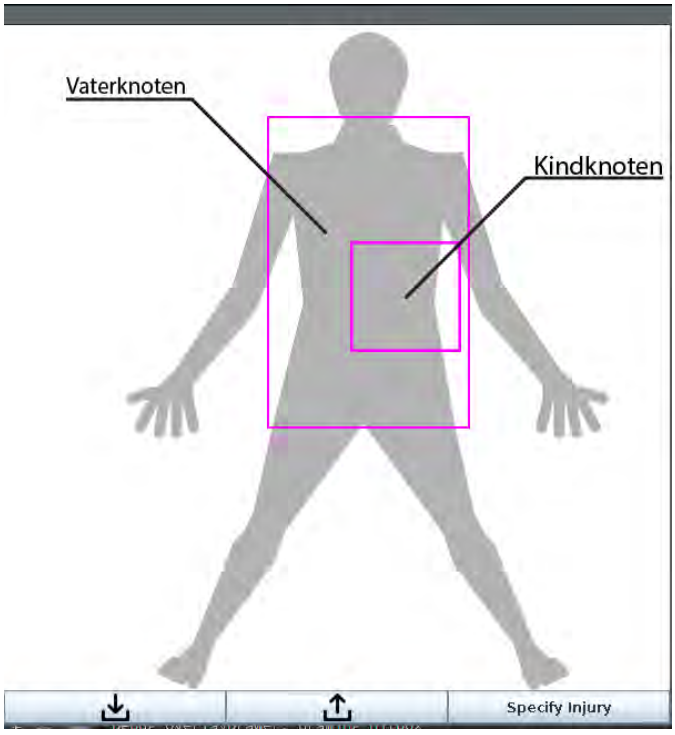

Abbildung 4.4: Erster Entwurf der Darstellung von Vater und Kindknoten

4.2.2 Darstellung definierter Regionen

Die Anzeige der definierten Körperregionen gestaltete sich als komplexer als zunächst angenommen, weshalb mehrere Anläufe mit unterschiedlichen Darstellungsmöglichkeiten entwickelt wurden. Die grundlegende Idee der Art der Darstellung der angestrebten Vater-Kind-Struktur wurde dabei jedoch beibehalten. Diese ist ein ausgewähltes Element als Vaterknoten festzulegen und dessen Kinder dann wiederum innerhalb des Vaterknotens anzuzeigen. Wird dann einer der Kindknoten ausgewählt, wird dieses zum Vater, worauf dann die Kinder des neuen Vaterknotens innerhalb dessen angezeigt werden (vgl. Abb. 4.4). Benachbarten Kindknoten soll es nicht möglich sein sich gegenseitig zu überlappen. Durch eine Überlappung kann eventuell das gewollte Element nur schwer ausgewählt werden, da durch den Auswahlalgorithmus das erste Kind, in welchem der Mausklick stattfindet, selektiert wird. Hinzukommt, dass diese Überlappungen auch eine Unübersichtlichkeit im Interaktionsbereich produzieren, die neben erschwerter Selektion auch erschwerte Differenzierung von Kindknoten verursachen.

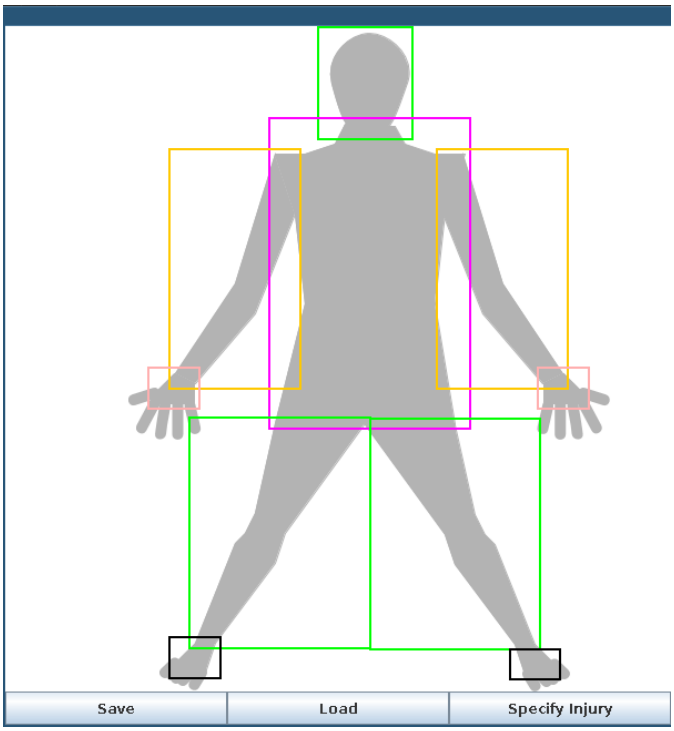

Abbildung 4.5: Alle verfügbaren Hitboxen werden angezeigt

Zuerst wurden alle verfügbaren Hitboxen als farbige Rechtecke markiert (vgl. Abb. 4.5). Der Vaterknoten wurde in diesem Entwurf als vollständig sichtbares Rechteck auf dem Körper dargestellt. Dieses Rechteck wurde so groß gezeichnet um dessen Kindknoten noch innerhalb des Rechtecks darstellen zu können. Bei genauerer Betrachtung stellte sich dieser Ansatz aber als nicht wirklich intuitiv heraus und hatte eher das Potenzial den Nutzer zu verwirren. Dabei war vor allem nicht eindeutig ersichtlich, was ein kleineres Rechteck innerhalb eines größeren Rechtecks zu bedeuten hat.

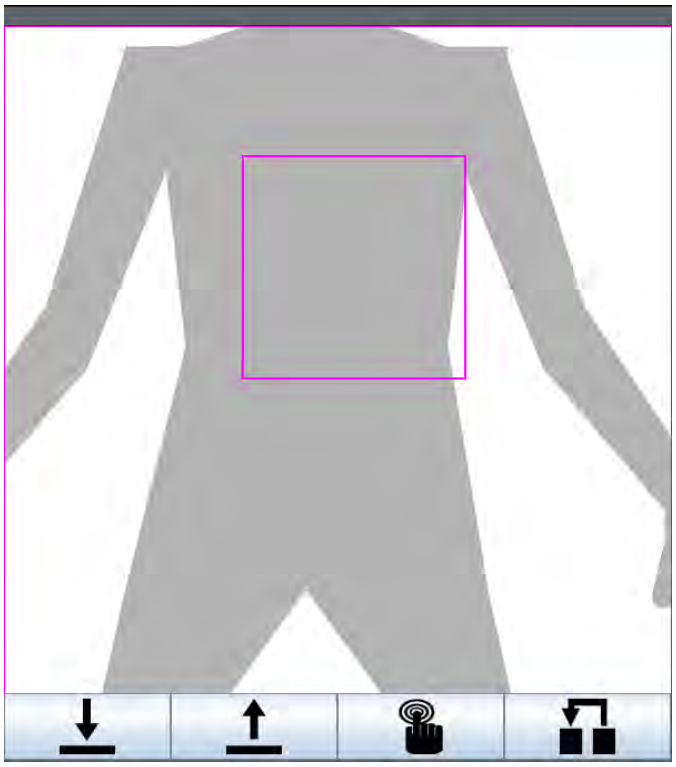

Abbildung 4.6: Zweiter Entwurf der Darstellung von Vater und Kindknoten

Als nächster Schritt wurde die definierte Körperregion soweit vergrößert, dass der gesamte Bildausschnitt als klickbarer Bereich zur Definition eines neuen Körperteils interpretiert werden konnte (vgl. Abb. 4.6). Darin werden weiterhin die Kindknoten als farbige Rechtecke angezeigt, um dem Benutzer zu zeigen, wo bereits Kindknoten definiert wurden. Die Verbesserung im Gegensatz zum ersten Entwurf besteht darin, dass der Vaterknoten nicht mehr sichtbar ist und so der Verwirrungsfaktor "kleines Rechteck in großem Rechteck” entfällt. Außerdem wurde auf Level 1, das heißt ein Körperteil

ist ausgewählt, sobald ein Kindknoten existiert, auf das Einzeichnen der Hitbox des Körperteils verzichtet. Der Grund dafür ist, dass die Kindknoten des Körperteils größer eingezeichnet werden als die Hitbox des Körperteils. Das erzeugt nur Verwirrung und sieht seltsam aus. Da die Anzahl der dargestellten Formen immer noch recht hoch war, wurden alle unnötigen Knoten entfernt. Diese waren jene Knoten in der Baumstruktur, die selbst beziehungsweise deren Kinder Verletzungen zugewiesen bekommen haben. Dadurch sind nur noch Knoten zu sehen sind, die auch eine Funktion (die Navigation zu einer Verletzung) oder Aussage (eine Verletzung) haben. Diese Verbesserungen entwirren die grafische Oberfläche etwas, wodurch die mentale Belastung des Benutzers reduziert wird.

