

# MASTERARBEIT

im Labor für medizinische Bildverarbeitung, Algorithmen und Krankenhaus IT

zur Erlangung des akademischen Grades  
**Master of Science (M. Sc.)**

---

**Entwicklung einer modularen und erweiterbaren Anwendung  
zur medizinischen Bildverarbeitung**

---

**eingereicht von**

Rudolf Franz Siegfried Korb, 790060

**Erstprüfer**

Prof., Dr. Holger Timinger

**Zweitprüferin**

Prof., Dr. Gudrun Schiedermeier

**Abgabetermin**

02.03.2014



# Inhaltsverzeichnis

<b>Kapitelübersicht</b>	<b>1</b>
<b>1. Einleitung</b>	<b>2</b>
1.1. Der sechste Kondratieff-Zyklus . . . . .	2
1.2. Wachstumsmarkt Gesundheit . . . . .	4
1.3. Der Studiengang Biomedizinische Technik . . . . .	5
1.4. Das Labor für medizinische Bildverarbeitung, Algorithmen und Krankenhaus IT . . . . .	5
<b>I. Anforderungen und Theoretische Grundlagen</b>	<b>7</b>
<b>2. Anforderungen an eine modulare und erweiterbare Bildverarbeitungssoftware</b>	<b>8</b>
2.1. Einführung . . . . .	8
2.2. Nichtfunktionale Anforderungen . . . . .	8
2.3. Funktionale Anforderungen . . . . .	9
2.4. Evaluierung bestehender Software . . . . .	10
2.5. Slicer 3D . . . . .	11
2.6. ImageJ . . . . .	12
2.7. Ein Vergleich der verfügbaren Anwendung mit den Anforderungen der Stakeholder . . . . .	13
<b>3. Grundlagen medizinischer Daten- und Bildformate</b>	<b>15</b>
3.1. DICOM . . . . .	15
3.1.1. Die Dicom Information Object Definitionen . . . . .	16
3.1.2. Von Patientendaten der realen Welt zu digitalen DICOM-Objekten	19
3.1.3. Tags in Datenelementen . . . . .	20
3.1.4. VR - Value Representation . . . . .	21

3.1.5. VM - Value Multiplicity . . . . .	21
3.2. DICOM Pixeldaten und Bildformate . . . . .	21
3.2.1. Kodierung der Pixel im Speicherabbild einer DICOM-Objekts . . . . .	21
3.2.2. Grauwertbilder . . . . .	24
3.2.3. Farbbilder . . . . .	25
3.3. 3D Bilddaten . . . . .	26
3.4. Bilder mit zeitlicher Abhängigkeit . . . . .	27
<b>4. Grundlagen zu Entwurfsmustern der Softwareentwicklung</b>	<b>29</b>
4.1. Strukturmuster . . . . .	30
4.1.1. Adapter . . . . .	30
4.2. Verhaltensmuster . . . . .	30
4.2.1. Observer . . . . .	31
4.2.2. Schablonenmethode . . . . .	31
4.3. Erzeugungsmuster . . . . .	32
4.3.1. Fabrik Methode . . . . .	32
4.3.2. Singleton . . . . .	33
4.4. Das Architekturmuster Plug-in . . . . .	35
<b>II. Entwicklung des Java Medical Imaging Toolkit</b>	<b>36</b>
<b>5. Softwarearchitektur des Java Medical Imaging Toolkit</b>	<b>37</b>
5.1. Die Eclipse Rich Client Platform . . . . .	37
5.2. Das Eclipse Application Model . . . . .	39
5.2.1. Visuelle Komponenten . . . . .	39
5.2.2. Steuernde Komponenten . . . . .	40
5.3. Die Benutzeroberfläche von jMediKit . . . . .	42
5.4. Erweiterbarkeit der Grundstruktur . . . . .	42
5.5. Modulare Werkzeuge . . . . .	43
5.6. Die Plug-in Architektur . . . . .	44
5.6.1. Plug-in als Grundstruktur . . . . .	45
5.6.2. Erweiterung mit der Schablonenmethode . . . . .	46
5.6.3. Singleton als Plug-in Manager . . . . .	46
5.7. Externe Bibliotheken . . . . .	47
5.7.1. dcm4che . . . . .	47

5.7.2. Simple ITK . . . . .	48
5.7.3. Der Adapter zur Auflösung von Abhängigkeiten . . . . .	48
5.8. Die Architektur der Bilddaten . . . . .	49
5.8.1. Die einfache Fabrik . . . . .	50
5.8.2. Struktur der Bilder im ImageViewPart . . . . .	51
<b>6. Implementierung</b>	<b>54</b>
6.1. Implementierung der DICOM-Objekte . . . . .	54
6.2. Der DicomBrowser . . . . .	55
6.2.1. Die Baumstruktur . . . . .	55
6.3. Repräsentation der Pixeldaten . . . . .	57
6.3.1. Räumliche Sortierung der Bilddaten . . . . .	59
6.4. Zeichnen der Bilddaten mit dem DicomCanvas . . . . .	61
6.4.1. Implementierung der Werkzeuge . . . . .	62
6.5. Das Utility-Fenster . . . . .	62
6.5.1. Debugging mit dem ConsoleView . . . . .	62
6.5.2. DICOM-Objekte über DicomTagView ausgeben . . . . .	62
6.6. Implementierung des Hauptmenüs und der Werkzeugleiste . . . . .	62
6.6.1. Das Hauptmenü . . . . .	62
6.6.2. Die Werkzeugleiste . . . . .	62
<b>7. Entwicklung von Erweiterungen</b>	<b>63</b>
<b>Literaturverzeichnis</b>	<b>I</b>
<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>V</b>
<b>III. Anhang</b>	<b>VI</b>
<b>A. Darstellung der DICOM-Elemente im Speicher</b>	<b>VII</b>
A.1. Explizite VR mit [ OB   OW   OF   SQ   UT   UN ] . . . . .	VII
A.2. Explizite VR . . . . .	VII
A.3. Implizite VR . . . . .	VII

<b>B. Installation der Eclipse e4 Umgebung</b>	<b>XI</b>
B.1. Eclipse . . . . .	XI
B.2. Eclipse e4 Tools . . . . .	XII
B.3. Import der jMediKit Projektdateien . . . . .	XIV

Einleitung	2	<b>1.</b>
Anforderungen an das zu entwickelnde Programm	8	<b>2.</b>
Grundlagen der medizinischen Bildverarbeitung	15	<b>3.</b>

# 1. Einleitung

Ziel dieser Abschlussarbeit ist die Entwicklung einer Software zur medizinischen Bildverarbeitung. Eine modulare und flexible Architektur soll eine leichte Erweiterbarkeit sowohl für Entwickler (Erstellen eigener Werkzeuge) als auch den Anwender (Integration selbst implementierter Bildverarbeitungsalgorithmen) ermöglichen.

Das Programm soll für Forschung und Lehre im Labor für medizinische Bildverarbeitung, Algorithmen und Krankenhaus IT eingesetzt werden. Nach einer Vorstellung des Labors und dem zugehörigen Studiengang der biomedizinischen Technik erfolgt die Ermittlung der Anforderungen der Software. Aufbauend werden die Grundlagen medizinischer Daten- und Bildformate dargestellt. Der Fokus dieses Grundlagenkapitels liegt auf dem DICOM-Standard und den Eigenschaften der medizinischen Bilddaten. Die folgenden Kapitel erläutern die Softwarearchitektur sowie die Implementierung. Praktische Anwendungsbeispiele schließen die technischen Aspekte der Arbeit ab. Die folgende Diskussion zeigt Erweiterungsmöglichkeiten für die zukünftige Entwicklung auf.

## 1.1. Der sechste Kondratieff-Zyklus

Im Jahr 1926 veröffentlichte der Wirtschaftswissenschaftler Nikolai D. Kondratieff (\* 1892, †1938) die Theorie „Die Langen Wellen der Konjunktur“ [HK11]. Leo Nefiodow erweiterte 2006 die Theorie, damit die Entwicklung des 20. Jahrhunderts einfließen konnte.

Kondratieff zeigte, dass sich die gesellschaftliche Wandlung nicht willkürlich vollzog. Seit der Industrialisierung Mitte des 18. Jahrhunderts stand der Wohlstand der Gesellschaft in direkter Beziehung zu besonderen Erfindungen. Er betrachtete die Phasen des Wohlstandes und die direkt folgende Wirtschaftskrise und entdeckte die später nach ihm benannten „Kondratieff-Zyklen“ Wie in Abbildung 1.1 zu sehen ist, war die Dampfmaschine die erste Basisinnovation<sup>1</sup> und revolutionierte die Textilindustrie.[Wie12]

Diese Erfindung gilt als Beginn des ersten Kondratieff-Zyklus. Vor dem maschinellen

<sup>1</sup>Basisinnovationen müssen nach Nefiodow vier Eigenschaften erfüllen: Entstehung eines neuen Marktes mit vielen Arbeitsplätzen; Innovation bestimmt den Zyklus; Basisinnovationen haben einen Zyklus von 40 - 60 Jahren; Sie bestimmen die Entwicklungsrichtung

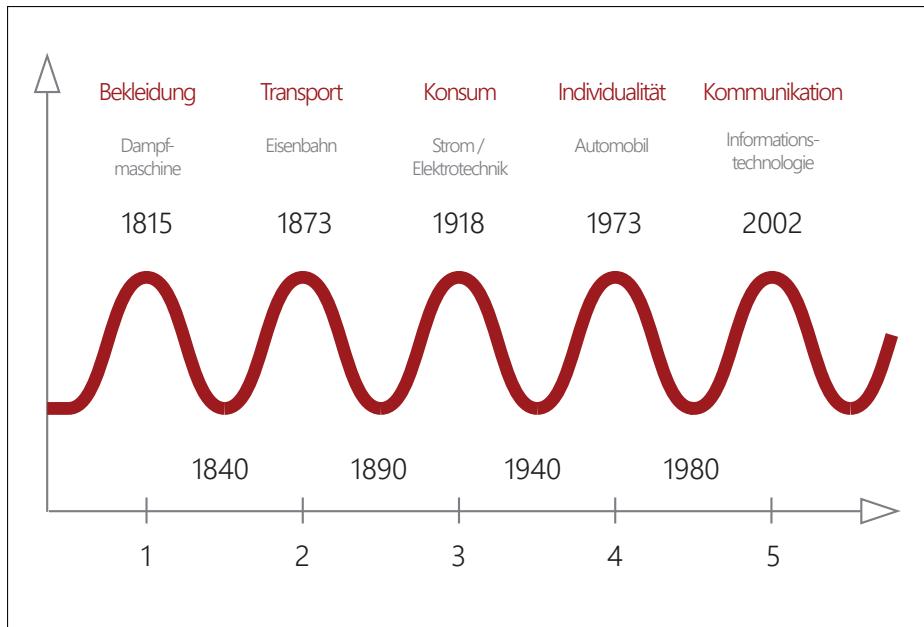


Abbildung 1.1.: Kondratieff-Zyklen

Betrieb wurden Spinnräder noch manuell bedient und Kleidung war teuer. Die dampfbetriebenen Webstühle steigerten die Effizienz um das 200-fache. In den 20er Jahren stagnierte die Branche, da die Rohstoffbeschaffung und Warenverteilung das Maximum der Effizienz erreicht hatte. Mit der Erfindung der Eisenbahn gelang der Übergang vom ersten in den zweiten Zyklus. In den folgenden Jahren konnte nun das Bedürfnis nach verbesserten Transportmöglichkeiten gestillt werden.

Dampfmaschine, Eisenbahn, Strom, Motor und der Mikrochip stehen alle für eine Basisinnovation, die zukünftige Gesellschaften geprägt haben. Im Lauf der Zeit verschwinden die Erfindungen aus dem Bewusstsein der Menschen und werden zu Gegenständen des Alltags. Motor und Mikrochip sind so stark im gesellschaftlichen Leben verankert, dass sie nicht mehr direkt wahrgenommen werden. Betrachtet man eine elektrische Zahnbürste, ist es selbstverständlich, dass die Energie aus dem Stromnetz bezogen und der Bürstenkopf von einem Motor angetrieben wird.

Das Jahr 2002 gilt als Höhepunkt des fünften Kondratieff-Zyklus und die Gesellschaft befindet sich gerade im Übergang zum Sechsten. Noch fehlt die aktuelle Basisinnovation und auch das zu nächste stillende Bedürfnis ist nach Individualität und Kommunikation (Abbildung 1.1) noch nicht bestimmt.

Nach Nefiodow [GN11] gibt es vier Möglichkeiten welcher Markt in Zukunft den sechsten

Kondratieff prägen wird:

- **Informationsmarkt**

Mobile Geräte und Soziale Netzwerke sind maßgebend für diesen Markt. So verhalf der Kurznachrichtendienst Twitter zum sogenannten „Arabischen Frühling“ durch die blitzschnelle Kommunikation über das Netz<sup>2</sup>.

- **Bio - und Nanotechnologie**

Die Erfindung des Mikroskops und die Entschlüsselung der DNA im Jahr 2000 gilt als Basisinnovation. Anfangs wurden die Erkenntnisse nur in Medizin und Pharmazie angewendet. Heute profitiert auch die Landwirtschaft und Lebensmittelindustrie davon.

- **Umwelttechnologie** Auch der Bereich Umwelttechnologie sorgte für einen Zuwachs an Arbeitsplätzen in Deutschland und Österreich. Zusätzlich siedelten sich Unternehmen besonders in ländlichen und strukturschwachen Gebieten an. [GN11, S. 107].

- **Gesundheit** Der Gesundheitsmarkt vereint technologische Komponenten wie die Medizintechnik und psychosoziale Gesundheit. Es erfolgt ein Wechsel vom heutigen „Krankheitswesen“ zum Gesundheitswesen, angefangen von der Burnout-Prophylaxe, Gesundheitstourismus zur Bionik und künstlichen computergesteuerten Prothesen.

## 1.2. Wachstumsmarkt Gesundheit

Nach Granig[GN11] ist der Gesundheitsbereich der derzeit am schnellsten wachsende Markt<sup>3</sup>. Die Bevölkerung ist gewillt in die eigene Gesundheit zu investieren und die Unternehmen positionieren sich im Gesundheitsbereich (Siemens beispielweise verstärkt sich im Bereich der Medizintechnik [GN11, S.111, 112]) Die Bio- und Nanotechnologie und die Medizintechnik ähneln sich in einigen Bereichen. Sowohl Siemon Cord [Cor07] als auch Granig [GN11, Seite 116 f] sprechen davon, dass der Markt sich nur gehemmt entwickeln kann. Grund dafür sind sowohl für die Nano- als auch Medizintechnik veraltete Vorschriften und auch ethische Hürden, die es zu überwinden gilt. Granig nennt hierzu unter anderem die Mensch-Computer-Einheit [GN11, S.117](computergesteuerte Prothesen und künstliche Ersatzteile).

<sup>2</sup><http://www.heise.de/tr/blog/artikel/Wie-funktioniert-die-Twitter-Revolution-1761481.html>  
aufgerufen am 06.01.2014

<sup>3</sup>Gemessen am Anteil der Branche am Bruttoinlandsprodukt

Die Entwicklung für den wachsenden Markt finden in den Unternehmen statt. Der Grundstein für Innovation wird jedoch bei den Studierenden der Hochschulen und Universitäten gelegt. Die Bildungseinrichtungen werden ein zentrales Standbein für den kommenden sechsten Kondratieff mit einem Schwerpunkt Bio-, Medizintechnik und Gesundheit sein.

### **1.3. Der Studiengang Biomedizinische Technik**

In einem Onlineartikel vom Februar 2012<sup>4</sup> veröffentlichte die Hochschule Landshut, dass ab dem Wintersemester 2012 der neue Bachelorstudiengang „Biomedizinische Technik“ angeboten wird. Auch der Artikel beschreibt, ähnlich wie Granig, die Medizintechnik als Wachstumsmarkt und bestätigt, auch durch die Einführung des Studiengangs, das gesellschaftlich gesteigerte Interesse am Gesundheitswesen.

Während des Studienverlaufs [Hoc13] erwerben die Studierenden vor allem im zweiten Studienabschnitt Kenntnisse im Bereich der Medizintechnik. Die Ausbildung behandelt unter Anderem bildgebenden Systeme, medizinische Bildverarbeitung und minimalinvasive Therapieverfahren.

Für die Ausbildung stehen Labore mit den benötigten Geräten zur Verfügung, um mit dem theoretischen Wissen praktisch zu experimentieren.

### **1.4. Das Labor für medizinische Bildverarbeitung, Algorithmen und Krankenhaus IT**

Das Labor erfüllt zwei Aspekte. Die Ausstattung steht für Forschungs- und Entwicklungsarbeiten zur Verfügung. Gemeinsame Forschungsprojekte von Hochschule und Krankenhäusern oder Unternehmen können hier umgesetzt werden. Für die Lehre soll Studierenden die Möglichkeit geboten werden, den Prozess der medizinischen Bildverarbeitung anschaulich und praxisnah zu erleben. Mittels Doppler-Ultraschallgerät können Bilddaten erzeugt und anschließend an das Picture Archiving and Communication System<sup>5</sup> (PACS) gesendet werden. Anschließend können Algorithmen zur Bildvorverarbeitung, Merkmalsextraktion oder auch Segmentierung implementiert und getestet werden.

<sup>4</sup><https://www.haw-landshut.de/aktuelles/news/news-archiv/news-detailansicht/article/neuer-studiengang-biomedizinische-technik-vielfältige-berufschancen.html>  
abgerufen am 10.01.2014

<sup>5</sup>Ein PACS dient als zentraler Bildspeicher, der über das Netzwerk angesprochen werden kann. Medizinische Geräte legen dort die Bilddaten ab, während die Software zu Betrachtung die Daten vom PACS holt

Medizinische Bildformate unterscheiden sich maßgeblich von allgemein bekannten Formaten wie JPEG oder Bitmaps, daher sind zur Betrachtung sogenannte DICOM-Viewer<sup>6</sup> notwendig. So enthalten medizinische Formate neben den reinen Bilddaten noch viele weitere Informationen, zum Beispiel Patientendetails oder die tatsächliche Lage des Patienten in der realen Welt zum Zeitpunkt der Aufnahme. Mit Hilfe der DICOM-Viewer lassen sich die erzeugten Bilder betrachten und grundlegende Operationen auf diesen anwenden (dazu zählt beispielsweise die Skalierung oder Verschiebung des Bildes). Komplexe Bildverarbeitungsalgorithmen können allerdings nicht ausgeführt oder selbst implementiert werden.

Für Forschung und Lehre wird eine Software benötigt, die sowohl die Grundfunktionen der Betrachtung liefert, als auch eine Schnittstelle zur eigenen Erweiterung zu Verfügung stellt.

---

<sup>6</sup>DICOM (Digital Imaging and Communications in Medicine) ist der heutige Standard der medizinischen Informationsverarbeitung und wird in den folgenden Kapiteln näher erläutert. Die Viewer ermöglichen die Betrachtung der Bilddaten

## Teil I.

# Anforderungen und Theoretische Grundlagen

## 2. Anforderungen an eine modulare und erweiterbare Bildverarbeitungssoftware

### 2.1. Einführung

In diesem Kapitel werden die Anforderungen an eine modulare und erweiterbare Bildverarbeitungssoftware gezeigt. In mehreren Besprechungen der Stakeholder<sup>1</sup> wurden die Anforderungen ermittelt. Diese werden in die beiden Bereiche *Nichtfunktionale Anforderungen* und *Funktionale Anforderungen* eingeteilt. Nach einer Evaluierung bestehender Anwendungen wird entschieden, ob eine eigene Software implementiert werden soll.

### 2.2. Nichtfunktionale Anforderungen

Nach Balzert [Bal11, 9] beschreiben nichtfunktionale Anforderungen, *wie* ein System arbeitet. Sie üben besonderen Einfluss auf die Softwarearchitektur eines Systems aus. Während der Besprechungen konnten folgende nichtfunktionale Anforderungen ermittelt werden.

- **Modularer Aufbau des Systems(1)**

Die Software soll in den Grundfunktionen erweiterbar sein, die nach einem neuen Build-Prozess zur Verfügung stehen. So soll es möglich sein, neue Bedienelemente der Benutzeroberfläche und neue Funktionen der Anwendung hinzuzufügen.

- **Erweiterbarkeit durch den Anwender(2)**

Anwender sollen die Möglichkeit haben, das Programm mit selbst programmierten Algorithmen und Plug-ins zu erweitern. Die eigene Implementierung von Bildverarbeitungsprozessen ist essentiell im Bereich der Lehre.

---

<sup>1</sup>Ein Stakeholder beschreibt einen Teilnehmer eines Projektes.

- **Die Programmiersprache Java(3)**

Die Studierenden des Studiengangs sollen im Modul „medizinische Bildverarbeitung“ Ihre Kompetenzen im Bereich der Programmierung mit Java erweitern. Dadurch soll die Anwendung in Java umgesetzt und erweiterbar sein.

- **Externe Bildverarbeitungsbibliotheken(4)**

Algorithmen in der Bildverarbeitung sind oft komplex und umfangreich. Nicht jeder benötigte Verarbeitungsprozess eignet sich zum selbst implementieren (sowohl im Lehr- als auch Forschungsbereich). Durch den Einsatz von Bibliotheken wird ein grundlegender Satz an Algorithmen vorgegeben, auf den der Benutzer zurückgreifen und in den Plug-ins verwenden kann.

- **Verarbeitung medizinische Daten mit Hilfe des DICOM-Standards(5)**

Medizinische Bilddaten besitzen neben den rohen Pixeldaten noch eine Vielzahl zusätzlicher Information, wie Patientendaten oder Seriennummern der Aufnahmen und benötigen eine spezielle Verarbeitung. Anders als übliche Grauwertbilder besitzen DICOM-Daten unter Anderem nicht 255 sondern bis zu  $2^{16}$  verschiedene Grauwerte.

## 2.3. Funktionale Anforderungen

Diese Art der Anforderungen legen fest, *was* das Programm leisten soll und bestimmen Funktionen und Verhalten [Bal11, 9]. Sie überschneiden sich oder stehen in Beziehung zu den nichtfunktionalen Anforderungen.

- **Dynamische Parameterübergabe bei einer Plug-in-Ausführung(6)**

Aus der Anwendererweiterbarkeit ergibt sich eine funktionale Anforderung. Nicht immer können alle Eigenschaften und Werte der Algorithmen während der Implementierung vom Anwender bestimmt werden. Durch die Abhängigkeit von den zu bearbeitenden Bilddaten und Algorithmen muss die Möglichkeit geboten werden, Parameter während der Laufzeit der Anwendung zu bestimmen.

- **Manuelle Auswahl signifikanter Punkte zur Verarbeitung in Plug-ins(7)**

Zusätzlich zu den dynamischen Parametern müssen einzelne Bildpunkte interaktiv vom Benutzer ausgewählt und später von Algorithmen benutzt werden können. Manche Bildverarbeitungsprozesse benötigen beispielsweise sogenannte Saatpunkte die von Anwender manuell ausgewählt werden müssen.

- **Anwendung der Algorithmen im dreidimensionalen Raum(8)**

Eine Vielzahl an Bildaufnahmen liegen als dreidimensionaler Datensatz vor. Eine Bildreihe muss folglich zusätzlich zur xy-Ebene auch in z-Richtung zu bearbeiten sein.