4.2.3 Darstellung von Verletzungen

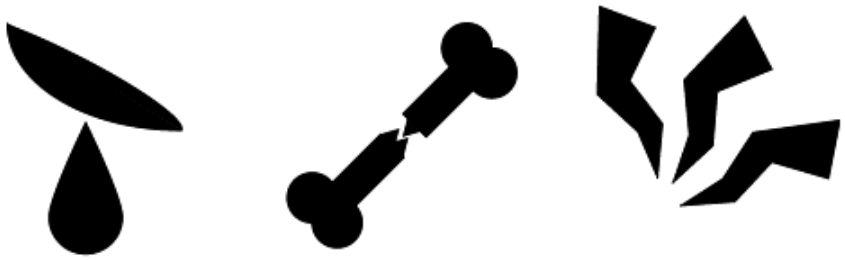

Abbildung 4.7: Von links nach rechts: Schnittwunde, Knochenbruch, Schmerz

Wenn der Benutzer einer Körperregion eine Verletzung zugewiesen hat, soll diese dem Benutzer auch angezeigt werden. Diese Verletzungen werden durch einfach gehaltene Icons dargestellt. Die Icons sollten für möglichst jedermann verständlich sein, das heißt diese sollten möglichst ohne typografische Elemente auskommen. Bestehende Symbole allerdings beinhalten meist eine Abbildung eines Körperteils, sind also nicht allgemein genug gehalten. Zum Beispiel enthält ein Symbol für Schmerz einen Arm oder einen Kopf. Nachdem der Patient einen Bereich auf dem Fuß erstellt hat sieht er dieses Symbol und will eine Verletzung auswählen, ist jedoch durch die Anwesenheit eines anderen Körperteils im Symbol verwirrt. Deshalb wurde ein kleiner Satz Symbole entwickelt, der die Art einer Verletzung als universal verständliches Piktogramm ohne

zusätzliche Körperteile repräsentiert. Die Abbildung enthält also nur eine abstrakte, bildliche Repräsentation der jeweiligen Verletzung.

Abbildung 4.8: Von links nach rechts: verletzter Vaterknoten, zwei Kindknoten; verletzter Vaterknoten, ein Kindknoten

Die Verletzung eines Körperteils soll immer in der linken oberen Ecke einer Körperregion angezeigt werden. Dabei soll die Verletzung des Vaterknotens in der linken oberen Ecke des Interaktionsbereichs und die des Kindknotens in der linken, oberen Ecke der Hitbox des Kindknotens angezeigt werden. Durch die ähnlich gehaltene Positionierung und der Darstellung innerhalb der Hitbox kann ein Benutzer einfacher unterscheiden, welche Verletzung welcher Körperregion zugewiesen wurde. Diese war dem aktuellen Körperteil zugeordnet, was aber auf Grund der angezeigten Kindknoten als eher verwirrend erschien. In diesem Ansatz hätten so die Kindknoten als Position der oben angezeigten Verletzung interpretiert werden können, was bei zwei Kindknoten nur Verwirrung gestiftet hätte. Wäre allerdings im Datenmodell statt n-Kindknoten nur einer vorgesehen gewesen, hätte diese Option durchaus eine verwendbare Lösung dargestellt. So hätte man die Verletzung des Kindknotens im Vater darstellen können, wobei der Kindknoten dann als Position der Verletzung gedient hätte (vgl. rechte Abb. 4.8).
Der nächste Schritt war die Verletzung optisch direkt dem betroffenen Körperteil zuzuordnen. Dazu wurde die Verletzung innerhalb des betroffenen Kindknotens angezeigt. Allerdings ergibt sich so die Frage, wie sich eine eventuelle Verletzung des Vaterknotens darstellen lässt. Das Übernehmen der ersten Art der Darstellung stellte sich als angemessene Lösung heraus. So wird die Verletzung des Vaterknotens in der linken oberen

Ecke angezeigt (vgl. Abb. 4.9). Um eine noch bessere Differenzierung zwischen einer Verletzung eines Kind- und Vaterknotens zu erreichen, wird das Icon des Vaters etwas größer angezeigt als das des Kindknotens.

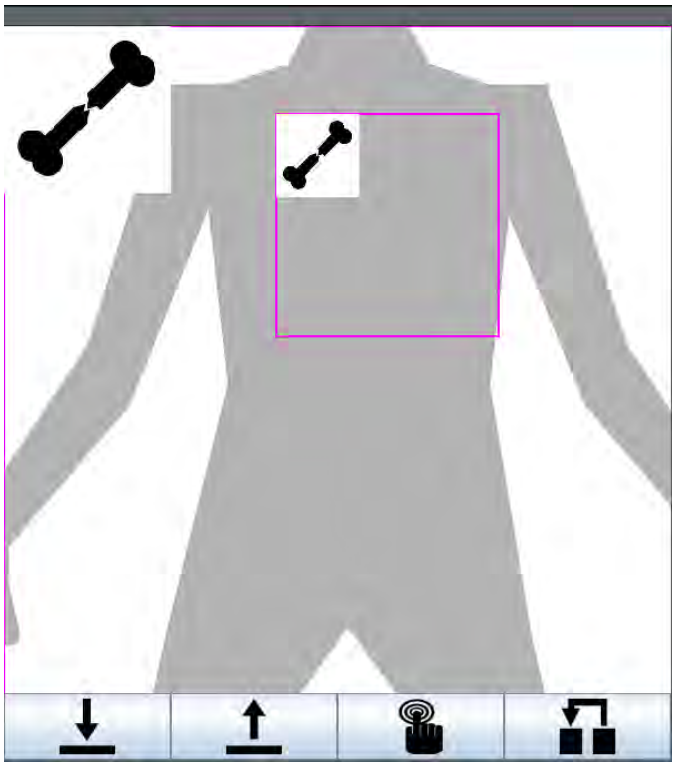

Abbildung 4.9: Eine verletzte Körperregion mit verletztem Kindknoten

4.2.4 Eingaben

Mauseingaben sollen hauptsächlich von einem in der SVG implementierten JavaScript Mouselistener entgegengenommen werden. Dieser bietet folgende sehr nützliche Eigenschaften und Vorteile:

- Das Mausevent finden nur auf dem eigentlichen, gezeichneten Körper statt. Das bedeutet: klickt der User neben den Körper wird auch kein Event gefeuert. Damit entfällt eine mühselige Implementierung von Grenzen und Hitboxen.

- Es müssen keine Bereiche auf dem Bild definiert werden, da für diesen Zweck dem Mouselistener ein Parameter übergeben werden kann. Dieser Parameter wird auf einen Wert gesetzt, der das ausgewählte Körperteil beschreibt.
- Der JavaScript-Mouselistener liefert sowohl x- als auch y-Koordinaten. Diese Koordinaten sollten nicht über Mouselistener auf der JSVGCanvas entgegengenommen werden, da die von der Apache Batik Library mitgebrachten Interactors den Mouselistener der SVG überlagern. In diesem Fall kann dann aber der String, der das getroffene Körperteil beschreibt, nicht mehr ausgelesen werden. Da die Koordinaten des JavaScript-Mouselisteners und die der JSVGCanvas-Mouselistener die selben Koordinaten sind, ist das nicht wirklich tragisch. Beide sind relative Angaben innerhalb der Viewbox der SVG und somit dem aktuellen Interaktionsbereich des Benutzers entsprechend.

Dabei sollen Linksklicks vom JavaScript-Mouselistener innerhalb der SVG übernommen werden. Wird ein Mausklick registriert, so soll der Interaktionsbereich um die Koordinaten dessen zentriert und vergrößert werden.
Rechtsklicks hingegen sollen von einem normalen Java Mouselistener verarbeitet werden. Dort soll keine Auswahl getroffen werden, sondern ein Level im Körper nach oben gegangen werden. Das heißt es soll der Zustand der Viewbox wiederhergestellt werden, die bei der Erstellung der Körperregion berechnet wurde.

5

Implementierung

Nun sollen Aspekte der Implementierung und Erstellung der zugehörigen Anwendung erläutert werden. Die Implementierung erfolgte in Java mit Hilfe der Apache Batik Library, die eine umfangreiche Möglichkeit zum Umgang und zur Verarbeitung von SVGs bietet. Die Anwendung soll einen skalierbaren Körper anzeigen und es möglich machen, eine Verletzung an einer speziellen Stelle des Körpers anzugeben. Eingegangen wird hierbei speziell auf die Benutzeroberfläche (siehe Kapitel 5.1), die persistente Speicherung von erstellten Körpermodellen (siehe Kapitel 5.2) und die Implementierung des Datenmodells (siehe Kapitel 5.3).

5.1 Benutzeroberfläche

Die Benutzeroberfläche ist einer der wichtigsten Teile der Implementierung, da sich die Anwendung um eine universell verständliche Möglichkeit zu Verständigung handeln soll. Dafür wurde die Benutzeroberfläche sehr minimal gehalten und auf eine möglichst geringe Benutzung typografischer Elemente geachtet. Dabei wird auf Aspekte der Implementierung der grafischen Oberfläche (siehe Kapitel 5.1.1) und die Skalierung dieser (siehe Kapitel 5.1.4) eingegangen. Darüber hinaus wird die Erstellung der Auswahldialoge (siehe Kapitel 5.1.2) und Markierungen durch Overlays (siehe Kapitel 5.1.5) aufgeführt. Außerdem wird die Implementierung der Benutzereingaben (siehe Kapitel 4.2.4) erläutert.

5.1.1 Oberfläche und Körpermodell

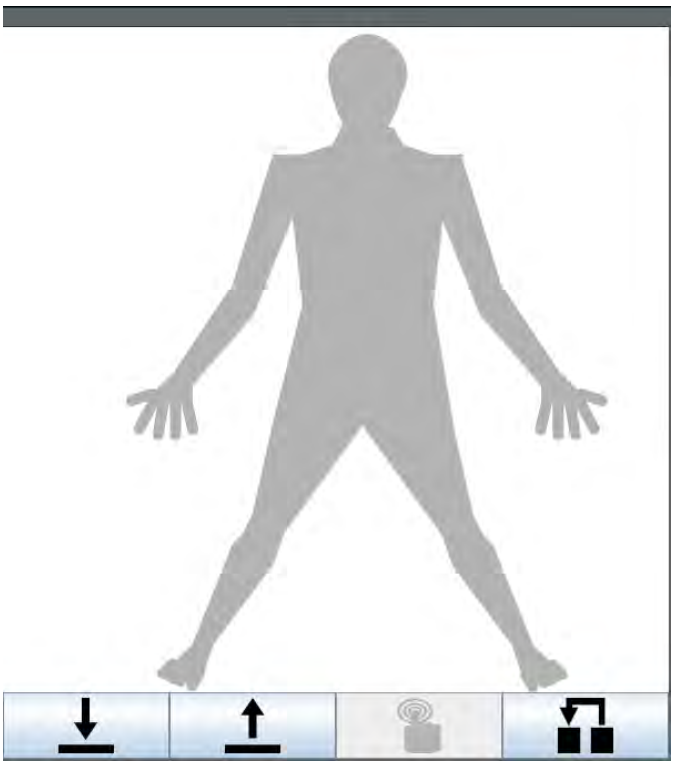

Abbildung 5.1: Screenshot der GUI

Die Oberfläche besteht aus einem Körpermodell, welche eine SVG (vgl. Abb. 4.3) sowie vier Buttons enthält. Die Buttons haben die Funktion von Speichern, Laden, Resetten des Körpermodells und Starten eines Auswahldialogs zur Selektion einer Verletzung. Die SVG des Körpermodells wird in einer JSVGCanvas der Apache Batik Library auf der grafischen Oberfläche angezeigt. Das Körpermodell stellt den eigentlichen Interaktionsbereich des Benutzers dar. Dort werden dem Benutzer verfügbare Informationen bereitgestellt und es lässt den Benutzer die eigentlich Eingaben machen, das heißt Körperregionen definieren und Verletzungen zuweisen. Die Verwendung der JSVGCanvas ermöglicht unter anderem auch die Manipulation der Viewbox der SVG (siehe Kapitel 2.3.2).

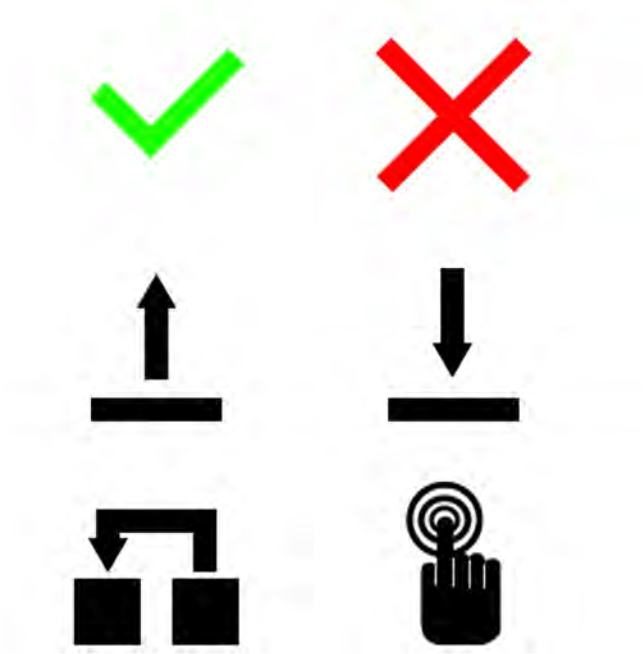

Abbildung 5.2: Von links nach rechts: 1. Reihe: Bestätigen, Abbrechen; 2. Reihe: Laden, Speichern; 3. Reihe: Reset, Auswahl

Auf Beschriftung von Buttons wurde so weit wie möglich verzichtet, und durch gängige, bereits vorhandenen Symbole ersetzt (vgl. Abb. 5.2). Für Speichern und Laden wurden hier die allgemein bekannten Symbole für Download und Upload verwendet, da diese beiden Symbole ähnlich aussehen und durch Pfeile ein Hineinlegen und Herausnehmen von etwas darstellen. Zum bestätigen wurde ein grüner Haken verwendet, da dies das gängige Symbol für diese Art von Eingabe ist. Als Symbol für Abbruch wurde das bekannte Symbol eines roten Kreuzes ("X") verwendet. Der Reset-Knopf erhält ein eigens entworfenes Icon das das Zurücksetzten verbildlicht. Dieses Icon enthält zwei benachbarte Quadrate und einen Pfeil. Dieser Pfeil hat seinen Ursprung im rechten Quadrat und zeigt in einem Bogen auf das linke. Dies soll das Wiederherstellen eines vorherigen Zustands symbolisieren. Das Auswahl-Symbol stellt eine Hand dar, die etwas mit dem Zeigefinger berührt. Unter diesem befindet sich eine Art "Bullseye”, was das Berühren optisch darstellen soll.

Bei einem Klick auf den Speichern-Knopf wird die `encodeJSON(Body)` der Klasse `JSONWriter` aufgerufen. Diese speichert den aktuellen Körper als JSON-Datei, um

diesen später wiederherstellen zu können (5.2). Klickt man auf den Lade-Knopf, so wird ein Java-Filechooser aufgerufen. Wird dann eine Datei ausgewählt, wird die Methode `decodeJSON(String)` der Klasse `JSONReader` mit dem Pfad der ausgewählten Datei ausgeführt. Hierdurch wird aus einer JSON-Datei ein vorher erstellter Körper wiederhergestellt (5.2). Beim Reset-Knopf wird wie schon beim Laden-Knopf eine JSON-Datei geladen, jedoch ohne Java Filechooser. Diese JSON-Datei enthält einen leeren Körper.

5.1.2 Auswahldialoge

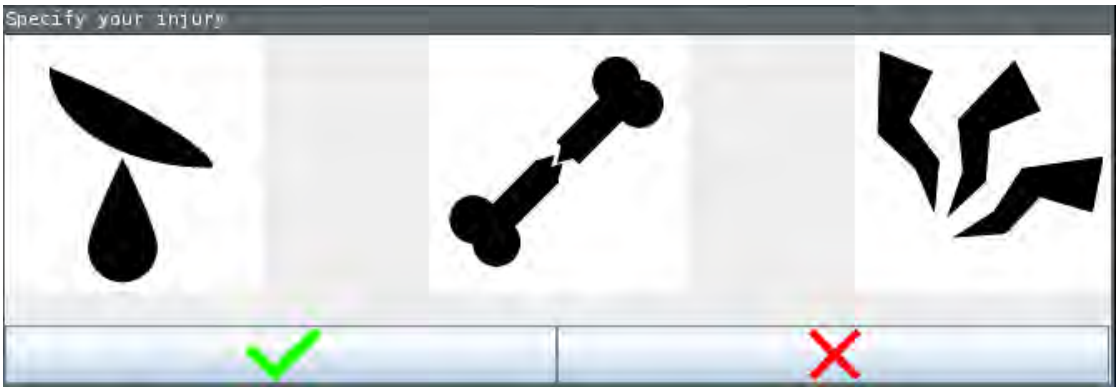

Abbildung 5.3: Dialog zur Auswahl einer Verletzung

Zur Auswahl von Verletzungen sowie zum Laden vorher erstellter Körper wurden Auswahldialoge implementiert. Der Dialog zum Laden ist durch einen normalen Java-Filechooser implementiert worden. Dieser bietet alle nötigen Funktionen und kann auf einen Default-Pfad gesetzt werden. WIrd eine Datei ausgewählt, wird der in der ausgewählten JSON-Datei enthaltene Körper wiederhergestellt.