- **Darstellung mehrerer Bilddatensätze(9)**

Das Programm soll mehrere dreidimensionale Bildreihen zur selben Zeit in verschiedenen Fenstern anzeigen können. Das erlaubt den Benutzern die Datensätze in Beziehung zueinander zu betrachten.

- **Darstellung aller Bildebenen(10)**

Ein dreidimensionaler Datensatz macht es möglich, nicht nur die Bilder der xy-Ebene (Axial) sondern auch die xz-(Coronal) sowie yz-Ebene(Sagittal) darzustellen. Zwischen den Ansichten soll der Benutzer wechseln können.

- **Dynamische Anzeige eines Punktes in allen Bildebenen(11)**

Sind drei gleiche Datensätze in drei verschiedenen Fenstern geöffnet sollen diese dynamisch reagieren, wenn der Benutzer einen Punkt eines Datensatzes auswählt. Angenommen ein Datensatz ist in axialer, coronaler und sagittaler Ansicht geöffnet. Wählt der Nutzer einen Bildpunkt aus der axialen Ansicht, sollen die beiden übrigen geöffneten Fenster aktualisiert werden und den gleichen Punkt in ihrer jeweiligen Ansicht referenzieren.

- **Anzeige der dreidimensionalen Datensätze in  $(x, y, z)$  (12)**

Damit die Darstellung der verschiedenen Ebenen möglich wird, muss der gesamte dreidimensionale Datensatz zur Anzeige zur Verfügung stehen. Der Anwender soll durch die einzelnen Schichten der Bildreihen scrollen können.

## 2.4. Evaluierung bestehender Software

Frei verfügbare Software im medizinischen Bereich beschränkt sich oft die vorgegebenen Funktionen des Programms und bietet keine Möglichkeiten der Erweiterung. Zusätzlich liegt der Fokus auf der Darstellung der Patientenbilder und weniger an den Algorithmen zur Bildverarbeitung. Abbildung 2.1 zeigt den Screenshot des DicomViewers RadiAnt<sup>2</sup>. Die Bilder können einzeln, oder wie auf dem Bild zu sehen, im Bezug zueinander betrachtet werden. Die Werkzeugleiste oben ermöglicht die für DICOM-Bilder typischen

---

<sup>2</sup><http://www.radiantviewer.com/de/>

Operationen. Zwar gibt es auf dem Markt auch Open-Source Lösungen mit Schwerpunkt auf Bildverarbeitung, jedoch eignen sich diese nur bedingt für den Einsatz in der Lehre. Die Programme bieten eine Vielzahl an Funktionen, allerdings benötigt die Entwicklung von Erweiterungen einen erheblichen Zeitaufwand.

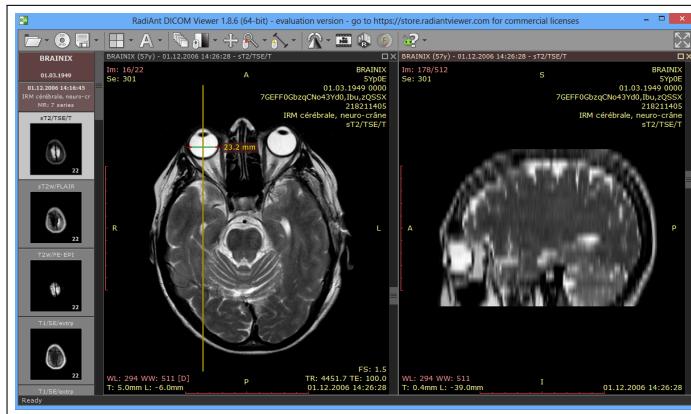


Abbildung 2.1.: RadiAnt - DicomViewer

## 2.5. Slicer 3D

Slicer 3D<sup>3</sup> (Abbildung 2.2) ist ein umfassendes Werkzeug für die medizinische Bildverarbeitung im zwei- und dreidimensionalen Raum. Die quelloffene Software bietet Möglichkeiten eigene Erweiterungen zu implementieren. Slicer verwendet als Bibliotheken unter anderem das Insight Toolkit und das Visualization Toolkit<sup>4</sup>. Die Erweiterungen werden in Python implementiert und dienen zur Anwendung der Bildverarbeitungsalgorithmen. Es fehlt die Möglichkeit der modularen Erweiterung bezüglich neuer Funktionen. Das Labor setzt in naher Zukunft ein PACS ein und die Software soll Möglichkeiten bieten, dieses System später zu integrieren, um DICOM-Daten über das Netzwerk zu beziehen.

<sup>3</sup><http://www.slicer.org>

<sup>4</sup>Insight Toolkit(ITK) und Visualization Toolkit (VTK) sind umfassende in C++ entwickelte Programmabibliotheken zur medizinischen Bildverarbeitung und Visualisierung.

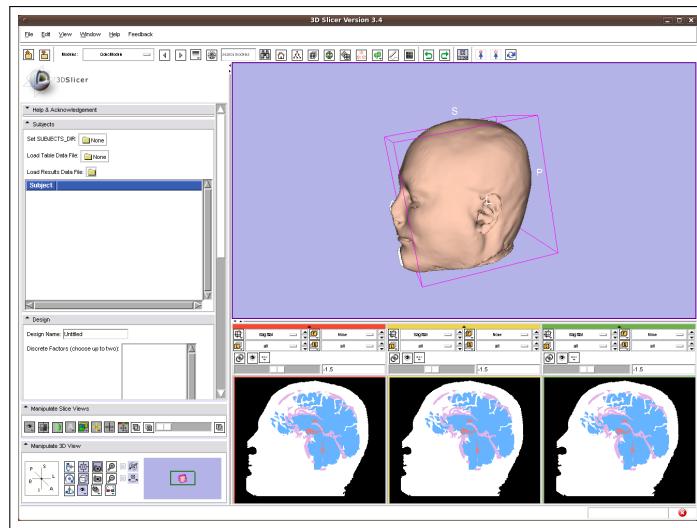


Abbildung 2.2.: Screenshot Slicer 3D

Quelle: <http://www.linuxlinks.com/portal/content/reviews/Health/Screenshot-3DSlicer.png> - abgerufen am 11.01.2014

## 2.6. ImageJ

ImageJ ist „State Of The Art“ im Bereich der Bildverarbeitung in Java<sup>5</sup>). Im Grundzustand liefert ImageJ die Standardfunktionen der Bildverarbeitung wie Abbildung 2.3 zeigt. Unter Anderem kann das Bildmaterial analysiert oder mit Filtern bearbeitet werden. ImageJ verarbeitet sowohl Grauwertbilder als auch Farbbilder in den gängigen Formaten wie PNG, JPEG und vielen Anderen. Mit Hilfe der „ImageStacks“ ist auch eine Bearbeitung im dreidimensionalen Bildraum möglich. Erweiterungen können schnell und zielgerichtet entwickelt werden. Im Modul „Bildverarbeitung“ der Fakultät Informatik wird ImageJ als Standard zum Bearbeiten der Übungsaufgaben verwendet. Für einen Einsatz im Studiengang Biomedizinische Technik fehlt allerdings die grundlegende Unterstützung von medizinischen Bilddaten im DICOM-Format. Die Funktionalität lässt sich über Plugins nachträglich hinzufügen, allerdings fehlt eine Bibliothek die bereits implementierte Algorithmen zur Verfügung stellt, sowie eine Verknüpfung von ImageJ-Klassen wie FloatProcessor oder ByteProcessor in Bildformate der Bibliotheken.

<sup>5</sup><http://rsbweb.nih.gov/ij/>

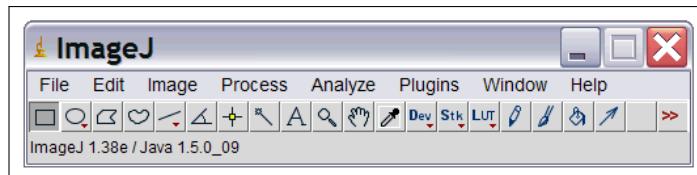


Abbildung 2.3.: Die Benutzeroberfläche von ImageJ

Quelle: <http://rsbweb.nih.gov/ij/features.html> - abgerufen am 11.01.2014

## 2.7. Ein Vergleich der verfügbaren Anwendung mit den Anforderungen der Stakeholder

Tabelle 2.1 zeigt eine Gegenüberstellung der evaluierten Software zu den Anforderungen. Slicer3D und ImageJ erfüllen einen hohen Grad davon, allerdings fehlen wichtige sogenannte harte<sup>6</sup> Anforderungen. Slicer3D lässt einen einfachen Ansatz zur modularen Erweiterung der Benutzeroberfläche und Anwendungsfunktionen vermissen. ImageJ hingegen bietet diese Variabilität. Allerdings ist die Verarbeitung von DICOM-Daten nur über ein Plug-in möglich und es fehlt eine Grundlage in Algorithmen zur medizinischen Bildverarbeitung. Zusätzlich zu den vorgestellten Anwendungen bietet auch MatLab<sup>7</sup> umfangreiche Bildverarbeitungskomponenten, unter anderem auch zur Verarbeitung von DICOM-Objekten an. Die hohen Lizenzkosten sprechen allerdings deutlich gegen einen Einsatz in der Lehre, da Anwendungen nur auf begrenzt vielen Maschinen installiert werden können und dadurch die Zahl der Arbeitsplätze für Studierende erheblich unnötig eingeschränkt ist. Aufgaben können nur im Labor bearbeitet werden, was voraussetzt, dass das Labor zu jeder Zeit geöffnet werden muss. Da in den vorgestellten Anwendungen keine Lösung verfügbar ist, die alle Voraussetzungen erfüllt, soll eine Software entwickelt werden, die für das Labor für medizinische Bildverarbeitung, Algorithmen und Krankenhaus IT die benötigten funktionalen und nichtfunktionalen Anforderungen erfüllt.

<sup>6</sup>Es wird zwischen harten und weichen Anforderungen unterschieden. Harte Anforderungen müssen erfüllt werden, während Weiche erfüllt werden können [Bal11, 9]

<sup>7</sup><http://www.mathworks.de/products/matlab/>

	<b>RadiAnt</b>	<b>Slicer3D</b>	<b>ImageJ</b>
modularer Aufbau (1)	✗	✗	✓
Java (3)	✗	✗	✓
multiple Datensätze anzeigen (9)	✓	✓	✓
Bildanzeige in $(x, y, z)$ (12)	✓	✓	✓
Ansicht der Ebenen $(xy, xz, yz)$ (10)	✓	✓	✗
Dynamische Anzeige der Punkte(11)	✓	✓	✗
DICOM (5)	✓	✓	✗
Plug-ins (2)	✗	✓	✓
Bibliotheken zur Bildverarbeitung (4)	n.A.	✓	✗
Dynamische Parameter (6)	n.A.	✓	✓
Manuelle Punktauswahl (7)	n.A.	✓	✓
Algorithmen in $(x, y, z)$ (8)	n.A.	✓	✓

Tabelle 2.1.: Gegenüberstellung der Anforderungen und verfügbarer freier Software

## 3. Grundlagen medizinischer Daten- und Bildformate

### 3.1. DICOM

Der Name DICOM steht für *Digital Imaging and COmmunication in Medicine*. Der Umgang mit diesem Standard ist essentieller Bestandteil der zu entwickelnden Software. Pianykh[Pia08, 1] beschreibt, dass DICOM nicht nur aus Pixel und deren zugehörigen Werten besteht. Wie der Name sagt, ist auch die Kommunikation fest im Standard verankert. Damit ist die Übertragung der Daten von medizinischen Geräten(Modalitäten) zum zentralen Speicher und deren Verteilung gemeint. Des weiteren spielt die dauerhafte Speicherung der digitalen Aufnahmen eine große Rolle. Daher wird im gleichen Zug mit DICOM immer ein PACS genannt. Das Akronym PACS bedeutet *Picture Archiving and Communication System* und besteht sowohl aus Hardware (Server, Speicherung) als auch Software(Verteilung und Kommunikation).

Abbildung 3.1 illustriert das Zusammenspiel des DICOM-Standards und dem zentralen Datenspeicher. Zuerst wird mittels der Modalitäten(z.B. mit dem Computertomographen oder einem Ultraschallgerät) die digitale Aufnahme erzeugt. Danach wird das Bild vom Gerät an das PACS gesendet. Hier werden die Aufnahmen und Patientendetails in die Datenbank und den Speicher abgelegt. Wird eine Aufnahme benötigt, können Clients Anfragen mit dem Patientennamen stellen und erhalten darauf die zugehörige Serie mit der digitalen Aufnahme.

DICOM<sup>1</sup> ist daher nicht nur ein einzelner Standard, sondern verknüpft die standardisierte

- Kommunikation,
- Erzeugung der Bilddaten,

<sup>1</sup>Unter <ftp://medical.nema.org/medical/dicom/> lässt sich der aktuelle Standard abrufen. Die Kapitel befinden sich im Ordner zum jeweiligen Jahr der Veröffentlichung. Aktuell sind die Dokumente von 2011.

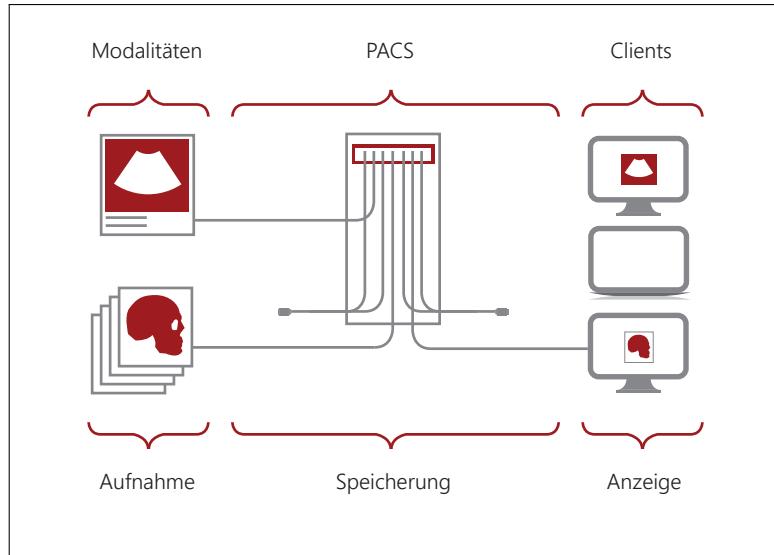


Abbildung 3.1.: Kommunikationsprozess von Aufnahme zur Verarbeitung

Vorlage für diese Darstellung ist die Grafik in [Pia08, Fig. 1]

- und Speicherung.

Im Rahmen dieser Abschlussarbeit wird zur Softwareentwicklung nur der Standard der Bilddaten verwendet. Daher wird in den Grundlagen nur auf die bildspezifischen Eigenchaften von DICOM eingegangen und Kommunikation sowie Speicherung zur Erläuterung vernachlässigt.

### 3.1.1. Die Dicom Information Object Definitionen

Bevor die Pixeldaten genauer betrachtet werden können, muss der prinzipielle Aufbau der Dicomobjekte beschrieben werden. Teil 3 des Standards[Nat11a, A.1.2] zeigt den relationalen Aufbau der Dicomobjekte. Vereinfacht können die elementaren Informationsobjekte in drei Teile aufgeteilt werden.

- **Patient**

Der Patient steht in der Hierarchie an oberster Stelle und ist die Grundlage für eine oder mehrere Studien(Study).

- **Study**

Study symbolisiert eine medizinische Studie. Eine Studie ist eine Sammlung von

mehreren Serien, die von Modalitäten wie CT und MR aufgezeichnet werden. Eine Studie ist exakt einem Patient zugeordnet.

- **Series**

Eine Serie ist eine Folge von Bildern, die von einer Modalität erzeugt wird. Die Aufnahmen eines CT werden einer Serie zugeordnet. Jede Serie gehört zu nur einer Studie. Eine Serie muss allerdings nicht zwingend aus Bildern bestehen. So können Serien auch aus Messdaten oder Registrierungsdaten zusammengesetzt werden [Nat11a, A.1.2].

- **Image, Real World Values**

Auf der unteren Hierarchiestufe stehen Objekte wie Bilddaten oder die Lage des Patienten im Raum während der Aufnahme. Ein Bild wird genau einer Serie zugeordnet.

Aus diesen vier elementaren Objekten ergibt sich folgende Informationsstruktur für Dicomobjekte, die in Abbildung 3.2 als Entity-Relationship-Modell<sup>2</sup> verdeutlicht wird.

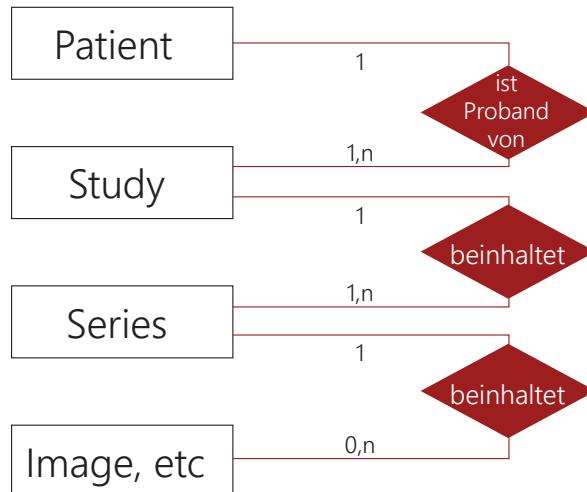


Abbildung 3.2.: Vereinfachte Darstellung der Informationsobjekthierarchie von Dicomelementen

Vorlage für diese Darstellung ist die Grafik in [Nat11a, A.1.2]

<sup>2</sup>Ein ER-Modell beschreibt die Beziehungen der Elemente zueinander. Dieser Diagrammtyp wird unter Anderem häufig beim Entwickeln der Struktur einer relationalen Datenbank verwendet

Der DICOM-Viewer OsiriX bietet auf der Herstellerseite<sup>3</sup> die Möglichkeit Testdaten zu beziehen. Betrachtet man die Repräsentation der Daten auf der Festplatte hält sich die Ordnerstruktur an obiges ER-Modell.

Abbildung 3.3 zeigt eine schematische Darstellung der Dateien. Die Beispieldaten von OsiriX bestehen aus einem Patient, dem eine Studie sowie zwei Serien à 100 Aufnahmen zugeordnet werden.

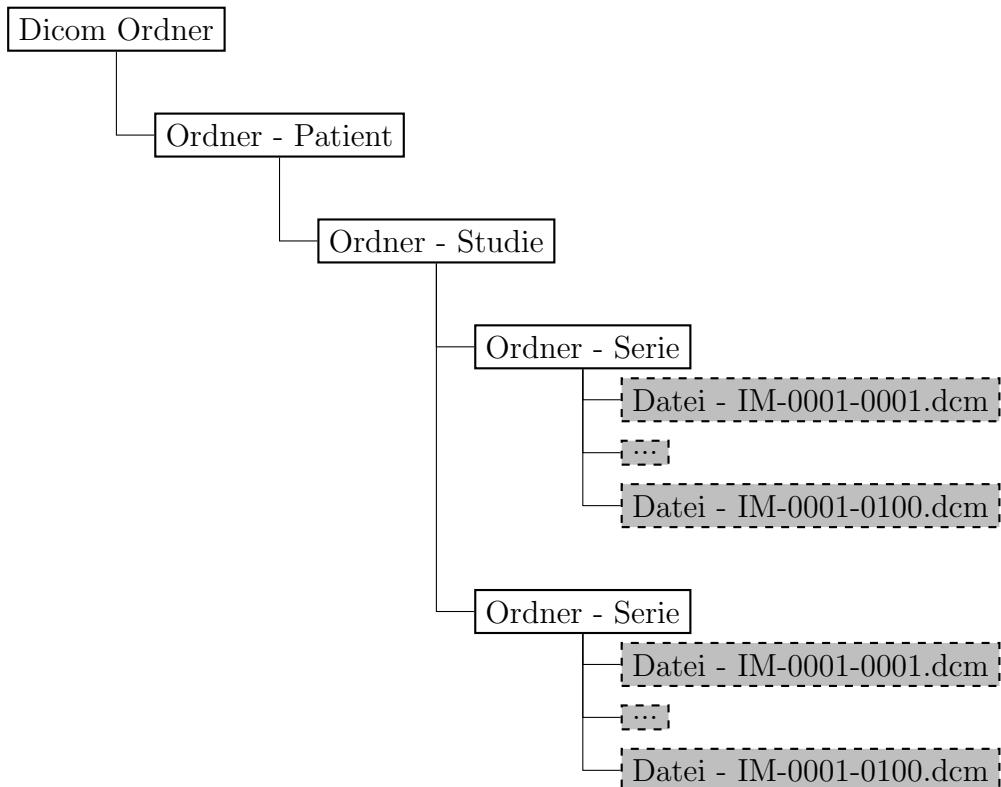


Abbildung 3.3.: Repräsentation der Information Objekte im Dateisystem

Bei näherer Betrachtung fällt auf, dass die Dateinamen beider Serien des Patienten identisch sind. Eine korrekte Zuordnung von DICOM-Dateien zur Serie ist daher nicht immer garantiert. Unabhängig von einer Repräsentation im Dateisystem oder Pfadangaben in der Datenbank eines PACS ist das Vertrauen auf Dateipfade unsicher, da über eine einfache Dateimanipulation die Zuordnung nicht mehr hergestellt werden kann.

Um eine zuverlässige Verknüpfung zu gewährleisten besitzt jeder Patient<sup>4</sup>, jede Studie und jede Serie eine eindeutige Identifikationsnummer. Diese Art der Informationen wird in den einzelnen Dateien mit Hilfe von Einträgen aus dem DICOM Data Dictionary[Nat11c] hin-

<sup>3</sup><http://www.osirix-viewer.com/datasets/>

<sup>4</sup>Die Identifikationsnummern von Patienten sind meist nur innerhalb einer Institution oder Krankenhauses einzigartig, da diese manuell vergeben werden können[Pia08, 5.6.2].

Tag	Tagname	VR <sup>a</sup>	Wert	VM <sup>b</sup>
(0010,0010)	PatientName	PN	John^Doe	1

Tabelle 3.1.: Repräsentation des Patientennamen als DICOM-Element

<sup>a</sup>ValueRepresentation - beschreibt den Werttyp, zum Beispiel numerischer Wert oder Zeichenkette<sup>b</sup>Value Multiplicity - wieviel Werte sind im DICOM-Element enthalten

terlegt. Die DICOM-Dateien beschreibt Pianykh [Pia08, S. 47] als eine Kopie im Speicher vom tatsächlichen DICOM-Objekt.

### 3.1.2. Von Patientendaten der realen Welt zu digitalen DICOM-Objekten

Wie der Name bereits andeutet ist das DICOM Data Dictionary vergleichbar mit einem Wörterbuch. Es enthält alle gültigen Elemente, die zur Beschreibung eines DICOM-Objekts verwendet werden können. Zusätzlich zu dem aus dem Standard bekannten Vokabular können Hersteller medizinischer Geräte ein eigenes Dictionary hinzufügen. Die proprietären Elemente können allerdings nicht standardisiert verarbeitet werden(vgl. [Pia08, S.45], da Software die diese Objekte verarbeitet, nichts von der Existenz dieser Elemente weiß). Aus diesen proprietären Elementen folgt, dass Bilddaten verschiedener Hersteller von Medizinprodukten untereinander nicht kompatibel sind und nicht ausgetauscht werden können.

Mit Hilfe des Wörterbuchs und den ca. 2000 enthaltenen Daten können nun Aussagen des wirklichen Lebens (vorausgesetzt die Aussage ist mit einem Element aus dem Wörterbuch darstellbar) in ein digitales DICOM-Objekt übersetzt werden.