Für die Auswahl von Verletzungen wurde eine eigene Dialogklasse entwickelt (vgl. Abb. 5.3). Diese zeigt einen Katalog von vorhanden Verletzungen an, welche dann ausgewählt werden können. Wird eine Verletzung ausgewählt, wird diese durch eine Umrahmung markiert (vgl. Abb. 5.4). Wird der Bestätigen-Knopf gedrückt, so wird die Art der Verletzung in einer Variable gespeichert. Diese wird nach Beendigung der Dialog-Oberfläche über eine `getValue()`-Funktion der InjuryDialog-Klasse von

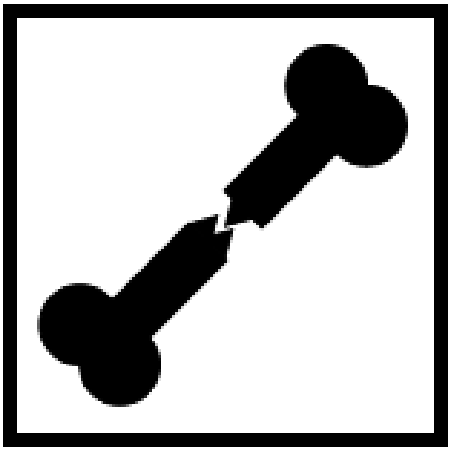

Abbildung 5.4: Selektiertes Icon des InjuryDialogs

der GraphicalUserInterface-Klasse abgerufen. Anschließend wird diese Verletzung im `currentBodyPart` gespeichert und durch ein Overlay (siehe Kapitel 5.1.5) dem Benutzer angezeigt. Da in der Regel nicht der ganze Körper eine Verletzung hat, ist die Auswahl einer Verletzung erst ab Level 1 der Baumstruktur möglich. Um eine Auswahl auf Level 0 zu verhindern, wird der Knopf zur Auswahl von Verletzungen auf Level 0 einfach deaktiviert.

5.1.3 Eingaben

Eingaben werden ausschließlich anhand von Mausklicks durchgeführt. Hierfür wird hauptsächlich auf einen in JavaScript implementierten Mouselistener innerhalb der SVG zurückgegriffen, welcher die Position des Klicks sowie einen Bezeichner des <g>-Elements an dieser Stelle empfängt. Diese werden zu einem String der Form `posX:posY:location` zusammengefasst.
Die Events, die dieser Mouselistener liefert, werden durch den SVGUserAgent der JSVGCanvas ausgelesen. Ein Mausklick auf eine SVG innerhalb einer JSVGCanvas löst normalerweise einen Alert beziehungsweise einen kleinen Dialog aus, der den Inhalt des zuvor genannten Strings ausgibt. Wenn im Konstruktor der JSVGCanvas allerdings ein SVGUserAgent übergeben wird, kann diese Methode überschrieben werden. Der Methode `showAlert()` empfängt nun den String, der normalerweise in dem Alert

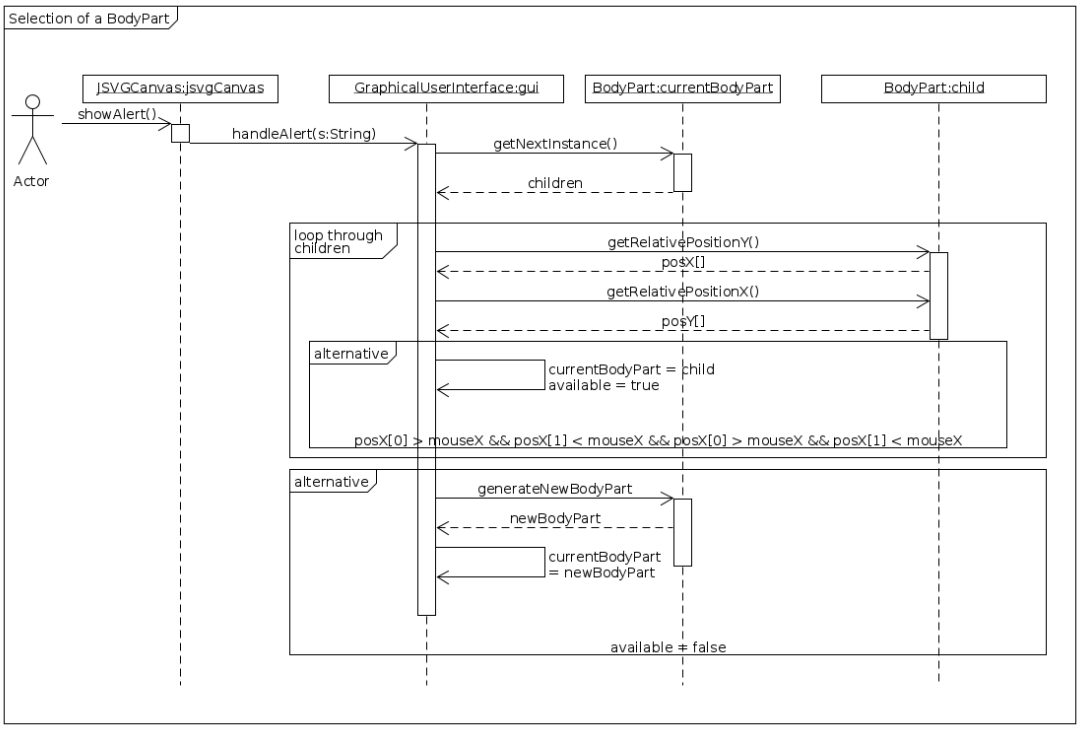

Abbildung 5.5: Ablauf der Auswahl einer Körperregion

ausgegeben wird. Dieser String wird anschließend wieder zerteilt und verarbeitet.
Um die angeklickte Körperregion bestimmen zu können, werden `relPosX` und `relPosY` der Kindknoten des `currentBodyPart` mit den Koordinaten des Mausklicks verglichen. Wenn ein Match gefunden wird, wird dieser Kindknoten als neues `currentBodyPart` gespeichert. Wird allerdings kein Match gefunden, wird ein neuer Kindknoten des `currentBodyPart` erzeugt und als neues `currentBodyPart` festgelegt. Wurde ein Match gefunden oder ein neues Körperteil erstellt, wird die `AffineTransform` (siehe Kapitel 2.6) des `currentBodyParts` geladen.
Die Klasse `GraphicalUserInterface` implementiert das Interface `MouseListener` von Java. Durch dieses Interface kann die Verarbeitung aller Arten von Mauseingaben (zum Beispiel `mouseClicked` oder `mouseReleased`) in einzelnen Methoden definiert werden. Hier wird allerdings nur die Methode `mouseClicked` verwendet. Dort wird zwischen den einzelnen Buttons und JSVGCanvas unterschieden. Bei einem Rechtsklick auf die JSVGCanvas wird im Gegensatz zu einem Linksklick nicht weiter nach unten durch die Baumstruktur traversiert oder diese erweitert, sondern einen Knoten nach oben traversiert. Dabei wird dann einfach nur das `currentBodyPart` auf den Vaterknoten des aktuellen Knotens gesetzt und die `AffineTransform` dessen als `RenderingTransform` gesetzt (5.1.4).

5.1.4 Skalierung des Körpermodells

Die `AffineTransform` wird in einer Methode `adjustView()` berechnet und aktiviert. Da sich eine `AffineTransform` nicht richtig in einer JSON-Datei speichern lässt, wird diese anhand der Position der Körperregion neu erstellt. Ist also ein Körperteil ausgewählt, dessen `affineTranform`-Attribut auf `null` gesetzt ist, so wird diese einfach neu erzeugt und dem Körperteil zugewiesen.
Um den gewünschten Bildausschnitt zu berechnen, wird eine neue `AffineTransform` erstellt und zuerst mittels der `scale(x, y)`-Funktion das Bild um den `zoomFactor` skaliert. Hierbei bedeutet ein Wert größer 1 eine Vergrößerung und ein Wert kleiner 1 eine Verkleinerung der SVG. Da beim Herauszoomen die Viewbox nach rechts unten

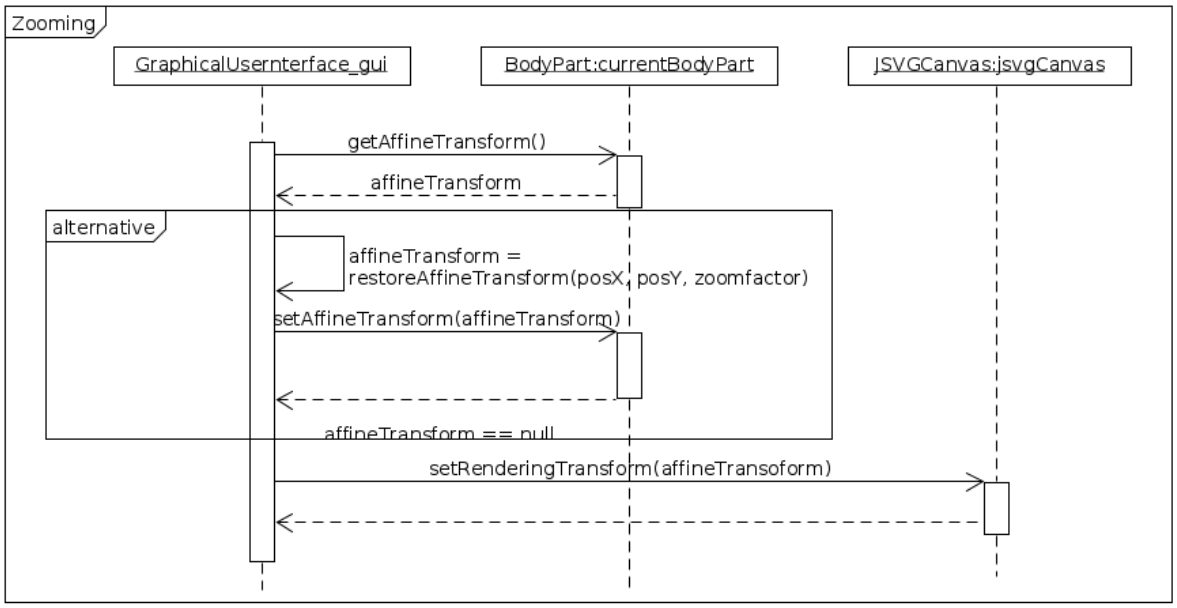

Abbildung 5.6: Ablauf eines Zooms

verschoben wird, muss die Viewbox durch die`translate(x, y)`-Funktion korrigiert werden. Dies geschieht folgendermaßen:

1. Zwei Integer-Variablen, `dx` und `dy`, werden erstellt. Deren Inhalt ist 600 * `zoomFactor`, wobei 600 der Breite und Höhe der Viewbox entspricht. Die Variable `zoomFactor` stellt den Vergrößerungsfaktor dar. Hier bedeutet `zoomFactor` < 1 ein Hineinzoomen und `zoomFactor` > 1 ein Herauszoomen. Verwendet wird hier nur ersteres, da das Herauszoomen anders realisiert wird (siehe unten). Die Multiplikation mit dem `zoomFactor` ist insofern nötig, da die Koordinaten des Mausklicks auf der noch unskalierten SVG stattfanden und diese dann in die neuen skalierten Koordinaten umgerechnet werden müssen.
2. Die Variable `scale` entspricht den Werten `sx` und `sy` der `AffineTransform` (siehe Kapitel 2.6). Da die SVG quadratisch ist, ist der Wert für `sx` und `sy` derselbe. Der Inhalt der Variable entspricht hier der Höhe oder Breite der Viewbox geteilt durch `dx` beziehungsweise `dy`.
3. Eine neue, leere `AffineTransform` wird erstellt.

4. Die Funktion `scale(scaleX, scaleY)` wird mit der Variable `scale` als Parameter aufgerufen.

5. Die Viewbox wird mit der Methode `translate(transX, transY)` verschoben. Diese wird mit den Parametern `-x + dx / 2` oder `-y + dy / 2` verschoben. Dabei muss die SVG um den Wert der x- beziehungsweise y-Koordinate von der Mitte aus nach links beziehungsweise oben verschoben werden, um die Verschiebung durch `scale(scaleX, scaleY)` zu korrigieren. Da die Verschiebung von der Mitte der Viewbox aus stattfindet, muss `dx` und `dy` durch 2 dividiert und zu x beziehungsweise y addiert werden.

6. Die so erstellte `AffineTransform` wird nun mit der `RenderingTransform` der JSVGCanvas mittel `concatenate(AffineTransform at)` verrechnet. Das Attribut `RenderingTransform` beschreibt die aktuelle Viewbox der SVG.

7. Die neuen daraus resultierende `AffineTransform` wird nun der JSVGCanvas als neue `RenderingTransform` übergeben.

Für das Herauszoomen wird jedoch nicht diese Funktion verwendet. Hierfür wird einfach die `AffineTransform` der ausgewählten Körperregion als `RenderingTransform` geladen, was dann genau den selben Bildausschnitt zufolge hat, wie beim erstellen dessen.
Wird ein Match im Selektionsverfahren gefunden, wo allerdings die `AffineTransform` nicht gesetzt ist, wird mittels der `restoreAffineTansform(x, y)` die entsprechende `AffineTransform` mittels der `relX`- und `relY`-Attribute berechnet. Dies ist der Fall, wenn ein Körper geladen wird aus einer JSON-Datei, da sich `AffineTransform` nicht vernünftig in JSON-Dateien speichern lassen.

5.1.5 Overlays

Um dem Benutzer anzuzeigen, wo er bereits Bereiche auf dem Körper definiert hat, werden bereits erstellte Bereiche als farbige Rechtecke angezeigt (vgl. Abb. 4.4). Diese werden mit den Overlay der Apache Batik Library realisiert. Zur Anzeige von Körperregionen werden simple Rechtecke gezeichnet und über den definierten Bereichen

angezeigt. Da dies mit einem Java Graphics-Objekt geschieht, lassen sich so auch für bereits angegebene Verletzungen die jeweiligen Icons als Overlays anzeigen. Gezeichnet werden Overlays nach jedem Heraus- oder Hineinzoomen. Sobald ein neues `currentBodyPart` gefunden oder erstellt wurde, werden dessen Kinder und Verletzung mittels Overlays gezeichnet.
Da die Methode `paint(Graphics g)` vom Overlay-Interface kommt, kann diese nicht zweimal implementiert werden. Um trotzdem zu unterscheiden ob ein Icon oder eine Körperregion gezeichnet werden soll, existiert eine Variable `injured` vom Typ `Boolean`. Um diesen zu setzten, gibt es zwei verschiedene Konstruktoren. Beide erwarten eine JSVGCanvas, Positionsangaben und eine Farbe. Der Konstruktor zum Zeichnen von Icons erwartet zusätzlich noch eine Objekt vom Typ `Injury`. Wird dieser Konstruktor gesetzt, so wird `injured` auf `true` gesetzt, andernfalls ist der Wert `false`. In der Methode `paint(Graphics g)` wird dann eine Fallunterscheidung durchgeführt und je nach Wert von `injured` ein Icon oder eine Körperregion gezeichnet.

5.2 Persistente Speicherung

Um eine spätere Auswertung eines verletzten Körpers zu ermöglichen, wurde ein Modul zum speichern eines `Body`-Objekts entwickelt. Der generelle Aufbau eines JSON-Objekts ohne spezielle Attribute (Finger, Zehen, etc.) wird in Abbildung 5.7 dargestellt. Beim Speichern eines `Bodys` werden rekursiv die Kinder der einzelnen Körperteile in JSON-Strings gespeichert. Dabei werden alle Attribute der abstrakten Klasse `BodyPart` in einer Methode in ein JSON-Objekt verpackt. Bei jedem Objekt wird überprüft, ob es sich um Hand, Fuß oder Kopf handelt. Ist das der Fall, so werden die zugehörigen Finger, Zehen, Augen und Ohren dem JSON-Objekt hinzugefügt. Dabei entsteht ein Abbild der Baumstruktur des Datenmodells innerhalb der JSON-Datei.
Beim Laden einer `Body`-JSON-Datei werden rekursiv die einzelnen Kinder des Körpers wiederhergestellt. Dabei wird wie beim Speichern eine Methode verwendet, die die generalisierten Attribute eines Körperteils wiederherstellt. Anschließend werden wieder eigene Methoden für jede Klasse verwendet, um spezielle Attribute wiederherstellen und eine korrekte Typisierung gewährleisten zu können.