Tabelle 3.1 zeigt das DICOM-Element für den Namen des Patienten aus dem Data Dictionary[Nat11c, S. 14].

Betrachtet man den folgenden Satz(vgl. [Pia08, S.46]), kann dieser in ein DICOM-Object, wie es Tabelle 3.2 darstellt, übersetzt werden:

„John Doe, männlich, geboren am 01. Januar 1970“

Wie aus dem Beispiel von Tabelle 3.1 zu erkennen, lässt sich ein DICOM-Element nochmals in atomare Teile aufgespalten. Folglich besteht ein Datenelement aus einem beschreibenden *Tag*, einer *VR*(*Value Representation*), einem Wert und der *VR* (*Value Multiplicity*). Das Element selbst nimmt eine von drei Darstellungsmöglichkeiten ein. Zusätz-

Tag (Gruppe, Element)	Tagname	VR	Wert	VM
(0010,0010)	PatientName	PN	John^Doe	1
(0010,0030)	PatientBirthDate	DA	19700101	1
(0010,0040)	PatientSex	CS	M	1
(0010,1010)	PatientAge	AS	44	1

Tabelle 3.2.: Das erzeugte DICOM-Objekt mit den Elementen zu Patientenname, Geburtsdatum, Geschlecht und Alter

lich liegt im Speicherabbild des Datenelements die Länge des Wertes<sup>5</sup>. Abhängig von der Transfersyntax<sup>6</sup> des DICOM-Objekts ist der VR-Teil optional. Die weiteren beiden Darstellungen unterscheiden sich in der Kodierung der benötigten Länge des Werts [Nat11b, 7.1]. Anhang A auf Seite VII zeigt wie Datenelemente im Speicher abgelegt werden und wie viel Speicherplatz pro Element reserviert werden muss.

### 3.1.3. Tags in Datenelementen

Ein DICOM-Element wie PatientName kann über ein Tag identifiziert werden. Ein Tag ist in einem DICOM-Objekt einzigartig und darf nur ein mal benutzt werden. Die numerische Darstellung hat die Form (gggg, eeee) wobei die hexadezimalen Ziffern *g* die Gruppe des DICOM-Elements beschreiben. Das *e* beschreibt das Element der Gruppe *g* definiert. Wie in Tabelle 3.2 zu sehen, steht 0010 für die Gruppe *Patient* und 0030 für das Element *PatientBirthDate* und ist der Patientengruppe zugehörig. Zusätzlich zu diesen Eigenschaften, kann bestimmt werden, ob der Ursprung eines DICOM-Elements im Standard- oder einem privaten Data Dictionary liegt. Eine gerade Gruppen-Ziffer zeigt, dass das Element Teil des Standards ist, während ungerade für proprietäre Elemente stehen<sup>7</sup>[Nat11b, 7.1]. Die Reihenfolge der Tags ist in numerischer Folge in aufsteigender Form sortiert. Fällt während des Einleseprozesses in einer Datei auf, dass die Reihenfolge nicht korrekt ist, kann die Integrität des DICOM-Objekts nicht gewährleistet werden und deutet zum Beispiel auf eine modifizierte Datei hin.

<sup>5</sup>John^Doe besitzt aufgrund der Zeichenmenge eine Länge von acht

<sup>6</sup>Unter der Transfersyntax verstehn man eine Menge an Kodierungsvorschriften von DICOM-Objekten[Nat11b, S.63 Section 10]. Zu diesen Vorschriften gehört zum Beispiel die Reihenfolge der Bytes im DICOM-Element oder die Komprimierung der Bilddaten

<sup>7</sup>Ausgenommen aus dieser Regel sind folgende Gruppen: (0000, eeee), (0002, eeee), (0004, eeee), (0006, eeee), (0001, eeee), (0003, eeee), (0005, eeee), (0007, eeee), (FFFF, eeee)

### 3.1.4. VR - Value Representation

Dieser Teil eines DICOM-Elements beschreibt den Typ und das Format des Wertes [Nat11b, 6.2]. Der Umfang an verschiedenen Value Representations reicht von Zeichenketten wie PersonName (PN) im Datenelement (0010,0010) PatientName über Datumsangaben (DA) bis zu numerischen Werten (FL) und Sequenzen (SQ). Eine vollständige Liste ist im Standard unter [Nat11b, Table 6.2.1] zu finden.

Bezüglich der Vollständigkeit soll erwähnt werden, dass ein Datenelement mit VR-Typ SQ wiederum ein DICOM-Object enthalten kann und dadurch eine Baumstruktur entsteht.

### 3.1.5. VM - Value Multiplicity

Value Multiplicity bestimmt die Anzahl an Werten, die in einem DICOM-Element enthalten sind. Die Werte werden durch einen Backslash \ voneinander getrennt. Der explizite Wert der Value Multiplicity kann aus dem Data Dictionary entnommen werden [Nat11b, 6.4]. Der Patientenname enthält einen Wert als Zeichenkette. Die Lage des Patienten im Raum wird mit Hilfe von drei Winkeln beschrieben. Dieses DICOM-Element hat eine Value Multiplicity von drei. Die Werte für die Winkel  $\alpha, \beta, \gamma$  werden als Zeichenkette 0\0\1 dargestellt.

## 3.2. DICOM Pixeldaten und Bildformate

Die Abschnitte 3 - 3.1.2 zeigen, dass DICOM mehr als ein Bildformat darstellt, ein essentieller Bestandteil bleiben jedoch die Pixel eines DICOM-Objekts (auch wenn diese nur optional vorhanden sein müssen, wie Abbildung 3.2 zeigt). Nach dem DICOM Data Dictionary gehören Bild-abhängige Informationen zur Gruppe (0028, eeee). Die Pixeldaten liegen im Bereich (7fe0, 0010) am Ende eines DICOM-Objekts.

Das bedeutet, dass die konkreten Werte der Pixel im gleichen DICOM-Objekt liegen und den selben Kodierungsrichtlinien der DICOM-Tags unterliegen.

### 3.2.1. Kodierung der Pixel im Speicherabbild einer DICOM-Objekts

Um die grundlegende Struktur der Pixel im DICOM-Objekt zu beschreiben sind drei Datenelemente notwendig: BitsAllocated, BitsStored und HighBit3.3. BitsAllocated beschreibt, wie viel Speicher für einen singulären Pixelwert reserviert wird. Mit Hilfe von Columns und Rows kann die Bilddimension bestimmt werden. Columns beschreibt die Breite, Rows die Höhe. Die Value Representation des Datenelements PixelData (7fe0,

0010) kann nach DICOM Data Dictionary entweder den Wert *OB* oder *OW* annehmen. *OB* bedeutet *Other Byte String* während *OW* für *Other Word String* steht. Nach Section 8.1 des Standards[Nat11b] besteht der Unterschied zwischen den beiden VRs darin, dass *OB* abhängig von der Byteordnung ist. Ob die Ordnung Little Endian oder Big Endian entspricht ist abhängig von der Transfersyntax des DICOM-Objekts. Grafik 3.4 zeigt die Kodierungsreihenfolge. Little Endian kodiert vom Least Significant Bit (LSB) zum Most Significant Bit (MSB). Big Endian verarbeitet die Byte in umgekehrter Reihenfolge. Ein Element des PixelData-Arrays (sowohl mit einer VR von *OB* als auch *OW*) fasst 16 Bit, was gleichzeitig die maximale Größe an allokiertem Speicher von BitsAllocated darstellt, denn nach dem Standard entspricht jede Zelle von PixelData genau einem Pixelwert [Nat11b, 8.1.1] und sowohl *OB* als auch *OW* belegen maximal 16-Bit im Speicher. Durch diese Limitierung wird deutlich, das die Grautwertbilder eine maximale Tiefe von 16-Bit erreichen können.

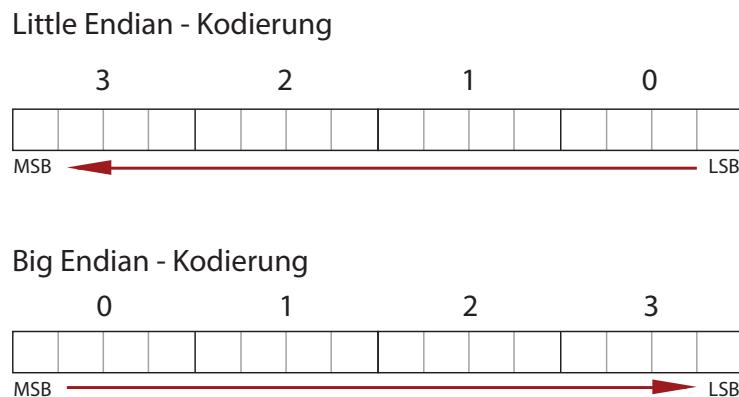


Abbildung 3.4.: Kodierungsreihenfolge von 4 Byte bei Little Endian- und Big Endian-Darstellung

BitsStored gibt darüber Auskunft, wie viel Bit pro Wert in Anspruch genommen werden. Schließlich gibt HighBit das in der Reihenfolge ranghöchste Bit von StoredBits an [Nat11b, 8.1.1].

Abbildung 3.5.1 verdeutlicht die Repräsentation eines Pixels im Datenelement PixelData. Die Darstellung entspricht einem eindimensionalen Array. Das erste Element ist das erste Pixel in der linken oberen Ecke, das letzte Element stellt den Pixel rechts unten dar. 3.5.2 und 3.5.3 zeigen die exakte Belegung an Bit bei BitsStored 16 und BitsStored 8. Der graue Hintergrund zeigt die allokierten Bit, während der rote Bereich den tatsächlich benutzten Speicher markiert. Der Pixelwert in Abbildung 3.5.3 nimmt den gesamten Speicherplatz

pro Pixel ein. Das schwarze Quadrat steht für das HighBit.

Das Intervall der Werte ist abhängig von der Anzahl an gespeicherten Bits. Hat das Element StoredBits den Wert 12 kann ein Pixel einen Wert aus dem Bereich von  $[0, 2^{12} - 1]$  annehmen. Entspricht StoredBits 8 ist das Intervall  $[0, 2^8 - 1]$ . Hier spricht man von einer Grauwerttiefe von 12 beziehungsweise 8 [Han00, 2.2].

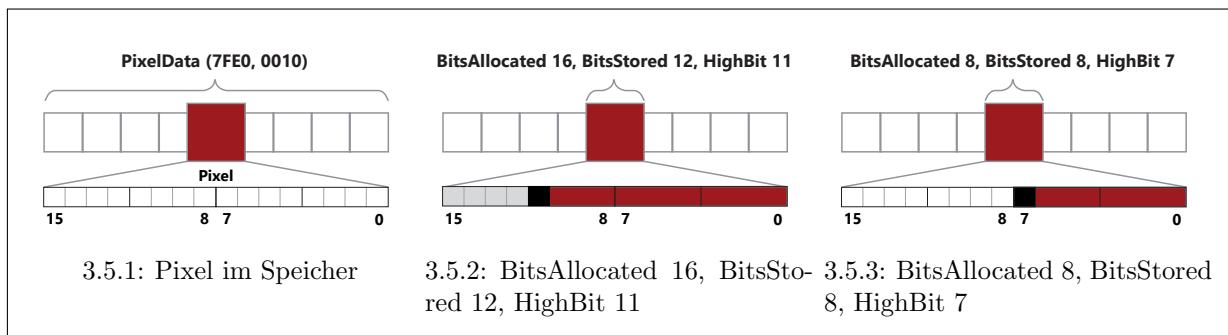


Abbildung 3.5.: Beispiele unterschiedlicher Speicherbelegung

Verschiedene Datenelemente aus dem DICOM-Standard liefern nähere Informationen zum Bildformat. So gibt das Element SamplesPerPixel (0028,0002) Auskunft darüber, wieviel Teile aus PixelData ein einzelnes Pixel repräsentieren. Hat Samples den Wert 1, so ist jedes Element aus PixelData genau ein Pixel. Daraus folgt, dass ein Grauwertbild vorliegt. 3 bedeutet, dass im DICOM-Objekt ein Farbbild mit den drei Kanälen rot, grün und blau liegt.

Die Werte der Pixel sind stark abhängig vom medizinischen Gerät, welches die Bilder aufzeichnet. Ein direkter Vergleich von Bildern, die von unterschiedlichen Geräten einer Modalität aufgezeichnet wurden ist daher nicht möglich. Um Gewebestrukturen von beispielsweise CT-Aufnahmen trotz dieser Abhängigkeiten patienten- und geräteübergreifend zu vergleichen, gibt es unter Anderem die Hounsfield Skala [Han00, 2.1.3]. Mit Hilfe der Datenelemente RescaleSlope und RescaleIntercept lassen sich die ursprünglichen Werte in brauch- und lesbare Pixelwerte konvertieren. RescaleType gibt die Skala an, mit der das Ergebnis interpretiert werden kann. Ein Umrechnung erfolgt mit der Formel aus [Nat11a, C.11-1b Seite 1168]

$$Output = m * SV + b \quad (3.1)$$

mit  $m = \text{RescaleSlope}$ ,  $b = \text{RescaleIntercept}$  und  $SV = \text{Pixelwert}$ .

Fettgewebe zum Beispiel nimmt nach der Hounsfield-Skala Werte zwischen 0 und -100 ein [BLT98, Abbildung 1.18 Seite 15]. Ob in einem DICOM-Objekt vorzeichenbehaftete-

Tag	Tagname	VR
(0028,0010)	Rows	US
(0028,0011)	Columns	US
(0028,0100)	BitsAllocated	US
(0028,0101)	BitsStored	US
(0028,0102)	HighBit	US

Tabelle 3.3.: Grundlegende Datenelemente für die digitale Repräsentation

te Werte vorhanden sind, sagt das Element PixelRepresentation. Eine 0 bedeutet kein Vorzeichen. Bei einem Wert von 1 können negative Werte enthalten sein.

### 3.2.2. Grauwertbilder

Für Bildverarbeitungsprozesse und Algorithmen bieten Grauwertbilder einige Vorteile im Vergleich zu Farbbildern. Der größte Unterschied liegt im Verhältnis zwischen Informationsgehalt zum benötigtem Speicherbedarf. Farben bieten bei Kantenübergängen oder Helligkeitsinformationen keinen Mehrwert. Das führt unter Anderem dazu, dass Industrie oder auch die medizinische Bildverarbeitung vorwiegend auf dieses Format zurückgreifen. Viele medizinische Geräte liefern bei der Bildaufnahme reine Intensitätswerte und beinhalten dadurch keine Farbinformationen.

Eine Grauwerttiefe von 8 Bit wird überlicherweise in der industriellen Bildverarbeitung eingesetzt. Das entspricht dem Intervall von [0, 255] und den Werten, die mit handelsüblichen Monitoren darstellbar sind. Medizinische Bilddaten können, wie in Abschnitt 3.2.1 beschrieben, eine Tiefe von bis zu 16 Bit annehmen und Grauwerte aus dem Bereich [0, 65535] repräsentieren, da eine 8-Bit-Repräsentation eine zu geringe Genauigkeit liefert. Spezielle kostspielige Befundungsmonitore sind dazu in der Lage diese Spanne an Grauwerten darzustellen.

#### Fensterung von Grauwerten

Wie bereits in 3.2.2 beschrieben, lassen sich auf üblichen Monitoren nicht alle Grauwerte zur gleichen Zeit darstellen. Dadurch müssen die maximal 65535 verschiedenen Grauwerte auf 255 Werte abgebildet werden können. Dies wird durch die in der Radiologie verwendete Fensterungstechnik möglich [Han00, Kapitel 8, Seite 249]. Es wird ein Fensterzentrum und eine Fensterbreite gewählt. Alle Werte innerhalb dieses Intervalls werden zwischen 0 und 255 umgerechnet. Abbildung 3.7.1 verdeutlicht dieses Prinzip. Ein CT-Bild mit einer Tiefe

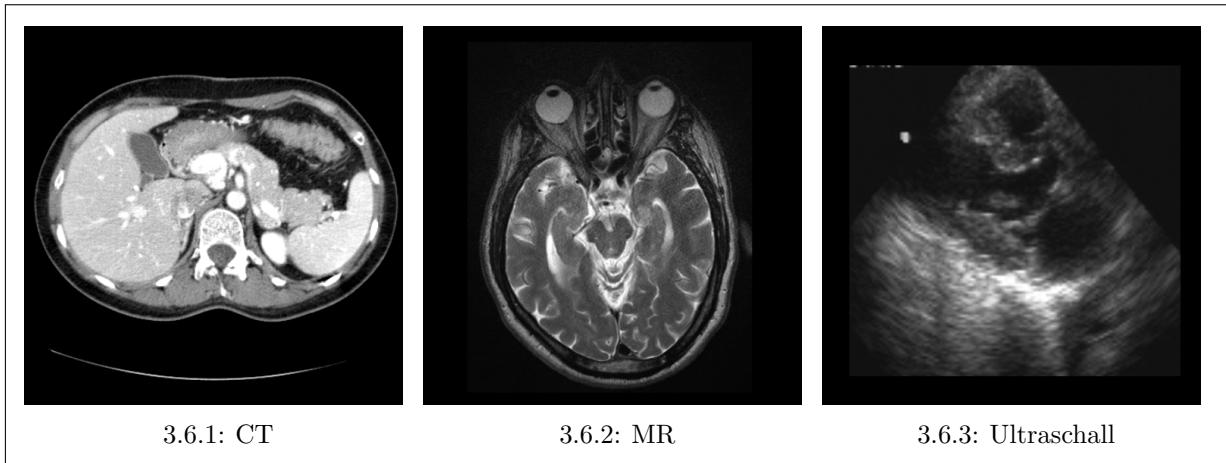


Abbildung 3.6.: Verschieden Graustufenbilder

Die Beispieldaten stammen von <http://www.osirix-viewer.com/datasets/> - abgerufen am 19.01.2014

von 12 Bit besitzt 4096 Grauwerte. Ein Zentrum von 2000 und eine Breite von 500 bildet alle Werte von 1750 bis 2250 auf 0 bis 255 ab. Ist ein Pixeldatum kleiner 1750 wird das Minimum 0 hinterlegt und ist der Wert größer 2250 bekommt das Pixel das Maximum 255. Im DICOM-Standard ist ein Algorithmus gegeben, um die Pixeldaten zu konvertieren[Nat11a, C.11.2.1.2].

---

Algorithmus 1: Berechne den Fensterungswert aus originalem Pixelwert

---

```

1:  $X \leftarrow \text{input}$  - tatsächlicher Pixelwert
2:  $Y \leftarrow \text{output}$  - konvertierter Wert zwischen 0 und 255
3:  $C \leftarrow \text{windowCenter}$ 
4:  $W \leftarrow \text{windowWidth}$ 
5: if  $X \leq C - 0.5 - (W - 1)/2$  then
6:    $Y = Y_{\min}$ 
7: else if  $X > C - 0.5 + (W - 1)/2$  then
8:    $Y = Y_{\max}$ 
9: else
10:   $Y = ((X - (C - 0.5))/(W - 1) + 0.5) * (Y_{\max} - Y_{\min}) + Y_{\min}$ 
11: end if
```

---

### 3.2.3. Farbbilder

Obwohl Grauwertbilder das am meisten verwendete Format in der medizinischen Bildverarbeitung darstellen, haben auch Farbbilder die verschiedensten Einsatzgebiete. So wird in der Dermatologie auf Farbdarstellungen zurückgegriffen um Hauterkrankungen

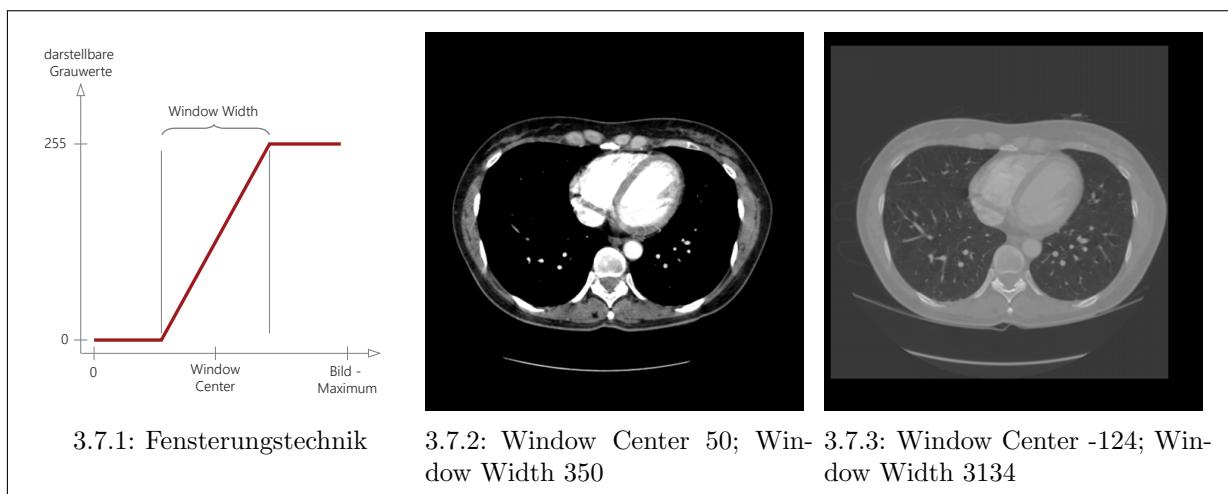


Abbildung 3.7.: Fensterungstechnik zur Darstellung medizinischer Bilddaten am handelsüblichen Monitor

Vorlage für die Grafik der Fensterungstechnik ist [Han00, Abbildung 8.1]; Die Beispieldaten stammen von <http://www.osirix-viewer.com/datasets/> - abgerufen am 19.01.2014

zu dokumentieren [Han00, 2.2.3.2]. Des weiteren kann mit Farbultraschallbildern die Fließrichtung und Geschwindigkeit des Blutes visualisiert werden und dienen zur Untersuchung von Venen und Arterien<sup>8</sup>.

Wie bereits beschrieben, ist das Datenelement SamplesPerPixel mit einem Wert von drei der Indikator, dass ein Farbbild vorliegt. Das bedeutet pro Pixel werden 3 Elemente von PixelData belegt mit je einem Element für den roten, grünen und blauen Farbkanal. Daraus resultiert der dreifach Speicherbedarf,  $\text{BitsAllocated} * \text{SamplesPerPixel}$  [Nat11a, C.7.6.3.1.1]. Der DICOM-Standard bietet zwei Möglichkeiten, wie diese Information im Speicher hinterlegt werden. Entweder werden die Pixelwerte fortlaufend gespeichert mit R1, G1, B1; R2, G2, B2; ...RN, GN, BN; oder nach dem Kanal R1, R2, ...RN; G1, G2, ...GN; B1, B2, ...BN [Nat11a, C.7.6.3.1.3]. Das Element dazu aus dem Data Dictionary heißt PlanarConfiguration(0028,0006). Abbildung 3.8 macht die beiden Schemata deutlich.

### 3.3. 3D Bilddaten

Sowohl die Grauwertdarstellungen, als auch Farbbilder wurden bisher nur als eine zweidimensionale Abbildung behandelt. Computertomographen, Magnetresonanztomographen

<sup>8</sup><http://www.diagnostikum-berlin.de/farbkodierte-duplexsonographie-fkds> - abgerufen am 19.01.2014

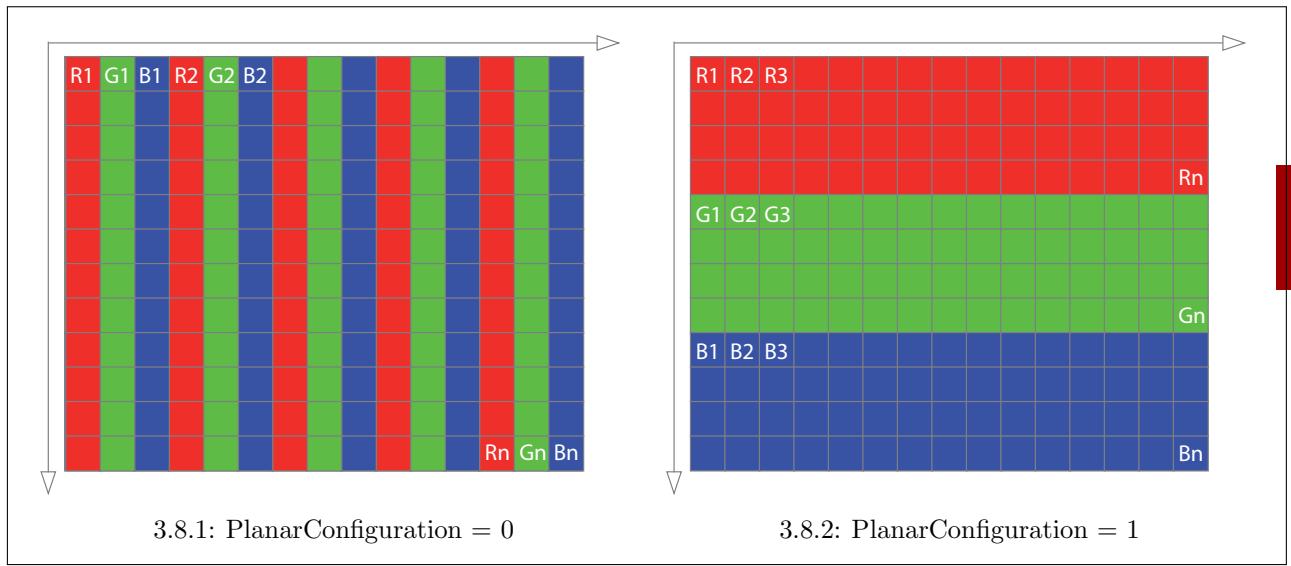


Abbildung 3.8.: Kodierung der RGB-Werte im Dataelement PixelData mit Hilfe der PlanarConfiguration

und auch 3D-Ultraschall sind in der Lage Schichten des menschlichen Körpers aufzunehmen. Die Menge der Schichtaufnahmen stellen eine Serie dar (vgl. Die Dicom Information Object Definitionen 16). Durch die Verbindung der einzelnen DICOM-Objekte wird der Pixelraum verlassen und der Voxelraum betreten. Ein Voxel ist die dreidimensionale Repräsentation eines Pixels, mit der Tiefe als zusätzliche Dimension zu Breite und Höhe.

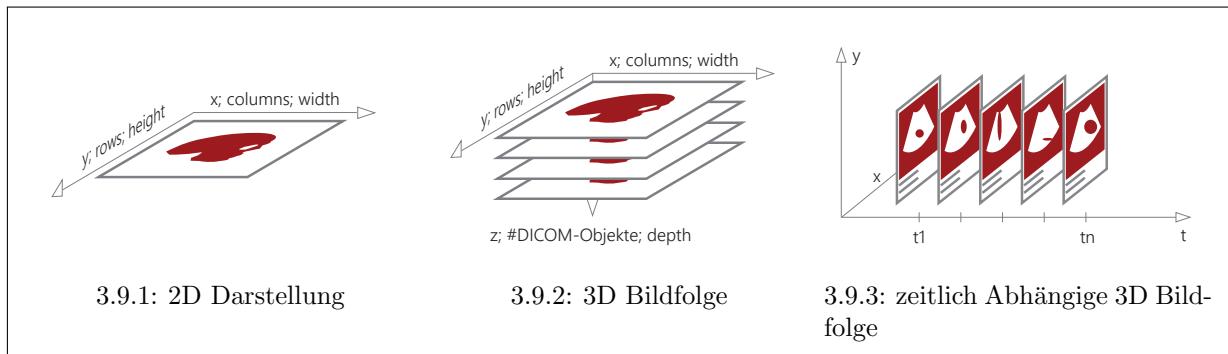


Abbildung 3.9.: Darstellung von 2- und 3-dimensionalen Bilddaten

## 3.4. Bilder mit zeitlicher Abhängigkeit

Auch Bildaufnahmen zu verschiedenen Zeitpunkten sind dreidimensionale Bildfolgen, wobei die Zeit die dritte Dimension darstellt [Han00, 2.2.5]. Häufige Einsatzgebiete für Be-

wegtbildfolgen ist die Endoskopie oder Sonographie. Die Abbildung 3.9.3 macht den Unterschied zwischen zeitlicher und räumlicher Dimension deutlich. Räumliche Darstellungen sind grundsätzlich in mehrere DICOM-Objekte und Dateien aufgeteilt. Zeitlich abhängige Daten sind in einem einzigen Objekt zusammengefasst. NumberOfFrames (0028,0008) gibt die Anzahl an unterschiedlichen Aufnahmen und Zeitpunkten an, die in PixelData enthalten sind.

## 4. Grundlagen zu Entwurfsmustern der Softwareentwicklung

Während der Entwicklung von Software stehen Programmierer oft vor ähnlichen Problemstellungen. Entwurfsmuster bieten hierfür verlässliche Lösungsvorschläge[GD13]. Muster werden unabhängig vom aktuellen Abschnitt des Entwicklungsprozesses eingesetzt. So gibt es Schablonen, die das Softwaresystem beschreiben, oder Muster, die auf Objektebene eingesetzt werden. Hierbei wird zwischen Architekturmustern und Entwurfsmustern unterschieden. Nach Goll und Dausmann [GD13, 3.1, 3.2] ist das Ziel der Entwurfsmuster, Lösungen wiederverwendbar zu gestalten und die Flexibilität der Software zu erhöhen. Muster verbessern unter anderem die beiden Eigenschaften der Verständlichkeit und Erweiterbarkeit. Im Rahmen der Arbeit werden vor allem objektorientierte Muster folgender Klassen verwendet:

- Strukturmuster
- Verhaltensmuster
- Erzeugungsmuster

Nach dieser ersten Klassifikation kann nochmals zwischen klassenbasierten und objektbasierten Mustern unterschieden werden. Der Unterschied besteht hauptsächlich darin, dass klassenbasierte Muster die Objekttypen während der Übersetzung festlegt. Bei objektbasierten Mustern können Objekte den Typ dynamisch zur Laufzeit ändern[GD13, 4.1].

Die im folgenden dargestellten Entwurfsmuster bilden nur einen kleinen Teil der verfügbaren Muster ab. Diese Auswahl wird durch die Nutzung in der zu entwickelnden Software besonders hervorgehoben und genauer erläutert. Die konkrete Umsetzung wird im Kapitel „5. Softwarearchitektur des Java Medical Imaging Toolkit“ detailliert dargestellt.

## 4.1. Strukturmuster

Strukturmuster bestimmen die Zusammensetzung der Klassen oder Objekte. Während dieser Abschlussarbeit wird im Speziellen auf das Muster *Adapter* eingegangen, um externe Bibliotheken mit internen Klassen zu verknüpfen.

### 4.1.1. Adapter

Der Adapter passt Schnittstellen an, um Zusammenarbeit inkompatibler Objekte und Klassen zu ermöglichen. Er wird vorwiegend eingesetzt wenn die Schnittstelle einer Komponente nicht zu der einer benötigten Klasse passt und somit nicht gemeinsam verwendet werden können[ES13, 5.1]. Ein Adapter aus der realen Welt ist gut mit dem Muster zu vergleichen. Ein Monitor mit VGA-Eingang(Client) benötigt eine Grafikkarte mit einem VGA-Ausgang zur Kommunikation. Hat die Karte jedoch nur eine DVI-Buchse(zu adaptierende Klasse) wird ein Adapter(Interface Adapter) benötigt, um das Signal umzuwandeln. Abbildung 4.1 zeigt das Schema des Entwurfsmusters.

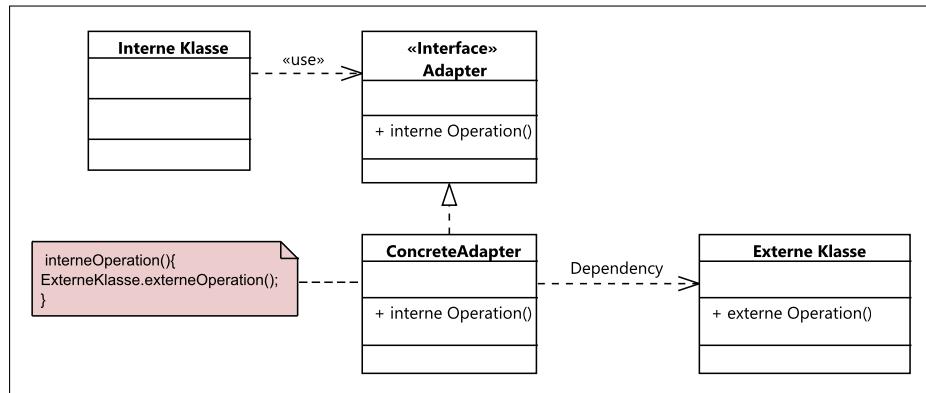


Abbildung 4.1.: UML Klassendiagramm zum Adapter

## 4.2. Verhaltensmuster

Die Interaktion zwischen Objekten kann mit Verhaltensmustern festgelegt werden. Die Anforderung aus Kapitel 2 zur Darstellungen verschiedener Ebenen erfordert eine Möglichkeit zur Kommunikation unter den Anzeigeflächen der DICOM-Objekte. Hier kommt das

Beobachtermuster zum Einsatz, um Veränderungen eines Objekts an andere Objekte zu melden.

#### 4.2.1. Observer

Wenn der Zustand mehrerer Objekte von einem anderen Objektzustand abhängt, kann man das Observer-Muster verwenden. So kann der Zustand abhängig vom beobachteten Objekt angepasst werden [ES13, 4.7]. Ein Abonnement einer Zeitung(ConcreteSubject) oder eines Newsletters(ConcreteSubject) ist ein reales Beispiel dieses Muster. Alle Abonnenten<sup>1</sup>(Observer) erhalten(update()) vom Herausgeber(Subject) automatisch(notifyObservers()) die neue Ausgabe eines Magazins oder Newsletter sobald dieses erscheint. Ein Abonnent muss sich bewusst für eine Anmeldung entscheiden(register()), um neue Ausgaben zu erhalten. Das Konzept verdeutlicht Abbildung 4.2.

4.

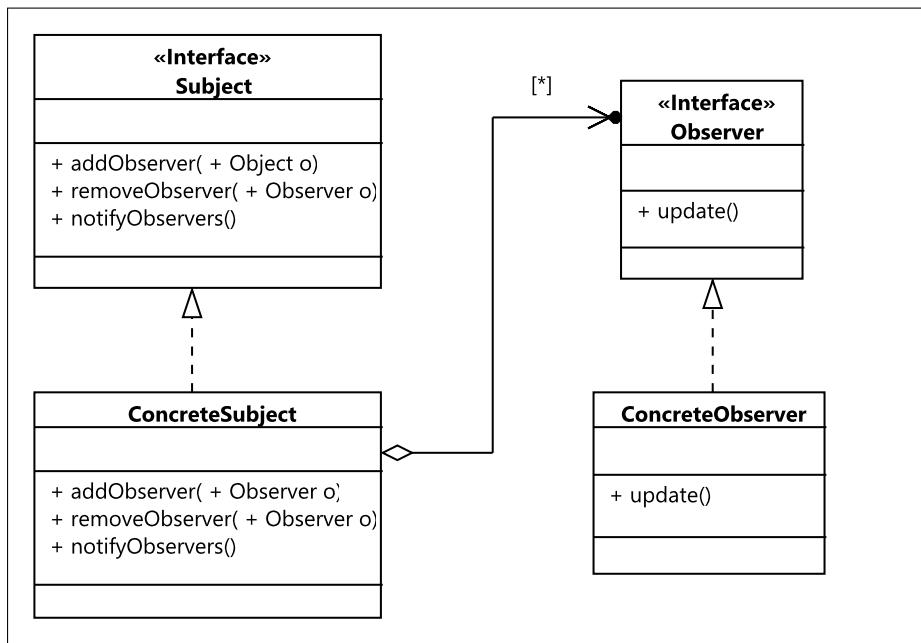


Abbildung 4.2.: UML Klassendiagramm zum Observer

#### 4.2.2. Schablonenmethode

Die Schablonenmethode wird eingesetzt, wenn Teile eines Algorithmus flexibel austauschbar sein sollen. Ein Rezept für Weihnachtsgebäck könnte zum Beispiellisting folgende

<sup>1</sup>Frau Müller und Herr Schmidt würden **ConcreteObserver** repräsentieren

Vorgehensweise wie in Listing 4.1 haben. Abbildung 4.3 zeigt das UML Diagramm dieses Entwurfsmusters. Eine Abstrakte Klasse definiert eine Vorgehensweise *templateMethod()*. Im Beispiel ist das die Klasse *Gebaeck*. Abgeleitete Klassen definieren die genaue Vorgehensweise des Algorithmus durch die implementierten abstrakten Methoden *Operation1()* - *Operation3()*( im Beispiel sind das die Methoden *vermengeZutaten()* - *backen()*). Die Backzeiten von Spekulatius und Keksen sind unterschiedlich. Durch die spezifische Umsetzung von *backen()* wird die Backzeit individuell angepasst. Das bedeutet die Vorgehensweise ist identisch, die Umsetzung kann spezifisch variieren.

```
1 public abstract class Gebaeck{  
2  
3     public void gebaeckBacken(){  
4         vermengeZutaten();  
5         verarbeiteZutatenZuTeig();  
6         rolleTeigAus();  
7         stecheFormenAus();  
8         backen();  
9     }  
10 }  
11 }
```

Listing 4.1: Beispiel zur Schablonenmethode

## 4.3. Erzeugungsmuster

Erzeugungsmuster übernehmen die Erstellung der Objekte zur Laufzeit. Wie beschrieben haben DICOM-Bilddaten eine Grauwerttiefe zwischen 8- und 16-Bit. Daher kann erst zur Laufzeit entschieden werden, welches Bildobjekt angelegt werden kann. Ein Fabrikmuster übernimmt diese Aufgabe.

### 4.3.1. Fabrik Methode

Dieses Entwurfsmuster bietet eine Schnittstelle(Fabrik) zur Erzeugung von Objekten(Produkte). Stehen die konkreten Objekttypen erst zur Laufzeit des Programms fest, kann dieses Erzeugungsmuster eingesetzt werden, um Konstruktion und Repräsentation zu trennen[ES13, 3.3].

Abbildung 4.4 zeigt den Aufbau in einem UML-Diagramm. Als Beispiel kann man einen

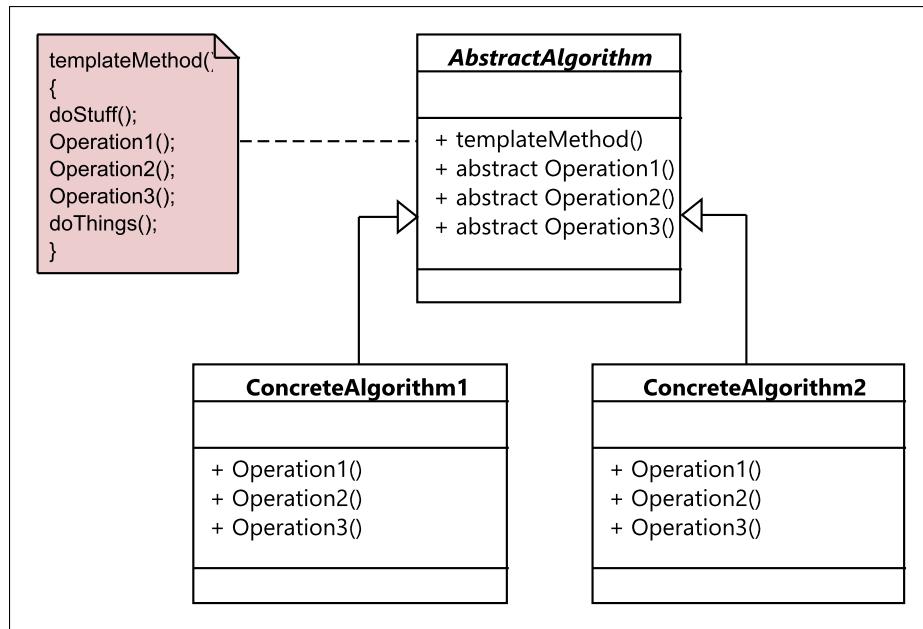


Abbildung 4.3.: UML Klassendiagramm der Schablonenmethode

Handwerker(Client) mit seinem Lehrling(Factory) betrachten, der Schrauben(*Product*→*Schraube8mm*, *Schraube10mm*) zureichen muss. Der Handwerker muss nicht selbst von der Leiter steigen und im Werkzeugkasten nach der passende Schraube suchen. Er bittet den Lehrling zum Beispiel eine Schraube8mm zu bringen. Das Wissen über konkrete Schrauben stellt eine konkrete Fabrik dar. So ist der Lehrlich auch in der Lage andere Werkzeugkategorien zu verwalten. Jedes Werkzeug spezialisiert von *AbstractProduct* und das Wissen über die Kategorie von *AbstractFactory*.

### 4.3.2. Singleton

Ein Singleton-Entwurfsmuster wird eingesetzt, wenn es nur eine Instanz einer Klasse geben darf[ES13, 3.4]. Das ist unter Anderem bei Konfigurationsklassen von Software sinnvoll, da eine Manipulation von unterschiedlichen Quellen ausgeschlossen werden kann. das Codelisting 4.2 zeigt eine primitive Umsetzung in Java, die Intention eines Singletons wird allerdings deutlich. Durch die Sichtbarkeit *private* des Konstruktors wird eine externe Instanziierung unterbunden. Die einzige Instanz der Klasse ist das private statische Datenelement *Singleton s*. Zugriff darauf erlaubt nur die statische Methode *getInstance()*.

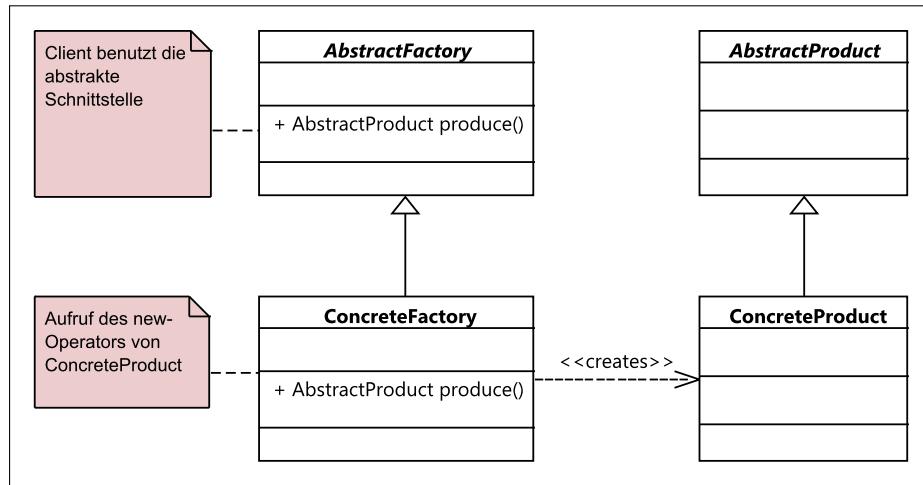


Abbildung 4.4.: UML Klassendiagramm zur Fabrik Methode

```

1 public class Singleton{
2
3     private static Singleton s = new Singleton();
4
5     private Singleton(){
6         //doThings();
7     }
8
9     public static Singleton getInstance(){
10        return s;
11    }
12 }
```

Listing 4.2: Implementierung des Singelton-Musters in Java

## 4.4. Das Architekturmuster Plug-in

Wird eine Software entwickelt, sollte das Open-Closed-Prinzip angewendet werden. Dieses besagt, dass Software sowohl offen, als auch geschlossen sein soll [GD13, 1.8]. Bei objektorientierten Programmiersprachen ist die Vererbung ein Beispiel, die den Widerspruch erläutert. Abgeleitete Klassen *erweitern* die Fähigkeiten der Superklasse, die wiederum *nicht verändert* werden. Die Superklasse ist offen für Erweiterung und geschlossen für Veränderung.

Das Beispiel bezieht sich auf eine Klasse. Durch eine Erweiterung auf die Ebene der Softwarearchitektur entsteht das Plug-in-Muster. Das bedeutet, ein Plug-in stellt einem fertigen System zusätzliche Erweiterungen bereit, die zur Laufzeit eingebunden werden können und vorher dem System unbekannt sind. Die Kommunikation wird durch eine Schnittstelle möglich, die das System zur Verfügung stellt. Wird diese Schnittstelle implementiert kann die Klasse eingebunden werden. Abbildung 4.5 zeigt die Interaktion von Schnittstelle, System und konkreten Plug-ins.

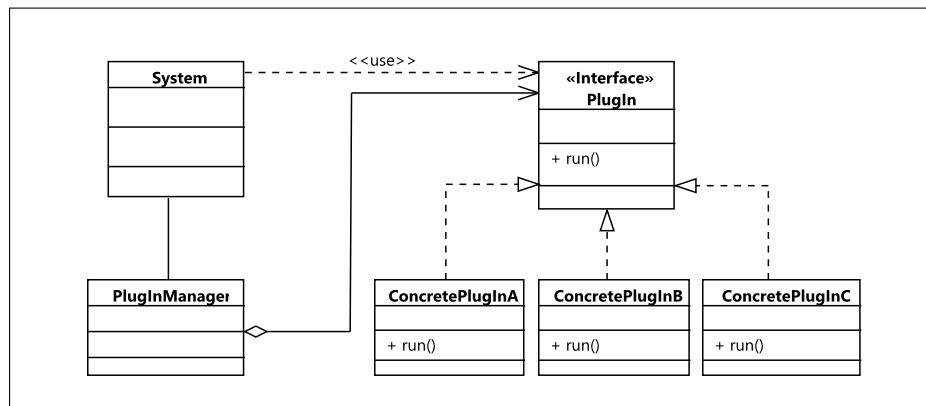


Abbildung 4.5.: UML Klassendiagramm des Architekturmusters Plug-in

# **Teil II.**

## **Entwicklung des**

## **Java Medical Imaging Toolkit -**

## **jMediKit**

# 5. Softwarearchitektur des Java Medical Imaging Toolkit

## 5.1. Die Eclipse Rich Client Platform

Die Grundlage für eine modulare Entwicklung der Software liefert die Eclipse Rich Client<sup>1</sup> Platform (RCP). Eclipse, basierend auf der Programmiersprache Java, wird seit 2001 von einer OpenSource-Gemeinde entwickelt[Vog13b]. Während es anfangs rein für die Java-Applikationsentwicklung entworfen wurde, ist es heute eine allgemeine erweiterbare Entwicklungsumgebung. So lässt sich zum einen das Grundprogramm mit Hilfe sogenannter Plug-ins erweitern und zum anderen eigenständige Applikationen erstellen (RCP), die auf dem Eclipse-Framework aufzubauen. Das „Aptana Studio<sup>2</sup>“ ist beispielsweise ein Plug-in, das der Grundentwicklungsumgebung mehrere Funktionalitäten im Bereich der Webentwicklung (Kommunikation zum Server, Syntaxhighlighting von HTML und CSS) hinzufügt. RSS Owl<sup>3</sup>, ein Programm zur Verwaltung von Newsfeeds, ist ein Beispiel für eine eigenständige Rich Client Applikation auf Basis von Eclipse.