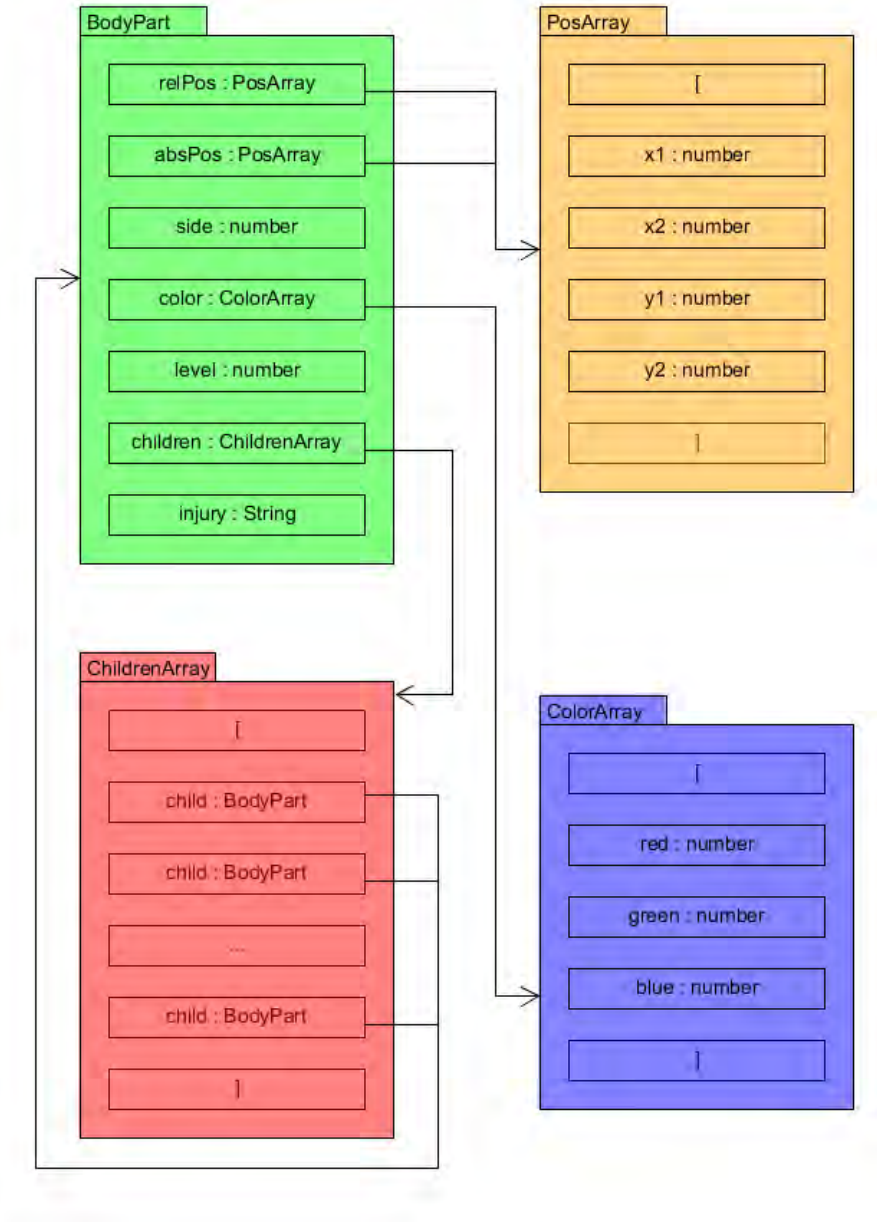

Abbildung 5.7: Aufbau einer JSON

5.3 Datenmodell

Das Datenmodell bildet den Kern der Anwendung. Hier wird das Körpermodell mit seinen Funktionen versehen und die Körperregionen verwaltet. Das gesamte Modell stellt alle nötigen Funktionen zur Erfassung von Verletzungen und Erstellen neuer Körperregionen zur Verfügung. In diesem Kapitel soll primär auf den Aufbau und die Struktur des Modells eingegangen werden (siehe Kapitel 5.3.1). Außerdem wird auch die Generierung neuer Körperregionen (siehe Kapitel 5.3.2) und das Traversieren durch die Baumstruktur des Datenmodells erläutert (siehe Kapitel 5.3.3).

5.3.1 Aufbau und Struktur

Der programmtechnische Aufbau des Datenmodells erfolgt als erstes über eine abstrakte Klasse `BodyPart`, welche den Grundstein jedes anderen Körperteils bildet. Diese Klasse enthält alle allgemeinen Attribute, die ein abgeleitetes Körperteil besitzt. Wichtig für die Navigation durch einen Körper sind hierbei die Attribute `level`, `nextInstance` (Liste von Kindknoten), sowie `previousInstance` (Vaterknoten). Außerdem ist ein Attribut `injury` enthalten, über welches eine eventuelle Verletzung des Körperteils gespeichert werden kann. Hinzu kommen Positionsangaben für eine absolute Position auf dem Körper (`absPosX` und `absPosY`) und zur relativen Positionierung abhängig vom Vaterknoten (`relPosX` und `relPosY`). Darüber hinaus besitzt die Klasse `BodyPart` noch Attribute, die Informationen zur Anzeige beinhalten. Diese sind `color`, was die Farbe der Markierung enthält, und `affineTransform`, was den anzuzeigenden Bildausschnitt darstellt (siehe Kapitel 2.6). Die Klasse enthält außerdem folgende Methoden:

- `BodyPart : generateNewBodyPart(BodyPart)` : Erstellt eine neue Körperregion als Kindknoten der Körperregion, die als Parameter übergeben wurde. Dabei wird überprüft, welche Klasse das übergebene `BodyPart` instantiiert und anschließend wird eine Instanz der entsprechenden Klasse erstellt. Die erstellte Körperregion wird dann der `nextInstance`-Liste des Vaterknotens und der Vaterknoten dem Kindknoten als `previousInstance` hinzugefügt.

Abbildung 5.8: Komplettes UML-Klassendiagramm der Anwendung (Package Data siehe Abb. 4.2)

- `BodyPart : removeUnnecessaryChildren()`: Mit dieser Methode wird die Baumstruktur aufgeräumt. Dabei wird überprüft, ob dem übergebenen `BodyPart` eine Verletzung zugewiesen wurde. Anschließend wird die Liste der Kindknoten überprüft. Sind keine Verletzungen in diesem Zweig vorhanden, wird der Zweig entfernt. Da dann alle Verbindungen getrennt sind, wird das Objekt vom Garbage Collector entsorgt. Körperteile des ersten oder nullten Levels können dabei nicht entfernt werden.
- `backTrackRec(int, BodyPart)` und `printChildren(BodyPart)`: Diese beiden Methoden sind Debug-Methoden, mit denen rekursiv entweder von unten nach oben (`backTrackRec`) oder von oben nach unten (`printChildren`) alle Kinder eines Elements ausgegeben werden können.

Alle weiteren Körperteile erben von der Klasse `BodyPart`. Die Klasse `Body` fungiert hier als eine Art Controller. Ein `Body` hat je Klasse ein Attribut, das heißt der Körper besitzt einen `mainArmLeft`, einen `mainHead` und so weiter. Finger werden hierbei direkt der Hand zugewiesen, genau wie Augen und Ohren beziehungsweise Zehen dem Kopf oder dem Fuß zugewiesen werden. Wenn diese Körperteile erstellt werden, werden Händen, Füßen und dem Kopf die jeweiligen Finger, Zehen, Augen und Ohren im Konstruktor erstellt und zugewiesen.
Es wurden bewusst linke und rechte Körperteile implementiert, um einen höheren Grad der Typisierung zu schaffen. Dies ermöglicht eine präzisere Unterscheidung von zum Beispiel rechten und linken Armen. Bei Zehen, Fingern etc. wurde hierauf verzichtet, da die zugehörige Hand beziehungsweise Fuß der linken oder rechten Seite des Körpers zugeordnet werden können.
Abseits der Klassen die Körperteile repräsentieren existiert noch eine Klasse `Injury`. Diese stellt eine Verletzung dar, die bestimmte Formen annehmen kann. Die Art der Verletzungen sind innerhalb einer Java `enum` definiert. Zudem sind statische Variablen vorhanden, welche den Pfad zu den zugehörigen Icons enthalten.

5.3.2 Erstellen neuer Körperregion

Wird während des Selektionsprozesses (siehe Kapitel 5.1.3) keine Körperregion an der Mausposition gefunden, so wird dort eine neue Körperregion erstellt. Zum Erstellen eines neuen Kindknotens existiert eine Methode innerhalb der abstrakten Klasse `BodyPart` namens `generateNewBodyPart(BodyPart)`. Da jede Körperklasse von `BodyPart` erbt, steht diese Methode jeder Klasse zur Verfügung. Da jedoch nicht jede aufrufende Instanz die selbe Klasse instantiiert, muss erst die passende Klasse gefunden werden. Dazu wird die aufrufende Instanz mit allen infrage kommenden Klassen verglichen. Dies wird durch eine Reihe von Fallunterscheidungen realisiert, die die aufrufende Instanz mittels `instanceof` vergleichen.

Wird ein Match gefunden, so wird der Konstruktor der entsprechenden Klasse aufgerufen. Dort wird das aufrufende Körperteil-Objekt als Vorgängerobjekt (`previousInstance`) gesetzt. Zudem wird das neu erstellte Objekt der Liste der Kindknoten des aufrufenden Objekts hinzugefügt.
Bei Händen, Füßen und beim Kopf werden zudem Finger, Zehen, Augen und Ohren hinzugefügt. Sind diese im aufrufenden Objekt vorhanden, werden diese vom neuen Kindknoten übernommen, andernfalls wird ein neues Set erstellt. Da meistens zwei Augen und Ohren und fünf Finger und Zehen vorhanden sind, werden diese in einem Array mit je zwei beziehungsweise fünf Plätzen erstellt.

5.3.3 Traversieren durch den Körper

Um durch die Baumstruktur des Körpers zu traversieren, wird, um weiter nach unten zu kommen, die Liste der Kindknoten verwendet und um wieder weiter nach oben zu kommen, der Vaterknoten des aktuellen `BodyParts` verwendet. Um zu navigieren wird dabei einfach der entsprechende Kind- oder Vaterknoten ausgewählt und als `currentBodyPart` gesetzt. Bei einigen Methoden, wie zum Beispiel der Hilfsmethode `removeUnnecessaryChildren()`, wird allerdings rekursiv vorgegangen. Dabei wird die rekursive Methode iterativ für jedes Element der Kindliste aufgerufen.