Eine Kernkomponente des Eclipse-Frameworks ist *Equinox*, eine Implementierung der OSGi-Spezifikation. OSGi bietet die Möglichkeit, modulare Java-Applikationen zu entwickeln und Softwarepakete (nach der Spezifikation „Bundles“ unter Eclipse „Plug-ins“ genannt) während der Laufzeit hinzuzufügen, zu entfernen, zu starten oder zu stoppen [Vog13a]. Das Java Medical Imaging Toolkit ist eine Implementierung eines Bundles beziehungsweise Plug-ins.

Abbildung 5.1 zeigt die Architektur der Eclipse Rich Client Platform.

Die untere Ebene bildet das Betriebssystem. Eclipse kann grundsätzlich plattformunabhängig unter Windows, verschiedener Linux Distributionen oder Mac OS eingesetzt

<sup>1</sup>Es lässt sich zwischen Rich Clients und Thin Clients unterscheiden. Rich Clients stellen sowohl die Präsentationsschicht als auch die logische Schicht auf dem Client zur Verfügung. Die Applikationsfunktionalität der Thin Clients wird komplett von einem Server bereitgestellt. Die Bedienung erfolgt meist über den Browser.

<sup>2</sup><http://www.aptana.com/>

<sup>3</sup><http://www.rssowl.org/>

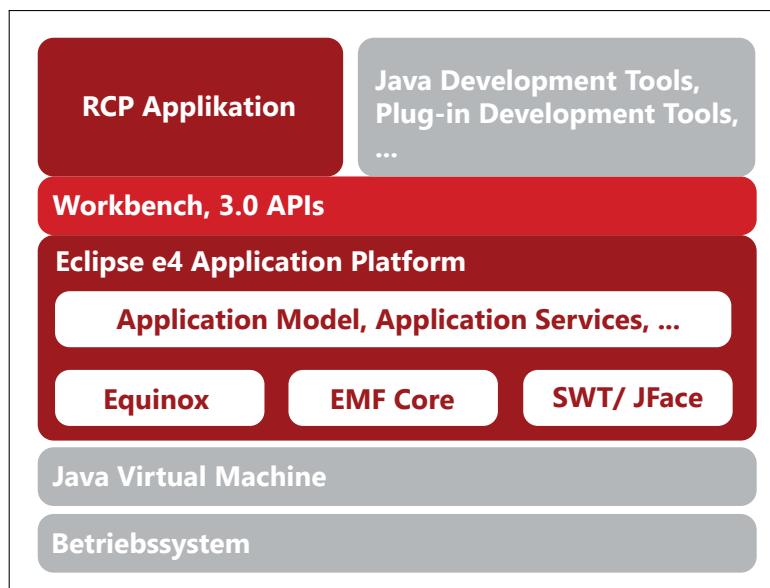


Abbildung 5.1.: Architektur der Eclipse e4 Plattform zur Entwicklung von Rich Client Applikationen

<https://wiki.eclipse.org/Eclipse4>

werden. Einzige Voraussetzung ist eine installierte Java Virtual Machine. Aufbauend auf Java steht der Eclipse-Kern, bestehend aus dem OSGi-Framework Equinox, dem Eclipse Modeling Framework EMF<sup>4</sup> und dem Standard Widget Toolkit SWT. Das SWT ist eine OpenSource Implementierung verschiedenster grafischer Bedienelemente wie Schaltflächen, Textfelder, Tabellen und vieles mehr. Der Unterschied zu den in Java integrierten grafischen Oberflächen besteht darin, dass das Standard Widget Toolkit auf die Ressourcen des Betriebssystems zugreift und sich in der Darstellung den Betriebssystemstandards anpasst [The14]. Aufbauend auf den Grundkomponenten liegt das *Application Model*, das die Struktur der Applikation (Menüs, Fenster, etc.) beschreibt [Vog13a, Kapitel 7]. Workbench und Eclipse 3.0 APIs bieten noch die Möglichkeit Anwendungen abwärtskompatibel zu entwickeln. Die gesamte Plattform bildet die Basis für das Java Medical Imaging Toolkit. Durch das Application Model kann modular entwickelt und die Programmstruktur in zukünftigen Versionen erweitert werden.

---

<sup>4</sup>Das EMF dient beispielsweise zur Entwicklung eines Datenmodells, wird allerdings für weitere Ausführungen nicht explizit benötigt.

## 5.2. Das Eclipse Application Model

Damit eine Anwendung strukturiert werden kann stellt das Application Model steuernde und visuelle Elemente zur Verfügung.

- **Strukturen zur visuellen Beschreibung der Applikation**

Das Aussehen wird mit Hilfe von Windows, Parts, PartStacks und Anderen beschrieben. Die Elemente enthalten noch keine Logik sondern definieren nur die Struktur der Anwendung.

- **Strukturen zur Steuerung des Verhaltens**

Zu den Komponenten, die das Verhalten der Applikation beeinflussen zählen zum Beispiel Tastatur-Shortcuts, Commands und Handler. Letztere dienen zur Verarbeitung von Benutzereingaben.

### 5.2.1. Visuelle Komponenten

#### Window

Windows sind einfache Repräsentationen eines Fensters der Benutzeroberfläche [The13, org.eclipse.e4.ui.model.application.ui.basic]. Sie bilden das Grundgerüst der Applikation und beinhalten Perspectives, PartStacks und andere Elemente. Ein einfaches Fenster ist in Abbildung 5.2.1 zu sehen.

#### Menüs

Ein Menü ist der Container für verschiedene Menü-Elemente und dient dazu Benutzereingaben entgegenzunehmen. Einem Element können Commands hinterlegt werden, damit die Eingaben weiter verarbeitet werden können. Ein Menüpunkt kann selbst ein Menü beinhalten.

#### Perspective

Perspectives beinhalten eine Menge von Elementen der Benutzeroberfläche wie PartStacks und Parts. Perspectives können unabhängig vom Rest der Oberfläche gewechselt werden [The13, org.eclipse.e4.ui.model.application.ui.advanced]. So können Perspectives beispielsweise die Anordnung der Parts definieren, oder neue Parts anzeigen, die in einer anderen Perspective nicht zu enthalten sein sollen. Unter Eclipse hat zum Beispiel

der Debug-Modul eine eigene Perspective und es kann dynamisch zwischen Debug- und Entwicklungsperspektive gewechselt werden.

### **PartSashContainer**

Wie der Name bereits andeutet, ist ein PartSashContainer ein Container für PartStacks und Parts. Die enthaltenen Elemente werden komplett angezeigt. In Abbildung 5.2.4 ist eine solche Kombination zu sehen. Die obere Hälfte stellt einen PartStack mit den beiden Parts *TestPart A* und *TestPart B* dar. Der Untere Teil ist ein Stack-unabhängiger Part.

### **PartStack**

Auch der PartStack dient als Behälter für einzelne Parts. Der Unterschied zum PartSashContainer liegt darin, dass bei PartStacks nur der aktuell ausgewählte Part angezeigt wird. Die Darstellung des Stacks ist mit üblichen Tabs zu vergleichen, wie Abbildung 5.2.3 zeigt. Der Stack enthält die beiden Parts *TestPart A* und *TestPart B*.

### **Part**

Ein Part ist die kleinste Einheit des Application Models und ist Kern der Benutzeroberfläche[The13, org.eclipse.e4.ui.model.application.ui.basic]. Innerhalb der Parts werden alle weiteren Bedienelemente angezeigt. Jede Abbildung von 5.2.2 - 5.2.4 enthält eine oder mehrere Parts. Betrachtet man das Application Model als Baumstruktur, symbolisieren Parts die Blattknoten. Parts können direkt einem Window unterstellt oder tief verschachtelt zwischen Perspectives und Stacks benutzt werden.

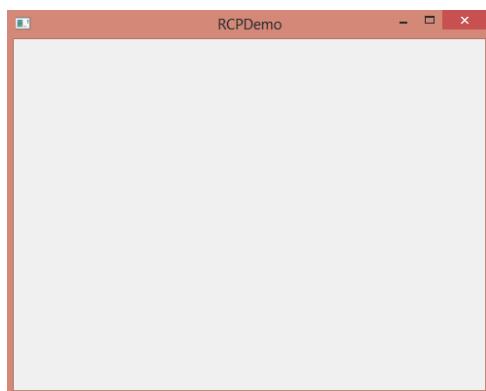
## **5.2.2. Steuernde Komponenten**

### **Commands**

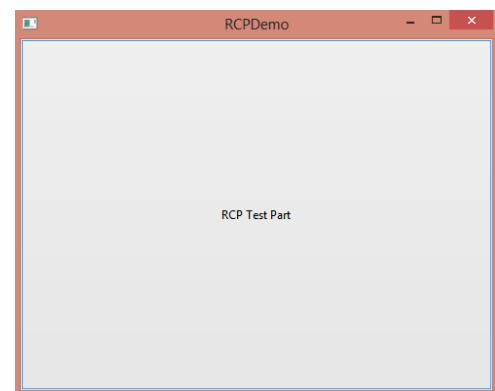
Commands bilden die abstrakte Schicht zwischen Benutzereingabe und Verarbeitung. Commands besitzen keine eigenen Implementierung. Das ermöglicht dem Entwickler das Verhalten individuell zu gestalten. So könnte ein Einfügen-Befehl im Editor ein anderes Verhalten auslösen als im Explorer-Part[The13, org.eclipse.ui.commands].

### **Handler**

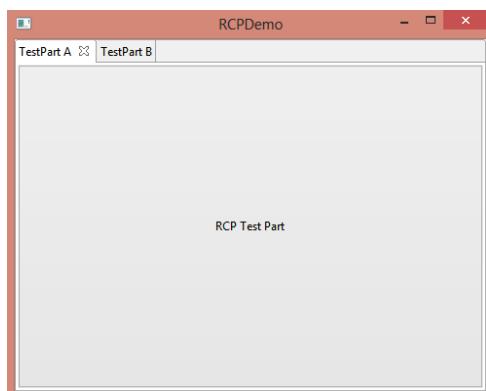
Handler sind die konkreten Implementierungen der Commands und sind für die Verarbeitung der Benutzereingaben verantwortlich.



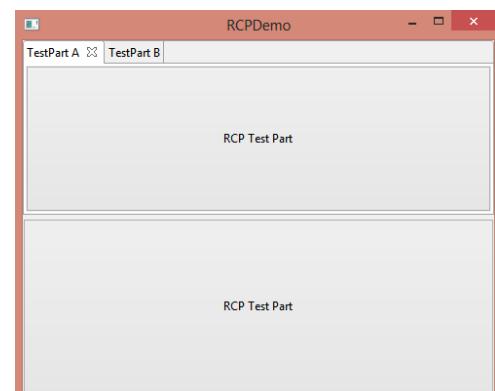
5.2.1: Window



5.2.2: Part



5.2.3: PartStack



5.2.4: PartSashContainer

Abbildung 5.2.: Verschiedene Elemente des Application Models

## 5.3. Die Benutzeroberfläche von jMediKit

Die Oberfläche des Java Medical Imaging Toolkit besteht aus sechs zentralen Elementen.

### 1. Hauptmenü

Das Hauptmenü stellt globale und bildspezifische Operationen zur Verfügung. So werden unter dem Menüpunkt *Erweiterungen* die importierten Plug-ins aufgelistet.

### 2. Werkzeugeleiste

Dieser Teil der Benutzeroberfläche stellt hauptsächlich Möglichkeiten zur Manipulation der Bilddaten zur Verfügung. Das Kapitel „6. Implementierung“ geht genau auf die verfügbaren Werkzeuge ein.

### 3. DicomBrowser

Dieser Part erlaubt die Navigation durch die vom Programm geladenen DICOM-Objekte. Die Anordnung entspricht der Darstellung des ER-Modells aus Kapitel „3. Grundlagen medizinischer Daten- und Bildformate“

### 4. ImageView

ImageView übernimmt die Anzeige der Pixeldaten der DICOM-Objekte.

### 5. Console

Die Konsole dient für die Fehlerausgabe bei der Plug-in-Entwicklung.

### 6. DicomTagView

Im DicomTagView werden die Tags eines ausgewählten DICOM-Objekts dargestellt.

Die hierarchische Struktur der Anwendung wird in Abbildung 5.4 dargestellt. Die sechs Blattknoten repräsentieren die nach außen für den Benutzer sichtbaren Teil der Anwendung.

## 5.4. Erweiterbarkeit der Grundstruktur

Die flexible Struktur des Eclipse Application Models und der Rich Client Platform erlauben komfortable Erweiterungen. So können einzelne Parts den schon bestehenden Elementen zugeordnet, oder neue Perspectives eingefügt werden, die eine neue Benutzeroberfläche abbilden könnten. Beispielsweise kann ein neuer Part den FileStack(Abbildung 5.4) damit erweitern, dass DICOM-Objekte von einem PACS geladen werden. Das Kapitel „7.

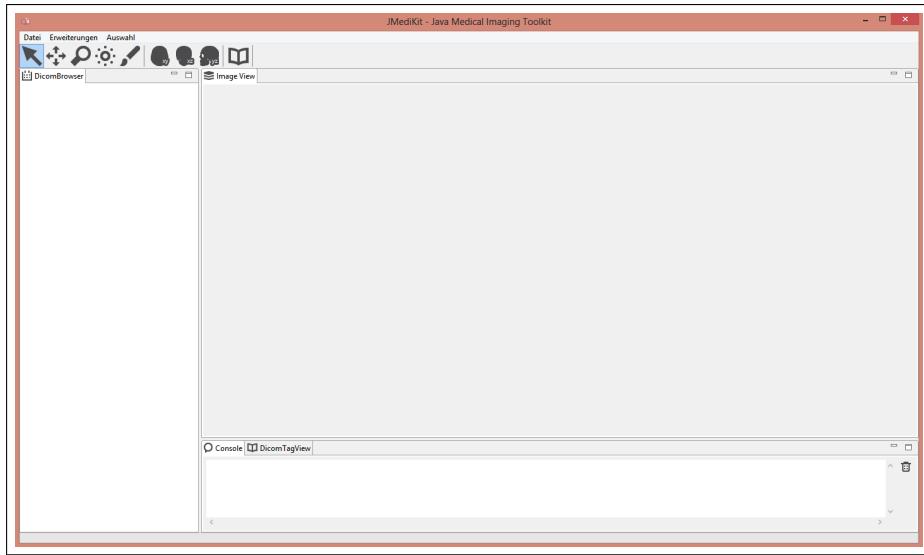


Abbildung 5.3.: Die Benutzeroberfläche von jMediKit

Entwicklung von Erweiterungen“ zeigt eine Möglichkeit, jMediKit einen neuen Part hinzuzufügen.

Durch das Hinzufügen von Elementen des Application Models ist es möglich in die Struktur und vor allem die Benutzeroberfläche einzugreifen. Das bedeutet auch, dass Parts oder andere Strukturen nur in der Lage sind neue Funktionen einzubauen. Es fehlt die Möglichkeit bereits existierende Elemente wie das Hauptmenü (MainMenu) oder die Werkzeugeleiste (Toolbar) zu erweitern.

## 5.5. Modularer Werkzeuge

Ein neuer Eintrag in das Hauptmenü und die Werkzeugeleiste ist mit Hilfe der Eclipse RCP-Bordmittel schnell erstellt. Dieser ist allerdings auch unabhängig der bisherigen Menüpunkte. So ist zum Beispiel der Eintrag *Datei öffnen* nicht abhängig vom Menüpunkt *Einstellungen*. Wird ein Eintrag ausgewählt, wird dieser ausgeführt und das Programm wartet auf die nächste Eingabe des Benutzers. Schwierigkeiten gibt es bei den Werkzeugen zur Bildmanipulation. Es kann immer nur ein Werkzeug aktiv ausgewählt sein. Zur Übersetzungszeit ist unklar, welche Werkzeuge wann erzeugt werden müssen. Daraus folgt, dass die verschiedenen Objekttypen der Werkzeuge erst zur Laufzeit erzeugt werden können, abhängig welches Werkzeug vom Benutzer ausgewählt wird. Wie in Abschnitt 4.3.1 beschrieben, kann die Fabrik Methode für Anwendungsfälle dieser Art der Objekterzeugung

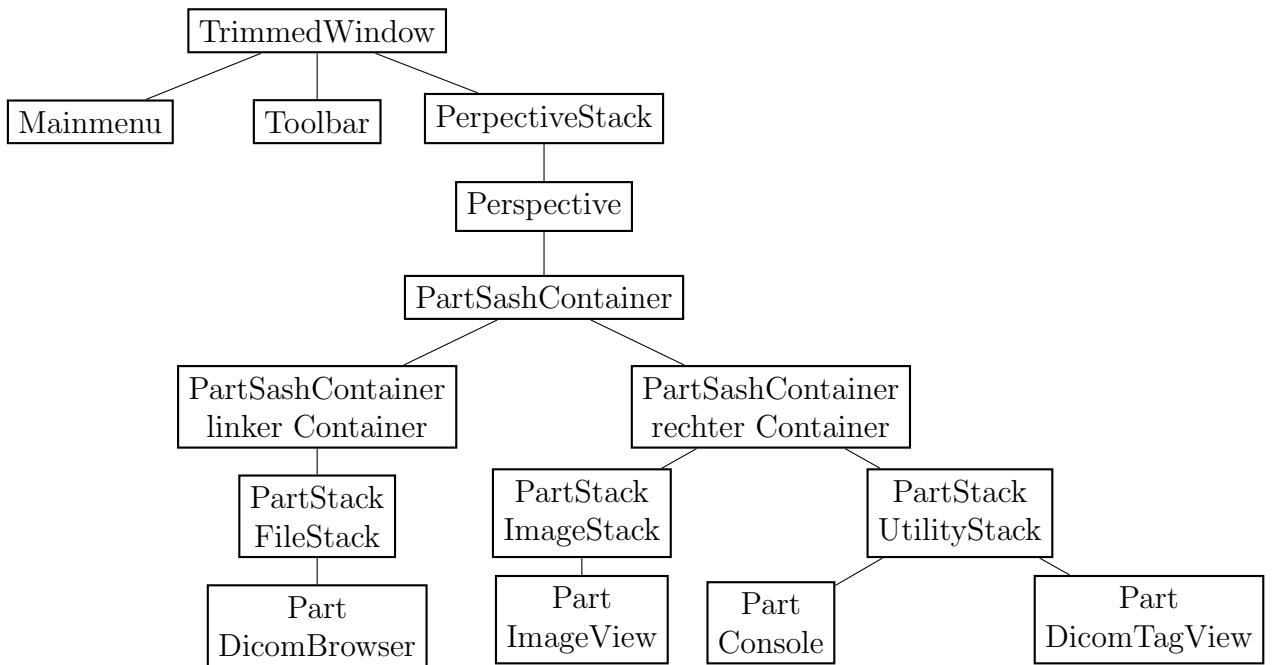


Abbildung 5.4.: jMediKit und die hierarchische Anordnung der Elemente des Application Models

eingesetzt werden. Bei den Klassen wird das Open-Closed-Prinzip beachtet. Die abstrakte Klasse *ATool* gibt ein Grundwerkzeug vor und bietet eine Schnittstelle zur Erweiterung. *AToolFactory* gibt die Vorgehensweise zur Werkzeugerzeugung vor. Abgeleitete Fabriken klassifizieren die Werkzeugkategorien.

Abbildung 5.5 zeigt die Umsetzung dieses Erzeugungsmusters. Die Grundversion von jMediKit enthält die zwei konkreten Fabriken *TransformationToolFactory* und *SelectionToolFactory*. Erstere beinhaltet Werkzeuge zur Bildtransformation wie die Skalierung (*ResizeTool*) und letztere enthält ein Werkzeug zur Auswahl von Punkten im Bild(*PointTool*). Sollen weitere Werkzeuge entwickelt werden, können diese von der Superklasse *ATool* abgeleitet werden. Passt das Tool in keine verfügbare Fabrik, kann auch hier eine neue Rubrik erzeugt werden.

## 5.6. Die Plug-in Architektur

Die Entwicklung von Erweiterungen durch den Anwender spielt eine zentrale Rolle in dieser Abschlussarbeit. Die Entwicklung von zusätzlichen Funktionen und Bedienelemente mittels des Application Models sind erst nach einem erneuten Build-Prozess verfügbar. Es soll den Anwendern eine Möglichkeit geboten werden, eigenständig Erweiterungen zu

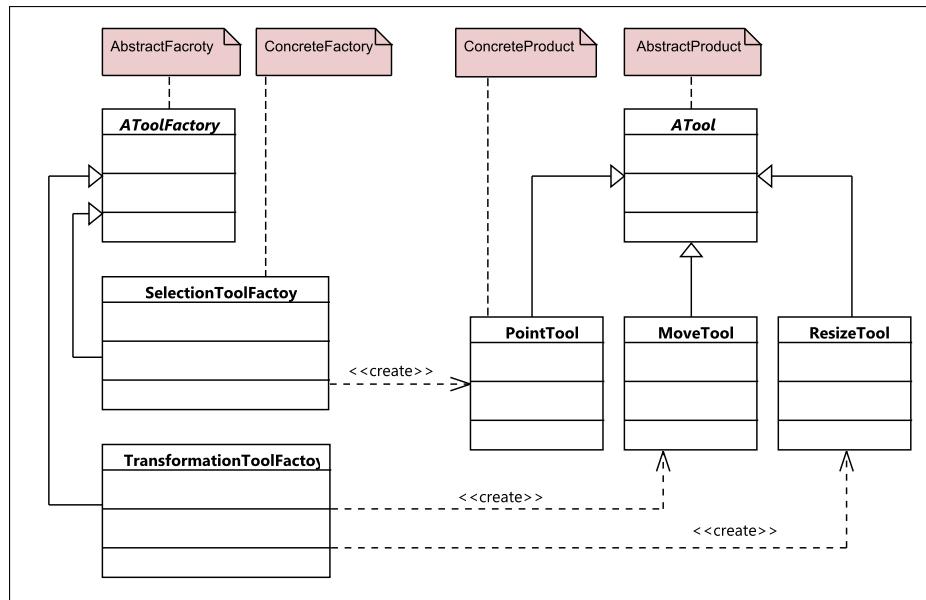


Abbildung 5.5.: Die Fabrik Methode zur Werkzeugerzeugung

entwickeln. Der Wirkungsbereich dieser Plug-ins beschränkt sich auf die Manipulation der Bilddaten. Dadurch ist kein explizites Wissen zur Eclipse-Plattform nötig.

Die Grundlage der Plug-in Architektur bildet das in Kapitel 4 vorgestellte Plug-in-Muster. In den Anforderungen ist festgelegt, dass diese Programmerweiterungen sowohl in zwei- als auch dreidimensionaler Richtung arbeiten sollen. Dadurch ist eine Anpassung des Architekturmusters notwendig. Als Ergebnis wird eine Kombination aus Plug-in, Schablonenmethode und Singleton eingesetzt. Das UML-Diagramm in Abbildung 5.6 zeigt die Architektur und Zusammenarbeit der drei Muster.

### 5.6.1. Plug-in als Grundstruktur

Die Manager-Klasse des Architekturmusters ist *PlugInClassLoader*. Diese Klasse dient sowohl zum Laden der Plug-ins, als auch zum Instantiieren der Plug-in-Objekte. Aufgrund des eingeschränkten Wirkungsbereich der Plug-ins ist die Zeichenfläche der Anwendung (*Canvas*) das einzige Modul, das über den Manager Plug-in-Objekte erzeugen kann.

Ein Unterschied zur UML-Darstellung in Abbildung 4.5 besteht darin, dass keine Schnittstelle zur Implementierung, sondern eine abstrakte Klasse *APlugIn* zur Ableitung zur Verfügung gestellt wird. Das bringt den Vorteil, gemeinsame Methoden zu definieren die

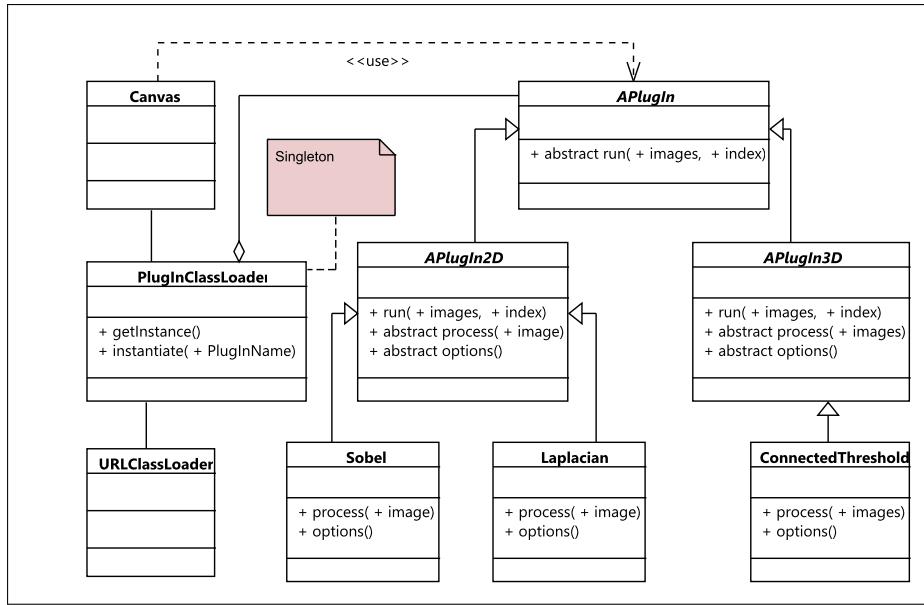


Abbildung 5.6.: Architektur der Plug-in-Struktur

in allen Plug-ins enthalten sind.

### 5.6.2. Erweiterung mit der Schablonenmethode

Nach den Anforderungen in Kapitel 2 ist es nötig, Plug-in-Schnittstellen sowohl für die Entwicklung von zwei- als auch dreidimensionalen Bilddaten anzubieten. Um dies zu erfüllen, stehen dem Benutzer die beiden abstrakten Klassen *APPlugIn2D* und *APPlugIn3D* zur Verfügung. Beide Varianten repräsentieren die Schablonenmethode. Das Template wird von der Methode *run()* repräsentiert. Der generische Teil des Algorithmus ist *process()*. Die Schablone ist notwendig, da abhängig von 2D und 3D andere Bilddaten an den Benutzer zum Bearbeiten übergeben werden müssen. Nach dem Aufruf von *process()* wird zusätzlich die Rückgabe des Benutzers von der Templatemethode *run()* validiert. Die Schablone ermöglicht dadurch eine Vorverarbeitung und Nachbearbeitung der Bilddaten.

### 5.6.3. Singleton als Plug-in Manager

Der Manager *PlugInClassLoader* ist als Singleton realisiert. Dadurch wird das Problem umgangen, dass mehrere Manager die gleichen Plug-ins laden. Passiert dies, sind gleiche Klassen zueinander inkompatibel und es kann nicht sichergestellt werden, welche Klasse von welchem Manager zur Verfügung gestellt wird.

Damit neue Klassen zum Programmstart eingebunden werden können, wird der Java System Classloader über den Manager um einen URL-Classloader erweitert. So werden zu den bisherigen Klassen der Java Virtual Machine alle die Plug-in-Klassen hinzugefügt.

## 5.7. Externe Bibliotheken

5.

Fremde Bibliotheken werden aus verschiedenen Gründen benötigt. Zum einen für das Lesen von DICOM-Objekten und zum anderen bieten Bildverarbeitungsbibliotheken eine gute Grundlage an Funktionen. Zur Verarbeitung von DICOM-Dateien verwenden jMediKit die freie Bibliothek „dcm4che“. Für die Bildverarbeitung wird „Simple ITK“ eingesetzt.

### 5.7.1. dcm4che

Im Kapitel „Grundlagen medizinischer Daten- und Bildformate“ wurden nur die Grundlagen gezeigt. Die Standards zu DICOM-Objekten, Kommunikation und Speicherung sind komplex. Daher wird zur Verarbeitung medizinischer Daten(vor Allem Patienten- und Bilddaten) auf externe Bibliotheken zurückgegriffen. Für eine Implementierung in Java standen folgende zur Auswahl: *Pixelmed*<sup>5</sup> und *dcm4chee*<sup>6</sup>. Beide bieten mit DICOM-Objekt-Verarbeitung und Netzwerkkommunikation ähnliche Dienste.

*dcm4che.org* bietet allerdings eine Software namens *dcm4chee* um ein PACS zu betreiben. Diese wird im Labor für medizinische Bildverarbeitung eingesetzt. Das ermöglicht eine leichtere Integration für spätere Erweiterungen von jMediKit in die bestehende Umgebung des Labors.

*Dcm4che2* wird unter der GNU General Public License. Dies ermöglicht die freie Verwendung der Software. JMediKit setzt die Version 2.028 ein.

### Java Advanced Imaging Image I/O Tools

Um alle Bildformate (abhängig von der Transfersyntax der DICOM-Objekte werden Bilddaten komprimiert als JPEG oder JPEG200 gespeichert) lesen zu können, verwendet *dmc4che2* die Bildverarbeitungsbibliothek *Java Advanced Imaging Image I/O Tools*. Diese ist *nicht* in *dcm4che2* integriert und muss vom Anwender selbst auf dem System installiert werden. Die benötigten Dateien und Pakete können unter <http://download.java.net/media/jai-imageio/builds/release/1.1/> - Stand 31.01.2014 bezogen werden.

---

<sup>5</sup><http://www.pixelmed.com/>

<sup>6</sup><http://www.dcm4che.org/>

### 5.7.2. Simple ITK

Das *Insight Segmentation and Registration Toolkit (ITK)*<sup>7</sup> ist eine umfangreiche Sammlung an Algorithmen zur Segmentierung und Registrierung. Das ITK ist plattformübergreifend einsetzbar und wurde in C++ implementiert. Zwar sind Java- oder Python-Wrapper<sup>8</sup> verfügbar, für die aktuelle Version 4 war es zum Zeitpunkt der Erstellung dieser Arbeit allerdings nicht möglich zuverlässig einen ITK-Build für eine Verwendung in Java zu erstellen.

Das *Simple ITK*<sup>9</sup> bietet eine kompakte Implementierung des ITK. Insgesamt sind 75% der Klassen aus dem ITK integriert [LVTN09]. Für einen Einsatz in einer Java Umgebung, kann das Projekt als Java-Bibliothek bezogen werden. Simple ITK steht unter der Apache 2.0 Lizenz und erlaubt eine uneingeschränkte Nutzung.

### 5.7.3. Der Adapter zur Auflösung von Abhängigkeiten

Eine Nutzung von externen Bibliotheken bedeutet gleichzeitig, dass Abhängigkeiten zwischen dem zu entwickelndem Programm und den fremden Paketen geschaffen werden. Werden bibliotheksspezifische Objekte im Quelltext erzeugt ist man abhängig von einer kontinuierlichen Weiterentwicklung der eingebundenen Projekte. Bei einem Wechsel der Bibliotheken müssen alle referenzierten Objekte angepasst werden und der Code ist nicht mehr ohne erheblichen Mehraufwand wartbar.

Mit Hilfe des Adapters werden bei jMediKit die Abhängigkeiten aufgelöst. In Abbildung 5.7 wird das Klassendiagramm dargestellt. Wie auch das Plug-in Muster wird auch der Adapter mit Anpassungen eingesetzt. Unter jMediKit wird ein DICOM-Objekt aus zwei unterschiedlichen Teilen repräsentiert. Ein Teil (*IDicomData*) beinhaltet den reinen Datenteil. Das bedeutet alle verfügbaren Tags des DICOM-Objekts sind enthalten. Der zweite Teil (*IDicomImageData*) besteht aus den reinen Pixeldaten. Diese beiden Schnittstellen bilden den Adapter für die externe Bibliothek *dcm4che2*. Durch die Trennung von Daten und Bilddaten bleibt die Flexibilität bei der Bibliothekswahl erhalten. So kann zum Beispiel Simple ITK nur für das Einlesen der Bilddaten verwendet werden, denn es werden keine Methoden zum Auslesen der Tags geboten. Dadurch wäre eine zweite Bibliothek für den Datenteil notwendig. In der Implementierung des Adapters wird in beiden Tei-

---

<sup>7</sup><http://www.itk.org/>

<sup>8</sup>Wrapper machen es möglich, Bibliotheken fremder Sprachen in aktuellen Sprachen einzubinden und zu benutzen. So wäre es möglich, das ITK, welches in C++ implementiert ist, über Wrapper in Java nutzen zu können

<sup>9</sup><http://www.simpleitk.org/>

len *dcm4ch2* eingesetzt, da diese Bibliothek sowohl Daten als auch Bilddaten verarbeiten kann. Die beiden Klassen *DicomImageData* und *DicomData* implementieren die Schnittstellen *IDicomImageData* und *IDicomData*. Keine anderen Klassen greifen auf Objekte und Methoden aus den Bibliotheken zu.

Um beide Aspekte zu vereinen, stellen die Schnittstellen *IDicomData* und *IDicomImageData* die neuen zu adaptierenden Funktionen dar, die von *ADicomObject* adaptiert werden. Der konkrete Adapter ist *DicomObject*. Um die Abhängigkeiten aufzulösen und die Flexibilität in der Bibliothekswahl zu wahren wird dieser geschachtelte Adapter eingesetzt.

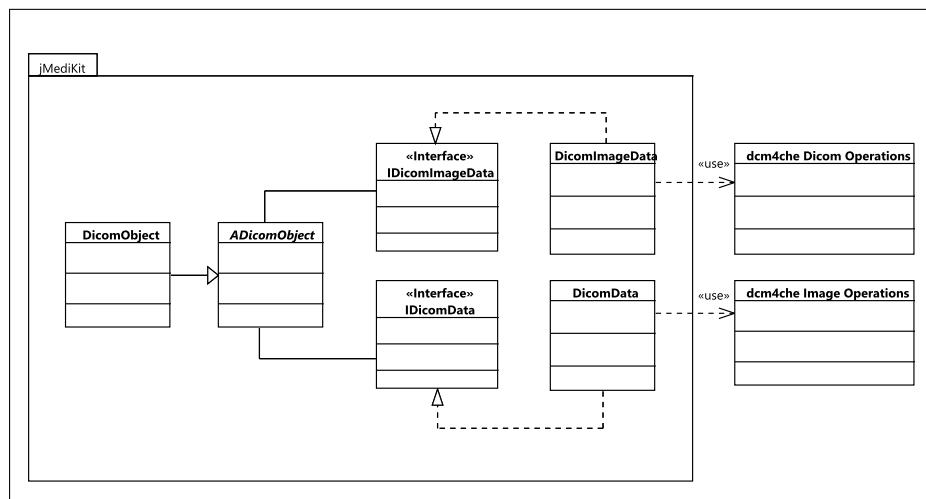


Abbildung 5.7.: Das Adapter-Muster unter *jMediKit*

Zusätzlich wird die Schnittstelle zur Bibliothek vereinfacht, indem nur Funktionen zur Verfügung gestellt werden, die tatsächlich eingesetzt werden. Werden zusätzliche Funktionen benötigt, kann das Interface der Adapter erweitert werden.

Soll zukünftig eine Bibliothek getauscht werden, muss nur das Interface neu implementiert werden und die Anwendung kann wie gewohnt eingesetzt werden.

## 5.8. Die Architektur der Bilddaten

Die Bilddaten sind ein Teil des *ADicomObjects*. Diese werden mit Hilfe der abstrakten Klasse *AImage* dargestellt und enthalten viele zusätzliche Informationen wie Bilddimension, Fensterungsdaten oder die Position des Patienten im Raum. Wie in Kapitel 3 erläutert, besitzen medizinische Bilder unterschiedliche Grauwert-Tiefen oder Farbdarstellungen.

Um die Unterschiedlichen Bildtypen abzubilden stehen die konkreten Klassen *UnsignedByteImage* → 8-Bit, *ShortImage* → 16-Bit, *UnsignedShortImage* → 16-Bit und *IntegerImage* → 32-Bit Farbbild zur Verfügung. Ein *ADicomObject* beinhaltet alle Bilddaten der entsprechenden Serie des DICOM-Objekts.

Um mit *AImage* zu arbeiten gibt es zwei Möglichkeiten: Die Erste ist das Auslesen über das *ADicomObject*. Hierzu muss ein DICOM-Objekt aus einer Datei eingelesen werden. Im Bereich der Plug-in-Entwicklung ist es sinnvoll leere Bilder zu erzeugen und während der Anwendung des Plug-ins mit neuen Werten zu Füllen. Hierbei hilft die Klasse *SimpleImageFactory*.

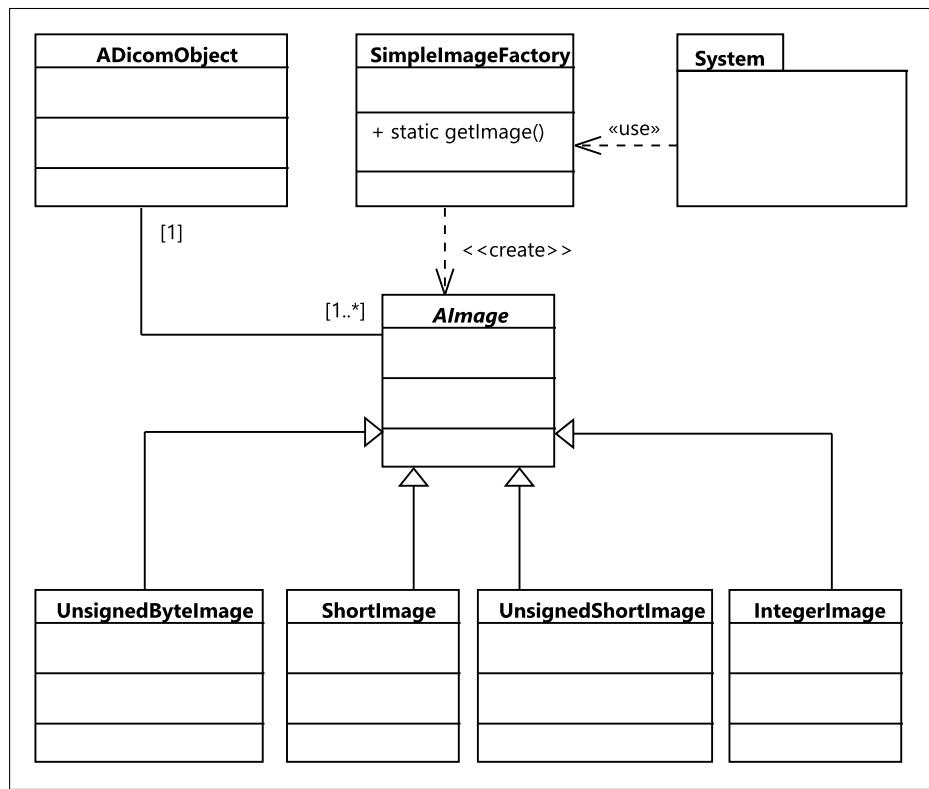


Abbildung 5.8.: Diagramm zur Klassenstruktur der Bilddaten

### 5.8.1. Die einfache Fabrik

Die einfache Fabrik ist ähnlich der Fabrikmethode, allerdings wird dieses Verfahren der Objekterzeugung nicht den Entwurfsmustern zugeordnet.

Beim Entwickeln von Plug-ins können Anwender vor dem Problem stehen, nicht zu wis-

sen, welche Grauwert-Tiefe das aktuell geladene Objekt besitzt. Nur ein mühsames Auslesen über die DICOM-Tags und oder einer Schleife zur Typ-Ermittlung würde Abhilfe schaffen. Der statischen Methode `getImage()` von `SimpleImageFactory` kann der Bildtyp des DICOM-Objekts übergeben werden, worauf ein typgerechtes Bildobjekt erzeugt und zurückgegeben wird.

### 5.8.2. Struktur der Bilder im ImageViewPart

Da die Struktur der Bilddaten dargestellt wurde, kann nun die Vorgehensweise zur Anzeige dieser vorgestellt werden.

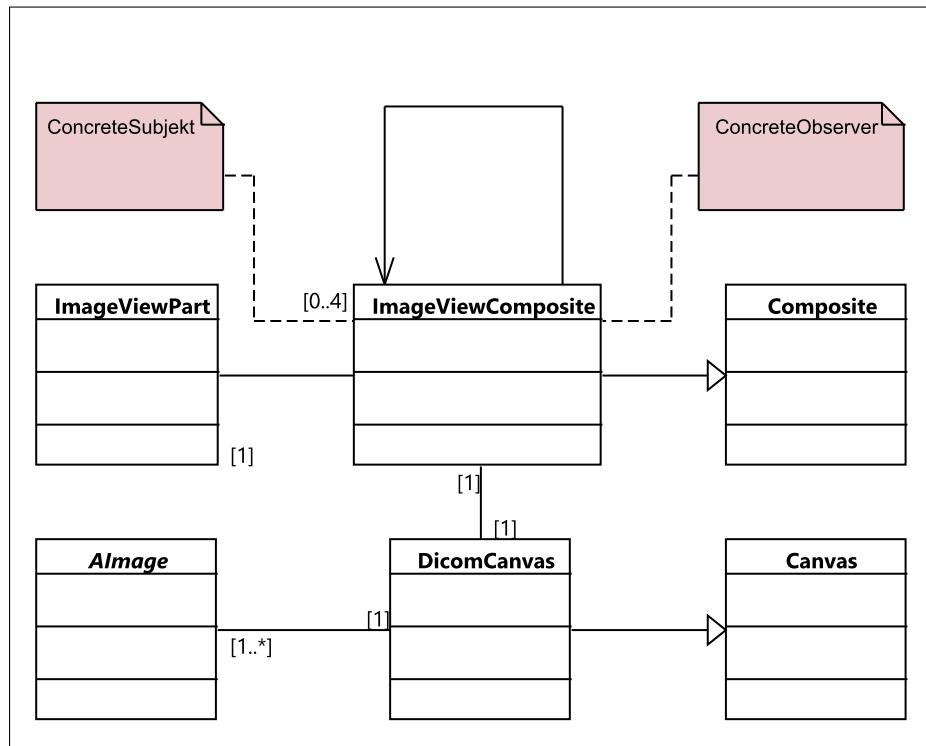
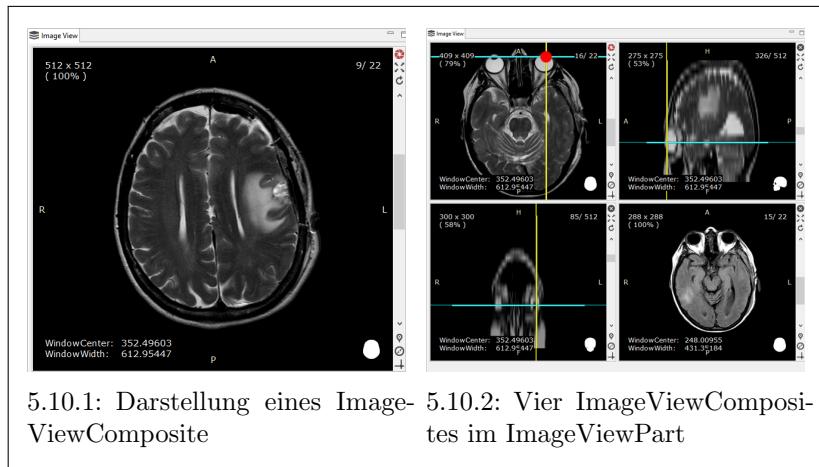


Abbildung 5.9.: Organisation der Klassen zur Anzeige der DICOM-Bilddaten

Auf der Benutzeroberfläche findet dieser Vorgang im ImageView Part statt (Abschnitt 5.4). Wie in Abbildung 5.9 zu sehen, bildet `ImageViewPart` die Grundlage und dient als Elternelement eines `ImageViewComposites`, von diesen maximal vier zum gleichen Zeitpunkt angezeigt werden können. `ImageViewComposites` erben von der SWT-Klasse `Composite` und enthalten weitere Bedienelemente für den Benutzer. Hierzu zählt die Scrollleiste am

rechten Rand wie in Abbildung 5.10.1 zu sehen ist. Mit diesem Scrollbalken wird die Schicht aus den dreidimensionalen Bilddaten gewählt, die vom *DicomCanvas* angezeigt werden soll. Die Superklasse von *DicomCanvas* gehört ebenfalls zum Repertoire des SWT und ist das zentrale Element zur Bildanzeige. *DicomCanvas* enthält sämtliche Bilder des DICOM-Objekts.



5.10.1: Darstellung eines ImageViewComposite  
5.10.2: Vier ImageViewComposites im ImageViewPart

Abbildung 5.10.: Benutzeroberfläche der ImageViewComposites

Abbildung 5.10.2 zeigt vier *ImageViewComposites*, wobei alle außer Composite unten rechts die gleichen DICOM-Objekte anzeigen. Angenommen ein Benutzer führt einen Klick mit der rechten Maustaste auf den in rot markierten Punkt aus, müssen alle *ImageViewComposites* den gleichen Punkt im dreidimensionalen Raum referenzieren. Dieser wird durch den Schnittpunkt der Linien angezeigt. Eine genaue Beschreibung dazu befindet sich in Kapitel 6.

Damit dieses Verhalten von Seiten der Architektur unterstützt wird, muss sichergestellt werden, dass die *ImageViewComposites* untereinander kommunizieren können, dass eine Änderung stattgefunden hat.

Aus dem Kapitel 4 löst das Observer-Muster diese Anforderungen. Der Zyklus in Abbildung 5.9 zeigt die Funktionsweise. Jede *ImageViewComposite* ist gleichzeitig *ConcreteSubject* als auch *ConcreteObserver*. Das bedeutet, bei jeder Registrierung eines neuen DICOM-Objekts wird geprüft, ob dieses schon im *ImageViewPart* vorhanden ist. Ist dies der Fall, wird es als neuer Observer an bisherigen Subjects angemeldet.

Bei diesem Einsatz besteht die Gefahr einer Endlosschleife, wenn der Zyklus nicht beachtet wird. Die Schleife tritt ein wenn ein Subject die Observer benachrichtigt, die Observer ändern den Status und benachrichtigen darauf wiederum ihre Beobachter und so weiter. Bei der Implementierung von jMediKit wird eine Änderung angenommen und das *Dicom-*

*Canvas* neu gezeichnet und löst keine Rückmeldung einer Änderung aus.