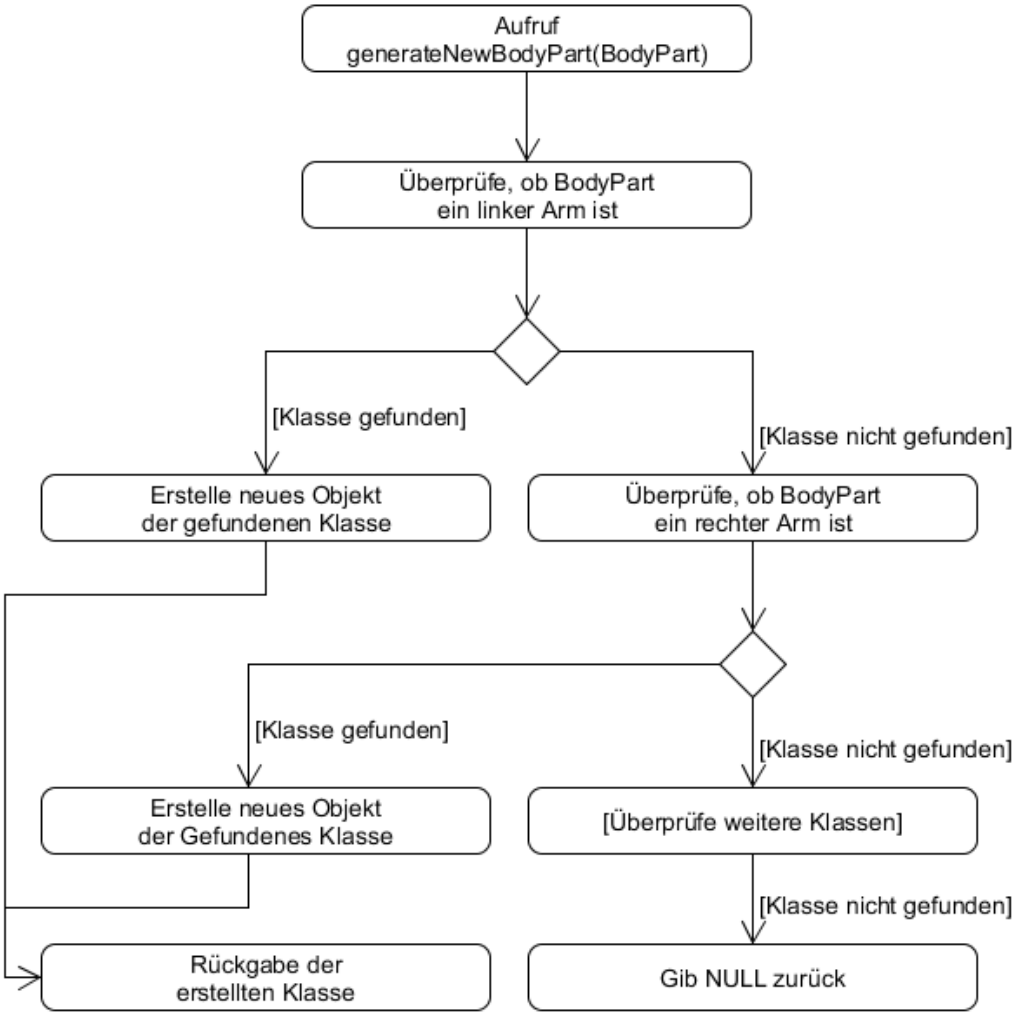

Abbildung 5.9: Schemenhafter Ablauf der Erstellung einer neuen Körperregion

6

Anforderungsabgleich

Im Anforderungsabgleich werden die funktionalen (siehe Kapitel 6.1) und nicht-funktionalen (siehe Kapitel 6.2) Anforderungen an die Anwendung aus dem Grundlagenkapitel (siehe Kapitel 3) mit den umgesetzten Funktionen und Charakteristika der Anwendung verglichen. Dabei wird die Wichtigkeit und wie viel Wert auf die einzelnen Anforderungen gelegt wurden mit folgender Symbolik angegeben:

- + + Anforderung vollständig erfüllt
- + Anforderung erfüllt
- - Anforderung teilweise erfüllt
- - - Anforderung konnte nicht erfüllt werden

6.1 Funktionale Anforderungen

Funktionale Anforderungen definieren den Funktionsumfang der Anwendung, welche dem Benutzer später zur Verfügung stehen sollen.

6.1.1 Grafische Benutzeroberfläche

I. *Definieren neuer Körperregionen*: + +
Dem Benutzer ist es möglich, neue Körperregionen durch Mauseingaben zu definieren.

II. *Auswählen von Verletzungen*: + +
Der Benutzer kann einer Körperregion eine Verletzung zuweisen.

III. *Navigation durch Baumstruktur*: + +
Dem Benutzer ist es möglich durch Mauseingaben durch die Baumstruktur des Körpers zu navigieren

IV. *Anzeigen bereits erstellter Körperteile*: +
Bereits erstellte Körperteile werden durch farbige Rechtecke gekennzeichnet. Die Art der Darstellung bedarf eventuell einer Überarbeitung.

V. *Speichern-Knopf*: + +
Ein Speichern-Knopf wurde implementiert und ist funktionsfähig.

VI. *Laden-Knopf*: + +
Dem Benutzer ist es möglich durch einen Laden-Knopf eine vorher erstellte Instanz auszuwählen und wiederherzustellen.

VII. *Reset-Knopf*: + +
Dem Benutzer ist es möglich sein durch einen Reset-Knopf den ursprünglichen Zustand der Anwendung wiederherzustellen.

6.1.2 Persistenz

I. *Speichern des aktuellen Zustands*: + +
Es ist möglich die aktuelle Instanz vollständig in einer JSON-Datei zu speichern.

II. *Laden eines vorherigen Zustands*: + +
Eine vorher erstellte Instanz eines Körpers kann aus einer JSON-Datei wiederhergestellt werden.

6.1.3 Datenmodell

I. *Generisches Erstellen von Körperregionen*: + +
Das Datenmodell ermöglicht eine generische Erzeugung von Körperregionen.

II. *Spezifizierung der Position*: +
Die Position eines Körperteils wird relativ zum Vaterknoten gespeichert, jedoch nicht absolut auf dem kompletten Körper.

III. *Zuweisung einer Verletzung*: + +
Einer Körperregion kann eine Verletzung zugewiesen werden.

6.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen definieren die Art und Weise wie die Anwendung umgesetzt werden soll und welche Eigenschaften abseits der Funktionalität erfüllt werden sollen.

I. *Technische Anforderungen*: + +
Die Anwendung ist mittels Java 1.8 implementiert und in einer entsprechenden Java VM lauffähig.

II. *Ergonomische Anforderungen*: +
Die Bedienung ist so einfach wie möglich gehalten.
Auf typografische Elemente wurde so weit wie möglich verzichtet.

III. *Anforderungen an Dienstqualität*: + +
Die Antwortzeit der Anwendung ist minimal.

IV. *Zuverlässigkeit*: + +
Falscheingaben sind nicht möglich.
Die Anwendung läuft stabil und konnte nicht zum Absturz gebracht werden.

V. *Wart- und Erweiterbarkeit*: +
Sowohl Anwendung als auch Datenmodell sind Modular aufgebaut, um einen möglichst hohen Grad an Erweiter- und Wartbarkeit zu gewährleisten.

VI. *Verwendung einer Rule-Engine*: - -
Es wurde keine Rule-Engine verwendet, da sich kein sinnvoller Anwendungszweck ergab.

7

Zusammenfassung

In diesem letzten Kapitel wird ein Ausblick zu denkbaren Weiterentwicklungen und Ergänzungen gegeben (siehe Kapitel 7.1), sowie ein Fazit (siehe Kapitel 7.2) gezogen.
Die Implementierung der Anwendung wurde anhand eines möglichst generischen Datenmodells erstellt. Dabei war es wichtig, ein Körperteil möglichst allgemein zu halten und dieses durch generische Methoden verarbeiten zu können. Eine Körperregion hat einen Vaterknoten und n-Kindknoten. Zudem hat eine Körperregion Positionsangaben, was es ermöglicht an jeder beliebigen Stelle des Körpers Regionen definieren zu Können. Die Verwendung einer SVG als Körpermodell und die durch eine Vater-Kind-Beziehung entstehende Baumstruktur ermöglichen so auch einen beliebigen Grad der Verfeinerung, da das Körpermodell beliebig weit skaliert werden kann. Dabei werden die Kindknoten als innerhalb des Vaterknotens befindliche Körperregionen interpretiert und das Körpermodell so skaliert und verschoben, dass die neue Körperregion zentriert dargestellt wird.
Die Anwendung ist gestalterisch so gehalten, dass eine Bedienung ohne Kenntnisse einer speziellen Sprache möglich ist. Zu eben jenem Zweck wurden allgemein bekannte Symbole aus den Bereichen Web und mobiler Anwendungen, die heutzutage so gut wie jedem bekannt sein dürften, in die Anwendung eingebunden. Durch die Art der Implementierung der Eingaben, die sich rein auf simple Mausklicks beschränken, sind außerdem keine Falscheingaben möglich. Das macht Eingaben vorhersehbar, wodurch die Anwendung auch nicht zum Absturz gebracht werden kann. Zudem wurde für die Selektion von Verletzungen ein eigener Auswahldialog implementiert, welcher auf einen Katalog von Symbolen zurückgreift. Diese Symbole stellen verschiedene Arten von Verletzungen dar, wobei versucht wurde diese so zu gestalten, dass sie allgemein

verständlich sind. Wird dort eine Verletzung ausgewählt, so wird diese an der ausgewählten Stelle vermerkt und angezeigt. Erstellte Körper lassen sich außerdem in dem plattformübergreifenden JSON-Format abspeichern und später wiederherstellen.