Auf dieser Architekturgrundlage findet die im folgenden Kapitel erläuterte Implementierung statt.

# 6. Implementierung

Dieses Kapitel erläutert den Implementierungsvorgang des Java Medical Imaging Toolkits. Besonderer Wert wird auf die Umsetzung der Blattknoten wie in Kapitel 5 Abschnitt 5.3 gelegt, da diese Anwendungsteile direkt mit dem Anwender in Aktion treten.

Nach einer Beschreibung wie die DICOM-Objekte repräsentiert werden, wird umfangreich auf die Bilddarstellung und Manipulation eingegangen.

## 6.1. Implementierung der DICOM-Objekte

Die beiden Schnittstellen *IDicomData* und *IDicomImageData* bilden die Grundlagen aller DICOM-Objekte die vom jMediKit erzeugt werden und sind gleichzeitig die einzige Kommunikationsmöglichkeit zu den externen Bibliotheken. Der abstrakte Adapter *ADicomObject* stellt die Funktionen beider Schnittstellen zur Verfügung. Die wichtigsten Methoden stellen das Auslesen der Pixel und den einzelnen Tags eines DICOM-Objekts dar.

Das eingebundene *dcm4che* bietet neben der Verarbeitung von DICOM-Objekten auch eine Implementierung des Kommunikations- und Speicherstandards von DICOM an. Während der aktuellen Entwicklung wurde allerdings nur ein Bruchteil der DICOM-Verarbeitung zur Erfüllung der Anforderungen benötigt und der Kommunikations- und Speicherprozess fand keine Beachtung. Bei einer Implementierung sollte dennoch eine zukünftige Erweiterung im Auge behalten werden. Dabei soll ebenso das Open-Closed-Prinzip Beachtung finden.

Hierbei werden die Schnittstellen in fachspezifische Domänen eingeteilt. Das bedeutet *IDicomData* ist für das Verarbeiten der Tags zuständig, während *IDicomImageData* ausschließlich Bilddaten verarbeitet. Für weitere Versionen können weitere Domänen implementiert werden, wie zum Beispiel eine Schnittstelle *IDicomNetworkData*. Der Adapter vereint die Domänen zu einem vollen DICOM-Objekt.

## 6.2. Der DicomBrowser

Der DicomBrowser ist der zentrale Part zum Transfer der DICOM-Dateien aus dem Dateisystem zu les- und verarbeitbaren DICOM-Objekten. Abbildung 6.1 zeigt die Darstellung nach dem Einlesen eines Ordners mit DICOM-Dateien. Die Repräsentation entspricht dem ER-Modell aus Kapitel 3 Abschnitt 3.1.1. Der Knoten \ symbolisiert die Wurzel. *BRAINX*<sup>1</sup> steht für den Patientenname, gefolgt von in diesem Beispiel einer Studie, die wiederum aus sieben Serien besteht. Die einzelnen DICOM-Objekte als Blattknoten werden aufgrund der Übersichtlichkeit nicht angezeigt.

Die Repräsentation der Dateien ist nicht immer geordnet wie es das ER-Modell vorgibt und man kann nicht von einer sortierten Ordnerstruktur ausgehen. Die PACS-Software *dcm4chee* ordnet die Daten beispielsweise nach dem Aufnahmedatum. Es wird eine Datenstruktur benötigt, die der DICOM Object Definiton entspricht.

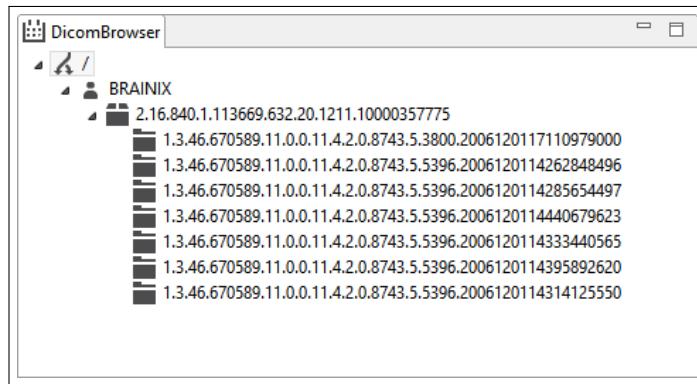


Abbildung 6.1.: Die Baumsicht des DICOM-Browsers mit geladenen Objekten

### 6.2.1. Die Baumstruktur

Sowohl die Anzeige, als auch die interne Behandlung der Daten soll so nah wie möglich an den DICOM-Standard angelehnt sein und unabhängig von der Auslieferung<sup>2</sup> der Dateien die Struktur des ER-Modells haben.

Ein Baum als Datenstruktur erfüllt die grundlegende Repräsentation mit den verschiedenen Knotentypen aus Patientenname, Studien, Serien und den DICOM-Objekten.

<sup>1</sup>Beispieldatensätze verfremden die tatsächlichen Patientennamen. Originale Datensätze hätten die Form  
*Nachname ^ Vorname*

<sup>2</sup>Unabhängig davon, ob Dateien von der Festplatte geladen oder über das Netzwerk bezogen werden.

Dadurch ergibt sich mit der Wurzel eine Höhe von

$$h = \max(4) \quad (6.1)$$

und folgende konkrete Höhen der Knotentypen.

## 6.

$$h_{Root} = 0 \quad h_{Patientenname} = 1 \quad h_{Study} = 2 \quad h_{Series} = 3 \quad h_{Object} = 4 \quad (6.2)$$

Der minimale Baum besteht nur aus der Wurzel und hat die Höhe  $h_{min} = 0$ . Nach der Multiplizität des Modells aus Abschnitt 3.1.1 hat der Baum, sobald ein Patientenname eingefügt wird, eine minimale Höhe von  $h_{min} = 3$ , da Patientenname und Study jeweils mindestens ein Kindelement enthalten. Die Breite des Baums ist unbestimmt, da Knoten eine beliebige Anzahl an Kindern besitzen können. Abbildung 6.2 zeigt einen Baum, wie er in der Anwendung repräsentiert werden könnte. Der Baum enthält alle Knotentypen von *Patientname* bis *Object*-Ebene.

Zwei Klassen des Quelltextes liefern die Basis des Baums:

- **DicomTreeRepository**

Diese Klasse repräsentiert den Baum. Sie enthält den Wurzelknoten, und einige graphentypische Operationen. Der Baum kann nach Knoten durchsucht werden und hat eine Funktion zum Einfügen neuer Knoten. Eine Löschfunktion wurde nicht implementiert, da der Baum nicht vom Benutzer manipuliert werden soll. Das Einfügen soll nur zum initialen Einlesen aufgerufen werden.

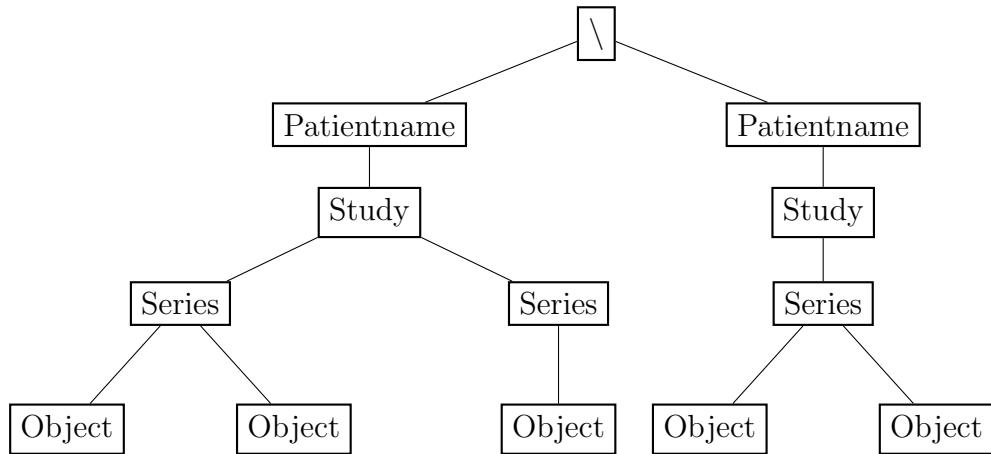
- **ADicomTreeItem**

*ADicomTreeItem* bilden die Knoten des Baumes. Jede Instanz besitzt eine Identifikationsnummer und kennt sowohl das Elternelement, als auch die Kindknoten.

Da nun die Datenstruktur bestimmt ist, fehlt das Vorgehen zur Sortierung der Daten. Abbildung 3.3 in Abschnitt 3.1 zeigt eine mögliche Dateistruktur. Das liefert allerdings keine Sicherheit, dass die Dateien immer vorsortiert zur Verfügung stehen. SLiegen alle DICOM-Dateien in einem Ordner, wäre eine Einteilung in Patienten und Serien etc. nicht mehr möglich.

Wie bereits in Abschnitt 3.1 beschrieben besitzt jeder Patient, jede Studie und jede Serie eine eigene eindeutige Identifikationsnummer(UID), die eine modellgerechte Sortierung ermöglicht. Die Tags, die diese UIDs enthalten sind *Patient ID*, *Study Instace UID*, *Series Instance UID* und *SOP Instance UID*.

Jede Datei repräsentiert ein Blatt im Baum und somit ein DICOM-Objekt. Dadurch muss

Abbildung 6.2.: Beispielhafte Darstellung eines Baumes mit  $h = 4$  in der Implementierung

die abstrakte Klasse `ADicomObjekt` aus der Architekturbeschreibung aus Kapitel 5 Abschnitt 5.7.3 angepasst werden und erweitert die abstrakte Superklasse `ADicomTreeItem`. Somit erben alle DICOM-Objekte die Eigenschaften eines Knoten im Baum. Zur weiteren Klassifizierung der Knoten werden die Klassen `DicomPatientItem`, `DicomStudyItem` und `DicomSeriesItem`, die alle von `ADicomTreeItem` erben, eingesetzt. Bei der Instantiierung des DICOM-Objekts wird direkt die UID über *SOP Instance UID* zugewiesen. Als nächster Schritt wird aus dem DICOM-Objekt der Pfad von der Wurzel zum Objekt ermittelt. Dazu werden *Patient ID*, *Study Instace UID* und *Series Instance UID* des Objekts ausgelernt. Nun wird der Baum nach den entsprechenden Objekten und den UIDs durchsucht. Sind diese nicht vorhanden, wird das zugehörige Objekt erzeugt und in den Baum eingefügt, bis letztendlich das DICOM-Objekt als Blatt eingehängt werden kann.

Mittels dieser Sortierung entsprechen die Elemente im `DicomBrowser` der Darstellung des ER-Modells.

### 6.3. Repräsentation der Pixeldaten

Ganzzahlige Datentypen in Java (*byte*, *short*, *int*, *long*) werden im Zweierkomplement kodiert[Ull07, S.106]. Dadurch sind nur Werte im Bereich von  $[-2^{BIT-1}, 2^{BIT-1} - 1]$ . Das entspricht beim Datentyp *short* dem Intervall von  $[-2^{15}, 2^{15} - 1] \rightarrow [-32768, 32767]$ . Medizinische Grauwertbilder besitzen meist eine Tiefe von 8-, 12- und 16-Bit. Zusätzlich bestimmt der DICOM-Tag *PixelRepresentation*, ob Pixelwerte vorzeichenbehaftet sind. Durch diese variablen Eigenschaften entstehen unterschiedliche Bildtypen. Aus dem Abschnitt 3.2.2 wird deutlich, dass Grauwertbilder mit einer Tiefe von 16-Bit den Bereich von

Datentyp	MIN	MAX	Unsigned
byte	-128	127	0 - 255
short	-32768	32767	0 - 65535
int	-2147483648	2147483647	0 - 4294967295
long	-9223372036854775808	9223372036854775807	0 - 18446744073709551615

Tabelle 6.1.: Ganzzahlige Datentypen in Java

Klassenname	Pixeltyp Code	Bittiefe Code	Bittiefe DICOM- Objekt	Vorzeichen
UnsignedByteImage	short	16	8	Ø
ShortImage	short	16	16	ja
UnsignedShortImage	int	32	16	Ø
IntegerImage (Farbbild)	int	32, 8-Bit je Kanal	32, 8-Bit je Kanal	Ø

Tabelle 6.2.: Von jMediKit implementierte Bildtypen

[0, 65535] abdecken. Dadurch entsteht eine Diskrepanz zwischen dem 16-Bit Java Datentyp *short* und den Grauwerten. Die Pixelwerte können vom Typ *short* nicht aufgenommen werden. Das gleiche Missverhältnis entsteht bei einer Tiefe von 8-Bit. Wie in Tabelle 6.1 zu sehen, fasst der Datentyp *byte* maximal einen Wert von 127 während der größte Pixelwert 255 entspricht. Daraus folgt, dass Grauwertbilder ohne vorzeichenbehaftete Werte (Unsigned) mit dem nächsthöheren Datentyp repräsentiert werden.

Tabelle 6.2 zeigt eine Darstellung der implementierten Bildtypen und die zugehörigen Datentypen der Pixel im Quelltext. Haben die Pixeldaten eines DICOM-Objekts eine Tiefe von 8-Bit und sind nicht vorzeichenbehaftet, wird zur Repräsentation in der Implementierung ein Array des Typs *short* verwendet, um alle Werte aufnehmen zu können. Der Datentyp *int* könnte alle Pixelwerte eines DICOM-Objekts aufnehmen. So liegt es nahe, dass Integer durchgehend als Datentyp verwendet wird. Arbeitet man allerdings mit 8-Bittiefe ohne Vorzeichen, wird der Speicherbedarf von 16 auf 32 Bit pro Pixel verdoppelt. Daher ist es sinnvoll die Bildtypen zu kategorisieren.

### 6.3.1. Räumliche Sortierung der Bilddaten

Beim Erstellen des Baums aus Abschnitt 6.2.1 wird rekursiv das Verzeichnis durchlaufen und nach lesbaren DICOM-Objekten gesucht. Ist die Suche erfolgreich, wird das Objekt dem Baum hinzugefügt. Hierbei kann das Problem auftreten, dass Dateien in der falschen Reihenfolge importiert werden. So kann es passieren, dass ein Importvorgang abhängig vom Dateinamen Objekte dem Baum hinzufügt. Die räumliche Reihenfolge der einzelnen Schichten entspricht allerdings keineswegs dem Dateinamen<sup>3</sup> oder anderen dateibezogenen Reihenfolgen. Dadurch wird, wie schon bei der Sortierung nach dem ER-Modell, eine Methode benötigt, mittels DICOM-Tags die korrekte Reihenfolge der Bilddaten herzustellen.

Der Standard verfügt über mehrere Tags, welche die richtige Reihenfolge andeuten. *Instance Number* ist nach dem Standard [Nat11a, C.7.6.1] eine Nummer, die ein Bild identifiziert. Während der Entwicklung entsprach dieser Wert der Testbilder der dargestellten Reihenfolge, jedoch ist keine Information enthalten ob diese der tatsächlichen Reihenfolge im Raum entspricht. So könnte der Wert für die Aufnahmenreihenfolge oder andere konsekutive auf- oder absteigende Folgen stehen. Dadurch ist *Instance Number* nur bedingt geeignet.

Ein Tag, der räumliche Informationen enthält ist *Slice Location*. Nach dem Standard [Nat11a, C.7.6.2] ist dieser Wert die relative Position der Bildebene in mm. Es ist allerdings keine Information enthalten, ob die Sortierung in steigender oder fallender Reihenfolge erfolgt. Da der Tag zusätzlich nur optional vorhanden ist, kann eine Nutzung zur Bestimmung der Anordnung ausgeschlossen werden.

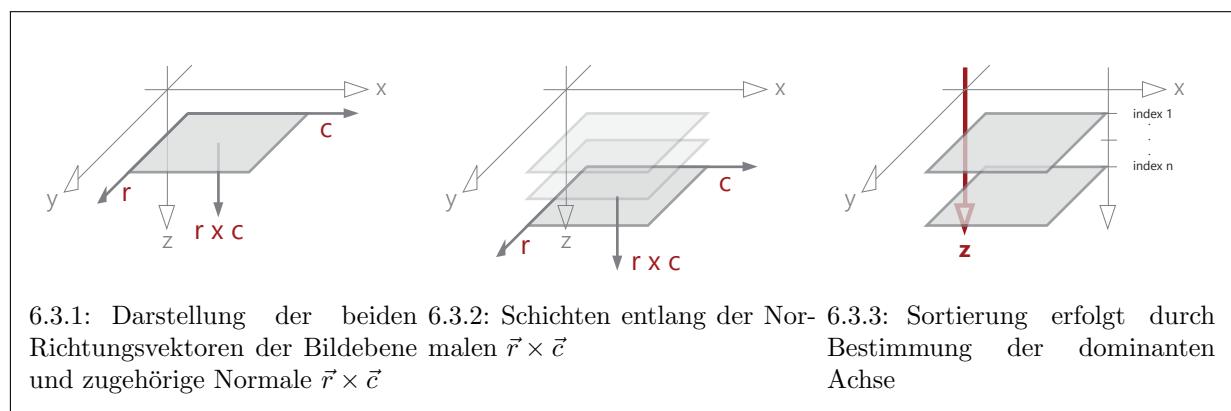


Abbildung 6.3.: Räumliche Sortierung der DICOM-Objekte

<sup>3</sup>Je nach Sortierung innerhalb des Betriebssystems könnte ein Import auch nach dem Änderungsdatum erfolgen

In der Mailingliste des Insight Segmentation and Registration Toolkits [Smi03] wird eine Berechnung über die Attribute *Image Position* und *Image Orientation* vorgeschlagen. *Image Position* enthält die  $x$ ,  $y$  und  $z$  Koordinate in mm und *Image Orientation* den Richtungskosinus<sup>4</sup> der ersten Reihe und Spalte des Bildes in Abhängigkeit des Patienten. Diese beiden Richtungsvektoren spannen die Bildebene auf und müssen nach [Nat11a, C.7.6.2.1.1] orthogonal zueinander sein.

Die Abbildungen in 6.3 zeigen das Koordinatensystem des Patienten als rechtshändiges System[Nat11a, S.419]. Dieses ist um 180 Grad um die x-Achse gedreht. In Abbildung 6.3.1 ist die Bildebene sowie die beiden Richtungsvektoren  $\vec{c}$  der ersten Spalte und  $\vec{r}$  der ersten Reihe dargestellt. Mit Hilfe des Kreuzproduktes lässt sich nun die Normale der Ebene bestimmen. Die räumliche Anordnung der Schichten erfolgt entlang des Normalenvektors  $\vec{r} \times \vec{c}$  (Abbildung 6.3.2). Die Richtungsvektoren könnten folgende Darstellung besitzen:

$$\vec{r} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \vec{c} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \vec{n} = \vec{r} \times \vec{c} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (6.3)$$

Nachdem die Normale  $\vec{n}$  der Ebene bestimmt ist, muss der dominante Anteil des Vektors ermittelt werden. Dazu wird der maximale Betrag aus den Elementen von  $\vec{n}$  mit  $|n_x|, |n_y|, |n_z|$  berechnet. Somit erhält man die Achse an der die Schichten angeordnet sind. Im Beispiel 6.3 und Abbildung 6.3.3 erfolgt die Anordnung entlang der z-Achse, da  $n_z$  den dominanten Anteil von  $\vec{n}$  darstellt. Ist  $n_z < 0$  erfolgt die Wuchsrichtung der Schichten entlang des negativen Anteils der z-Achse. Wenn  $n_z >= 0$  wachsen die Ebenen in die positive Richtung.

Mit Hilfe dieser Kriterien lassen sich die Bilder über den Tag *Image Position* sortieren. Es ist bekannt, welche Achse die Reihenfolge im Raum symbolisiert. Beispiel 6.4 zeigt drei Vektoren mit möglichen Daten von *ImagePosition*. Ist der dominante Anteil des Vektors kleiner 0, verläuft die Wuchsrichtung negativ und der größte Wert ist das erste Bild der Folge. Ist der dominante Anteil positiv, hat das erste Bild des kleinsten Wert. Da im

---

<sup>4</sup>Ein Richtungskosinus beschreibt die Winkel des Vektors zu den drei Koordinatenachsen

Beispiel 6.3  $n_z >= 0$  entspricht die Sortierte Reihenfolge aus Beispiel 6.4  $\vec{b} \rightarrow \vec{a} \rightarrow \vec{c}$ .

$$\vec{a}_{position} = \begin{pmatrix} 13.6 \\ 122 \\ 75 \end{pmatrix} \quad \vec{b}_{position} = \begin{pmatrix} 13.7 \\ 121.6 \\ 65 \end{pmatrix} \quad \vec{c}_{position} = \begin{pmatrix} 12.6 \\ 122.1 \\ 85 \end{pmatrix} \quad (6.4)$$

**6.**

Durch eine Implementierung des Interface *Comparable<AImage>* in *AImage* unter Berücksichtigung dieser Vorgehensweise ist eine einfache und für diesen Zweck ausreichend schnelle Sortierung der Bildebene mögliche. Somit erfolgt die Anordnung des Baums und der Bilder unabhängig der Dateistruktur.

Voraussetzung für diese Umsetzung ist, dass die Bildebene parallel zu den Koordinatenachsen verlaufen. Bei Bildreihen mit Kurven oder einem schrägen Verlauf kann keine dominante Achse bestimmt werden.

## 6.4. Zeichnen der Bilddaten mit dem DicomCanvas

Sowohl die DICOM-Objekte, als auch Bilddaten stehen im Speicher zur Verarbeitung bereit. Dieser Abschnitt befasst sich mit der Visualisierung dieser Daten.