7.1 Ausblick

Da es sich bei der erstellten Anwendung nur um eine Konzeptimplementierung handelt, können hier noch einige nützliche Aspekte umgesetzt werden. Gerade für den tatsächlichen Anwendungsfall wären einige weitere Funktionen sehr nützlich. Ein paar werden im Folgenden aufgeführt.

7.1.1 Neue Implementierung

Es sollte eine neue Implementierung erstellt werden, da nur ein Konzept zur Umsetzung des Problems erstellt wurde. Hier könnte vor allem eine neue Oberfläche entworfen werden, die mehr Wert auf Gestaltung legt. Zudem ist bisher nur ein begrenzter Katalog von Verletzungen enthalten. Dieser sollte noch erweitert werden, da die vorhandenen Verletzungen bei weitem nicht alle Möglichkeiten abdecken.

7.1.2 Anmeldefunktion

Um den Ablauf mit möglichst geringem Verwaltungsaufwand durchführen zu können, sollten sich Benutzer in irgendeiner Form anmelden können (zum Beispiel durch eine Nummer oder ein Benutzerkonto). Dadurch können Daten für einen bestimmten Nutzer gespeichert und zum Beispiel zu einer Akte hinzugefügt oder verlinkt werden. So würde ein erstellter Körper einfach einem Patienten zugeordnet werden können.

7.1.3 Auswertungsmodul

Durch das verwendete Format der gespeicherten Daten, kann die Auswertung automatisiert erfolgen. Dabei kann die JSON-Datei ausgelesen werden und bestehende

Verletzungen können verarbeitet werden. Dieses Modul kann in einer beliebigen Form umgesetzt werden, da das JSON-Format plattformunabhängig ist und von so gut wie jeder Programmiersprache verarbeitet werden kann. Durch dieses Auswertungsmodul können beispielsweise neue Einträge in einer Akte erstellt oder dem Patient ein Termin bei einem Spezialisten zugewiesen werden, ohne zusätzlichen Verwaltungsaufwand betreiben zu müssen [17, 18].

7.1.4 Portierung auf mobile Systeme

Auf einem Tablet-PC könnte die Anwendung in einer Arztpraxis oder in einem Krankenhaus einfach ausgelegt werden. So könnte die Zuweisung zu einem Spezialisten in Verbindung mit dem vorher erwähnten Auswertungsmodul automatisiert werden. Von Vorteil wäre hier eine Portierung auf ein Android-basiertes System, da hier einzelne Module der Konzeptanwendung übernommen werden können [19, 18, 20, 21, 17, 22].

7.1.5 Neugestaltung des Körpermodells

Da das aktuelle Körpermodell sehr reduziert ist, würde es sich hier anbieten, ein detaillierteres Modell zu erstellen. Bei einem hohen Skalierungsfaktor fällt zum Beispiel die Navigation etwas schwer, da sich keine Details auf dem Körper befinden.

7.1.6 Neues Selektion-Icon

Das Icon des Selektion-Knopfs ist zugegebenermaßen nicht optimal. Deshalb sollte ein neues Symbol entwickelt werden, welches den Sachverhalt eventuell etwas besser darstellt.

7.2 Fazit

Das Datenmodell und die dazugehörige Anwendung sind lediglich ein erster Schritt in Richtung einer Lösung des Problems von Sprachbarrieren. Durch eine Weiterentwick-

lung der Anwendung sowie eventuelle Modifikationen des Datenmodells, könnte der Nutzen in einem tatsächlichen Anwendungsszenario noch immens vergrößert werden. Der bisherige Stand der Konzeptanwendung ist hierbei allerdings noch in einem noch sehr frühen Stadium der Entwicklung. Trotzdem, dass schon ein Großteil der Funktionalität implementiert ist und sich auch Verletzungen an bestimmten Stellen angeben lassen, ist noch einiges zu tun. Vor allem der Aspekt der Verarbeitung der durch die Anwendung erhaltenen Datensätze ist im jetzigen Stadium sehr unkomfortabel. Wie schon im Abschnitt Ausblick (siehe Kapitel 7.1) erklärt ist noch eine ganze Reihe sinnvoller Erweiterungen denkbar, die zu einer Erleichterung von Aufnahmeabläufen beitragen könnten. Alles in allem wurde hier aber zumindest ein Grundstein dafür gelegt.

Literaturverzeichnis

[1] www.sportsinjuryclinic.net: Sports Injury Clinic. http://www.sportsinjuryclinic.net/about-us/iphone-app (2012) abgerufen am: 23.03.2016.

[2] Lingraphica: SmallTalk Intensive Care. https://www.aphasia.com/products/communication-practice-apps/ (2015) abgerufen am: 23.03.2016.

[3] W3C: Introduction - SVG 1.1 (Second Edition). https://www.w3.org/TR/SVG11/intro.html (2011) abgerufen am: 23.03.2016.

[4] W3C: Interactivity - SVG 1.1 (Second Edition). https://www.w3.org/TR/SVG11/interact.html (2011) abgerufen am: 23.03.2016.

[5] W3C: Paths - SVG 1.1 (Second Edition). https://www.w3.org/TR/SVG11/paths.html (2011) abgerufen am: 23.03.2016.

[6] MEDIA ENGINEERING: SVG Koordinaten. http://www.mediaevent.de/tutorial/svg-viewbox-koordinaten.html (2014) abgerufen am: 30.03.2016.

[7] Ecma International: The JSON Data Interchange Format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (2013) , Seite 1, abgerufen am: 31.03.2016.

[8] Ecma International: The JSON Data Interchange Format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (2013) , Seite 2, abgerufen am: 31.03.2016.

[9] Ecma International: The JSON Data Interchange Format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (2013) , Seite 3, abgerufen am: 31.03.2016.

[10] Apache Software Foundation: Apache™ Batik SVG Toolkit. https://xmlgraphics.apache.org/batik/ (2016) abgerufen am: 19.03.2016.

[11] Apache Software Foundation: Apache Batik API Specification 1.7. https://xmlgraphics.apache.org/batik/javadoc/ (2008) abgerufen am: 19.03.2016.

[12] Oracle Corporation: SVG Salamander. https://svgsalamander.java.net/ (2014) abgerufen am: 01.04.2016.

[13] Object Refinery Limited: JFreeSVG. http://www.jfree.org/jfreesvg/ (2015) abgerufen am: 01.04.2016.

[14] McCormack, C.: Using the Apache Batik toolkit for client- and server-side SVG processing. http://mcc.id.au/2007/09/batik-course/ (2007) , Folien 7-9, abgerufen am: 19.03.2016.

[15] McCormack, C.: Using the Apache Batik toolkit for client- and server-side SVG processing. http://mcc.id.au/2007/09/batik-course/ (2007) , Folien 14-16, abgerufen am: 19.03.2016.

[16] Oracle Corporation: AffineTransform, JavaDoc. https://docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html (2016) abgerufen am: 16.03.2016.

[17] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using vital sensors in mobile healthcare business applications: Challenges, examples, lessons learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps. (2013) 509–518

[18] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards process-driven mobile data collection applications: Requirements, challenges, lessons learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[19] Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Location-based mobile augmented reality applications: Challenges, examples, lessons learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 383–394

[20] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-driven data collection with smart mobile devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[21] Schickler, M., Pryss, R., Schobel, J., Reichert, M.: An engine enabling location-based mobile augmented reality applications. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 363–378

[22] Schickler, M., Reichert, M., Pryss, R., Schobel, J., Schlee, W., Langguth, B.: Entwicklung mobiler Apps: Konzepte, Anwendungsbausteine und Werkzeuge im Business und E-Health. Springer-Verlag (2015)

Abbildungsverzeichnis

Name: Maximilian Grabscheid Matrikelnummer: 791745

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Maximilian Grabscheid