#### **6.4.1. Implementierung der Werkzeuge**

**Das Bild bewegen mit dem MoveTool**

**Skalierung mit dem ResizeTool**

**Justierung der Fensterung mit dem WindowTool**

**Punktauswahl mit dem PointTool**

**6.**

### **6.5. Das Utility-Fenster**

**6.5.1. Debugging mit dem ConsoleView**

**6.5.2. DICOM-Objekte über DicomTagView ausgeben**

### **6.6. Implementierung des Hauptmenüs und der Werkzeugleiste**

**6.6.1. Das Hauptmenü**

**6.6.2. Die Werkzeugleiste**

## 7. Entwicklung von Erweiterungen

# Literaturverzeichnis

- [Bal11] BALZERT, Helmut: *Lehrbuch der Softwaretechnik - Entwurf, Implementierung, Installation und Betrieb*. Spektrum, 2011
- [BLT98] BÜCHELER, Egon ; LACKNER, Klaus-Jürgen ; THELEN, Manfred: *Einführung in die Radiologie: Diagnostik und Interventionen*. Georg Thieme Verlag, 1998
- [Cor07] CORD, Siemon: Innovationspolitik im 6. Kondratieff: Hinterherlaufen oder Vorauseilen? In: *Wirtschaftsdienst* 87 (2007), Juli, Nr. 7, S. 450–457
- [ES13] EILEBRECHT, Karl ; STARKE, Gernot: *Patterns kompakt - Entwurfsmuster für effektive Software-Entwicklung*. Springer Vieweg, 2013
- [GD13] GOLL, Joachim ; DAUSMANN, Manfred: *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer Vieweg, 2013
- [GN11] Kapitel Der sechste Kontratoeff. In: GRANIG, P. ; NEFIODOW, L. A.: *Gesundheitswirtschaft – Wachstumsmotor im 21. Jahrhundert*. Gabler Verlag, 2011
- [Han00] HANDELS, Heinz: *Medizinische Bildverarbeitung*. B.G. Teubner Stuttgart Leipzig, 2000
- [HK11] Kapitel Die gesunde Gesellschaft und ihre Ökonomie – vom Gesundheitswesen zur Gesundheitswirtschaft. In: HENSEN, P. ; KÖLZER, Christian: *Die gesunde Gesellschaft*. VS Verlag für Sozialwissenschaften, 2011
- [Hoc13] HOCHSCHULE LANDSHUT: *Modulhandbuch BA BMT*. [https://www.haw-landshut.de/fileadmin/hs\\_landshut\\_english/electrical\\_engineering/download/pdf/Modulhandb%FCcher/Modulhandbuch\\_BA\\_BMT\\_WS\\_13\\_14\\_SS\\_13\\_beschlossen\\_FR\\_2013\\_11\\_26.pdf](https://www.haw-landshut.de/fileadmin/hs_landshut_english/electrical_engineering/download/pdf/Modulhandb%FCcher/Modulhandbuch_BA_BMT_WS_13_14_SS_13_beschlossen_FR_2013_11_26.pdf). Version: November 2013
- [LVTN09] LEHMANN, Gaëtan ; VILVERT, Domaine de ; TONDDAST-NAVAEI, Ali: *WrapITK release page for version 0.3.0*. [http://code.google.com/p/wrapitk/wiki/Release030#Class\\_coverage](http://code.google.com/p/wrapitk/wiki/Release030#Class_coverage). Version: 2009

## LITERATURVERZEICHNIS

---

- [Nat11a] NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATION: *Digital Imaging and Communications in Medicine (DICOM) - Part 3: Information Object Definitions.* [ftp://medical.nema.org/medical/dicom/2011/11\\_03pu.pdf](ftp://medical.nema.org/medical/dicom/2011/11_03pu.pdf). Version: 2011
- [Nat11b] NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATION: *Digital Imaging and Communications in Medicine (DICOM) - Part 5: Data Structures and Encoding.* [ftp://medical.nema.org/medical/dicom/2011/11\\_05pu.pdf](ftp://medical.nema.org/medical/dicom/2011/11_05pu.pdf). Version: 2011
- [Nat11c] NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATION: *Digital Imaging and Communications in Medicine (DICOM) - Part 6: Data Dictionary.* [ftp://medical.nema.org/medical/dicom/2011/11\\_06pu.pdf](ftp://medical.nema.org/medical/dicom/2011/11_06pu.pdf). Version: 2011
- [Pia08] PIANYKH, Oleg S.: *Digital Imaging and Communications in Medicine (DICOM)*. Springer-Verlag Berlin Heidelberg, 2008
- [Smi03] SMITH, Jolinda: *Create 3D image data by a series of 2D dicom files t.* <http://www.itk.org/pipermail/insight-users/2003-September/004762.html>. Version: 2003. – Stand 05.02.2014
- [The13] THE ECLIPSE FOUNDATION: *Eclipse documentation - Current Release.* <http://help.eclipse.org/kepler/index.jsp>. Version: 2013
- [The14] THE ECLIPSE FOUNDATION: *SWT: The Standard Widget Toolkit.* <http://www.eclipse.org/swt/>. Version: 2014. – Stand 23.01.2014
- [Ull07] ULLENBOOM, Christian: *Java ist auch eine Insel - Das umfassende Handbuch.* Galileo Press, 2007
- [Vog13a] VOGEL, Lars: *Eclipse 4 RCP - Building Eclipse RCP applications based on Eclipse 4.* [http://www.vogella.com/tutorials/EclipseRCP/article.html#e4overview\\_eclipse4](http://www.vogella.com/tutorials/EclipseRCP/article.html#e4overview_eclipse4). Version: 2013. – Stand 23.01.2014
- [Vog13b] VOGEL, Lars: *Eclipse IDE Tutorial.* <http://www.vogella.com/tutorials/Eclipse/article.html#eclipseoverview>. Version: 2013. – Stand 23.01.2014
- [Wie12] Kapitel Kondratieff – Von der Dampfmaschine zum Menschen. In: WIEDER, M.: *Liquid Work*. Springer Fachmedien Wiesbaden, 2012

# Abbildungsverzeichnis

1.1.	Kondratieff-Zyklen	3
2.1.	RadiAnt - DicomViewer	11
2.2.	Screenshot Slicer 3D	12
2.3.	Die Benutzeroberfläche von ImageJ	13
3.1.	Kommunikationsprozess von Aufnahme zur Verarbeitung	16
3.2.	Vereinfachte Darstellung der Informationsobjekthierarchie von Dicomelementen	17
3.3.	Repräsentation der Information Objekte im Dateisystem	18
3.4.	Kodierungsreihenfolge von 4 Byte bei Little Endian- und Big Endian-Darstellung	22
3.5.	Beispiele unterschiedlicher Speicherbelegung	23
3.6.	Verschieden Graustufenbilder	25
3.7.	Fensterungstechnik zur Darstellung medizinischer Bilddaten am handelsüblichen Monitor	26
3.8.	Kodierung der RGB-Werte im Datenelement PixelData mit Hilfe der PlanarConfiguration	27
3.9.	Darstellung von 2- und 3-dimensionalen Bilddaten	27
4.1.	UML Klassendiagramm zum Adapter	30
4.2.	UML Klassendiagramm zum Observer	31
4.3.	UML Klassendiagramm der Schablonenmethode	33
4.4.	UML Klassendiagramm zur Fabrik Methode	34
4.5.	UML Klassendiagramm des Architekturmusters Plug-in	35
5.1.	Architektur der Eclipse e4 Plattform zur Entwicklung von Rich Client Applikationen	38
5.2.	Verschiedene Elemente des Application Models	41
5.3.	Die Benutzeroberfläche von jMediKit	43

## ABBILDUNGSVERZEICHNIS

---

5.4. jMediKit und die hierarchische Anordnung der Elemente des Application Models . . . . .	44
5.5. Die Fabrik Methode zur Werkzeugerzeugung . . . . .	45
5.6. Architektur der Plug-in-Struktur . . . . .	46
5.7. Das Adapter-Muster unter jMediKit . . . . .	49
5.8. Diagramm zur Klassenstruktur der Bilddaten . . . . .	50
5.9. Organisation der Klassen zur Anzeige der DICOM-Bilddaten . . . . .	51
5.10. Benutzeroberfläche der ImageViewComposites . . . . .	52
6.1. Die Baumansicht des DICOM-Browsers mit geladenen Objekten . . . . .	55
6.2. Beispielhafte Darstellung eines Baumes mit $h = 4$ in der Implementierung .	57
6.3. Räumliche Sortierung der DICOM-Objekte . . . . .	59
B.1. Standardinstallation eines 32-Bit Eclipse unter Windows 8 . . . . .	XI
B.2. e4 Projekt Builds . . . . .	XII
B.3. e4 Repository Link . . . . .	XIII
B.4. Installation neuer Software unter Eclipse . . . . .	XIII
B.5. Angabe des Repository . . . . .	XIV
B.6. Auswahl der e4 Tools zur Installation . . . . .	XIV
B.7. Vorgang zum Importieren bereits bestehender Projekte . . . . .	XV
B.8. Der Package Explorer nach dem Importvorgang . . . . .	XVI
B.9. Der Package Explorer nach dem Importvorgang . . . . .	XVI
B.10. Konfigurationsfenster zu Anwendungseinstellungen und Anwendungsstart .	XVII

# **Tabellenverzeichnis**

2.1.	Gegenüberstellung der Anforderungen und verfügbarer freier Software . . .	14
3.1.	Repräsentation des Patientennamen als DICOM-Element . . . . .	19
3.2.	Das erzeugte DICOM-Objekt mit den Elementen zu Patientenname, Ge- burtsdatum, Geschlecht und Alter . . . . .	20
3.3.	Grundlegende Datenelemente für die digitale Repräsentation . . . . .	24
6.1.	Ganzzahlige Datentypen in Java . . . . .	58
6.2.	Von jMediKit implementierte Bildtypen . . . . .	58
A.1.	Darstellung des Datenelements im Speicher wenn VR vom Typ OB, OW, OF, SQ, UT oder UN . . . . .	VIII
A.2.	Darstellung des Datenelements für alle anderen VR-Typen . . . . .	IX
A.3.	Darstellung des Datenelements für implizite VR . . . . .	X

## **Teil III.**

### **Anhang**

# **A. Darstellung der DICOM-Elemente im Speicher**

## **A.1. Explizite VR mit [ OB | OW | OF | SQ | UT | UN ]**

Bei expliziter VR-Struktur besteht das Element aus vier konsekutiven Feldern. Ist die VR vom Typ OB, OW, OF, SQ, UT oder UN wird das Datenelement wie in Tabelle A.1 im Speicher abgelegt. Die reservierten 2 Byte im VR-Teil sind für zukünftige Weiterentwicklungen des DICOM-Standards.[Nat11b, 7.1.2]

## **A.2. Explizite VR**

Diese Darstellung wird gewählt wenn VR *nicht* vom Typ OB, OW, OF, SQ, UT oder UN ist. Der Unterschied besteht im Feld „Value Length“ Bei der Form von Tabelle A.1 ist dieses Feld 32 Bit lange. Hier beträgt es lediglich 16 Bit [Nat11b, 7.1.2]. Der Grund liegt am erhöhten Speicherbedarf von A.1, da die Länge des Wertes eine undefinierte Länge haben kann.

## **A.3. Implizite VR**

Bei einer impliziten VR Darstellung besteht das Datenelement aus den drei konsekutiven Feldern Tag, Value Length und dem Wert selbst [Nat11b, 7.1.3].

<b>Tag</b>	<b>VR</b>			<b>Value Length</b>	<b>Value</b>
Group # 16-bit unsigned integer	Element # 16-bit unsigned integer	VR 2-byte character String [OB – OW – OF – SQ – UT – UN ]	Reservierter Bereich	32-bit unsigned integer	Gerade Anzahl an Byte. Enthält den Wert des Datenelements. Kodierung abhängig von VT-Typ und Transfersyntax. Wenn die Länge nicht definiert ist wird diese auf „Sequence Delimitation“ limitiert.
2 Byte	2 Byte	2 Byte	2 Byte	4 Byte	Anzahl an Byte entsprechend der „Value Length“ wenn von expliziter Länge

Tabelle A.1.: Darstellung des Datenelements im Speicher wenn VR vom Typ OB, OW, OF, SQ, UT oder UN

Tag		VR 2	Value Length	Value 4
Group #	Element #	VR 2-byte character String 2	16-bit unsigned integer	Gerade Anzahl an Byte. Enthält den Wert des Datenelements. Kodierung abhängig von VT-Typ und Transferyntax.
16-bit unsigned integer	2 Byte	2 Byte	2 Byte	„Value Length“ Byte

Tabelle A.2.: Darstellung des Datenelements für alle anderen VR-Typen

Tag		Value 2	Length	Value
Group # 16-bit unsigned integer	Element # 16-bit unsigned integer	32-bit unsigned integer		Gerade Anzahl an Byte. Enthält den Wert des Datenelements. Kodierung abhängig von VT-Typ spezifiziert in [Nat11c] und Transfersyntax. Wenn die Länge nicht definiert ist wird diese auf „Sequence Delimitation“ limitiert.
2 Byte	2 Byte	2 Byte		„Value Length“ Byte oder undefinierte Länge

Tabelle A.3.: Darstellung des Datenelements für implizite VR.

# B. Installation der Eclipse e4 Umgebung

## B.1. Eclipse

Voraussetzung für Eclipse ist eine bereits installierte Java Virtual Machine. Werden keine Werkzeuge aus dem Java Development Kit benötigt, reicht eine Java Runtime Environment aus. Grundlage für die Plug-in und Rich Client Entwicklung ist eine Eclipse-Installation. Unter <http://www.eclipse.org/downloads/> kann der aktuelle *Eclipse Standard Client* für ein beliebiges Betriebssystem bezogen werden. Bei der Wahl zwischen 32- und 64-Bit muss die Version mit der installierten Java-Variante übereinstimmen, da sonst die Native Libraries nicht geladen werden können. Nach dem Entpacken des Zip-Archivs kann der Client über *eclipse.exe* werden. Abbildung B.1 zeigt das Programmfenster von Eclipse nach dem ersten Ausführen.

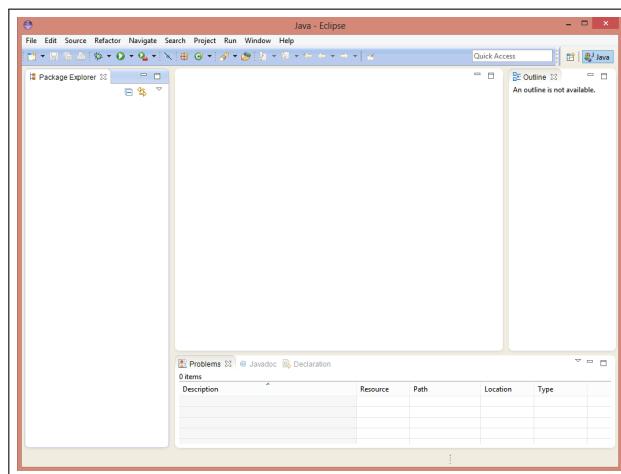


Abbildung B.1.: Standardinstallation eines 32-Bit Eclipse unter Windows 8

## B.2. Eclipse e4 Tools

Eine weitere Voraussetzung ist eine Installation der e4 Tools. Das e4 Projekt ist unter <http://download.eclipse.org/e4/downloads/> mit dem Punkt *Stable Build* zu finden (Abbildung B.2).

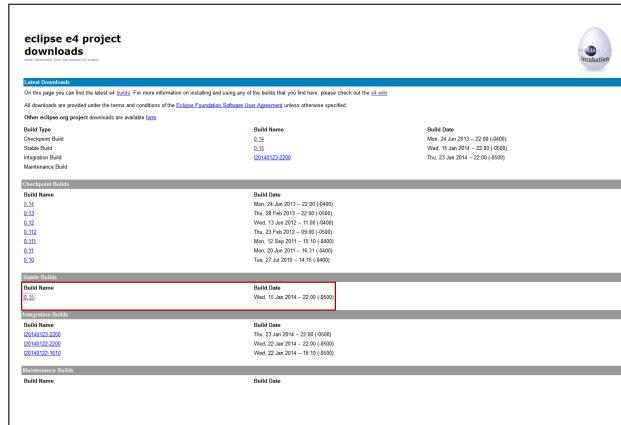


Abbildung B.2.: e4 Projekt Builds

Nach einem Klick auf den aktuellen Build öffnet sich die Seite mit dem Link zum Repository wie in Abbildung B.3 rot markiert.

Dieser Link (<http://download.eclipse.org/e4/downloads/drops/S-0.15-201401152200/repository> - Stand 24.01.2014) muss nun in die Zwischenablage kopiert werden.

## Installation der Eclipse e4 Umgebung

---

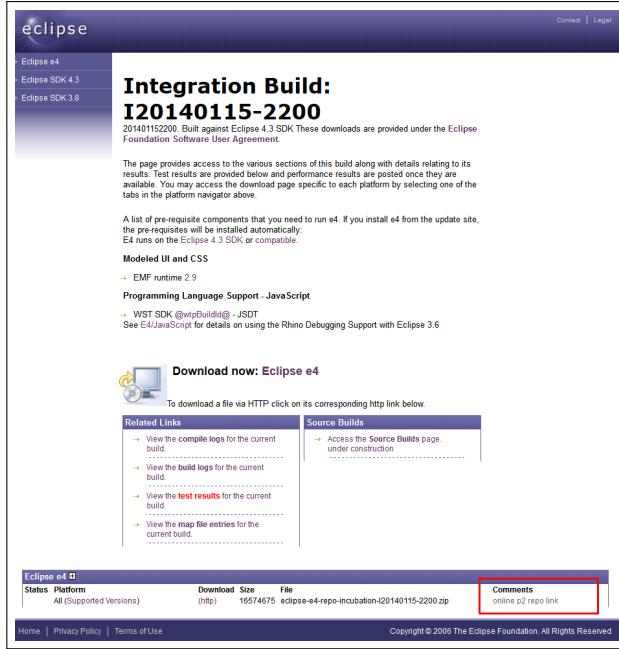


Abbildung B.3.: e4 Repository Link

Nachdem der Link kopiert wurde, kann Eclipse geöffnet werden. Nach einem Klick auf den Menüpunkt *Hilfe* → *Install New Software* öffnet sich ein Fenster mit dem Titel „Available Software“ wie in Abbildung B.4 zu sehen ist.

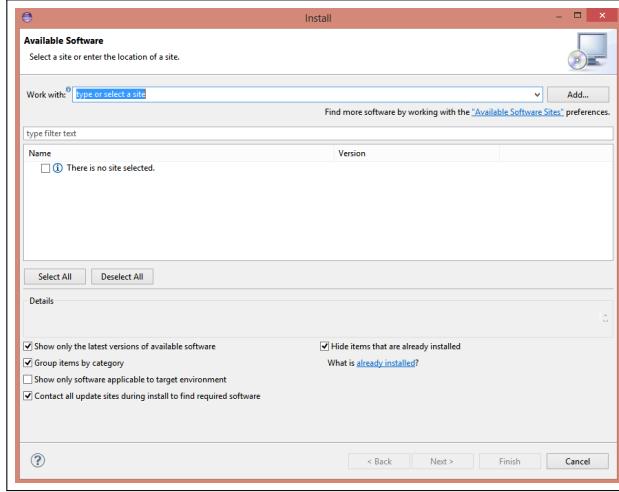


Abbildung B.4.: Installation neuer Software unter Eclipse

Mit dem Button *Add* muss nun der zuvor kopierte Links als Repository angegeben

## Installation der Eclipse e4 Umgebung

---

werden. Der *Name* kann frei vergeben werden und unter *Location* muss der Link zum Repository eingetragen werden. Danach mittels *OK* die Aktion bestätigen.

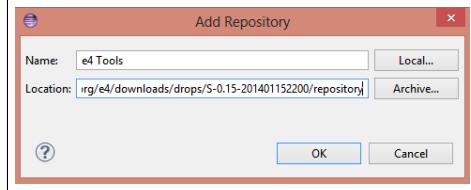


Abbildung B.5.: Angabe des Repository

Nach einem korrekten Eintrag wird die Liste der verfügbaren Software, wie in Abbildung B.6 zu sehen ist, aktualisiert. Hier muss das Paket *Eclipse 4 core tools* samt Unterpakete ausgewählt werden. Mit einem Klick auf *Next* startet die Installationsroutine. Hierbei den Anweisungen auf dem Bildschirm folgen. Nach einer erfolgreichen Installation muss Eclipse neu gestartet werden.

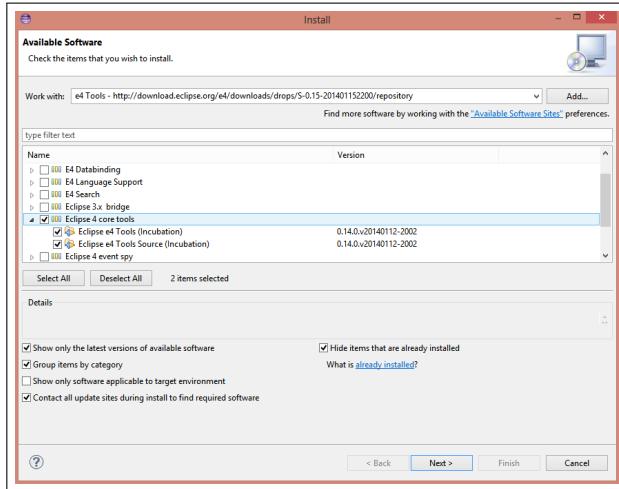


Abbildung B.6.: Auswahl der e4 Tools zur Installation

## B.3. Import der jMediKit Projektdateien

Die Projektdaten befinden sich als auf dem beiliegenden Datenträger. Das Wurzelverzeichnis kann als Eclipse Workspace benutzt werden. Dabei ist darauf zu achten, dass der Ordner */.metadata/.plugins* leer ist. Sollten sich Daten darin befinden, können diese

## Installation der Eclipse e4 Umgebung

gelöscht werden. Eclipse erstellt bei Bedarf die Plug-in-Daten neu. Der Workspace besteht aus den drei Projekten *org.jmedikit.product*, *org.jmedikit.feature* und *org.jmedikit.plugin*. Beim ersten Öffnen ist der Package Explorer leer und die Projekte müssen importiert werden. Nach einem Klick auf *File → Import* erscheint ein Dialog wie in Abbildung B.7.1 zu sehen ist. Bei der Auswahl muss der Punkt unter *General → Existing Projects into Workspace* markiert und mit *Next* bestätigt werden. Im folgenden Fenster (Abbildung B.7.2) muss unter ausgewähltem *Select Root Directory* unter *Browse* das Wurzelverzeichnis von jMediKit angegeben werden. Danach können die verfügbaren Projekte hinzugefügt und mittels Klick auf *Finish* in den Eclipse Workspace importiert werden.

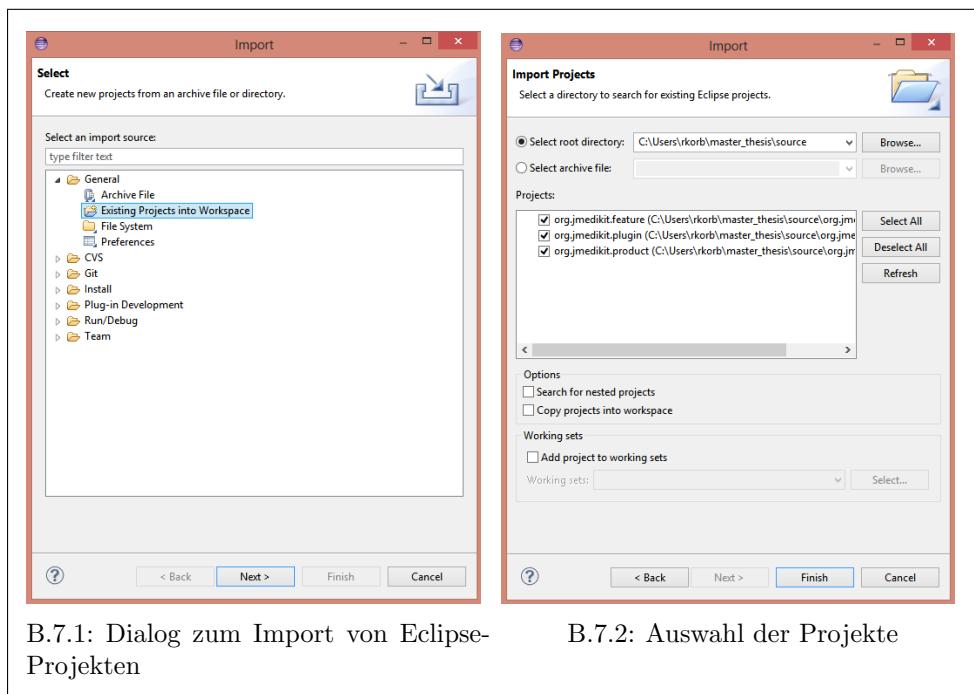


Abbildung B.7.: Vorgang zum Importieren bereits bestehender Projekte

War der Importvorgang erfolgreich, sind die drei Projekte

- *org.jmedikit.product*
- *org.jmedikit.feature*
- *org.jmedikit.plugin*

wie in Abbildung B.8 zu sehen, im Package Explorer vorhanden.

## Installation der Eclipse e4 Umgebung

---

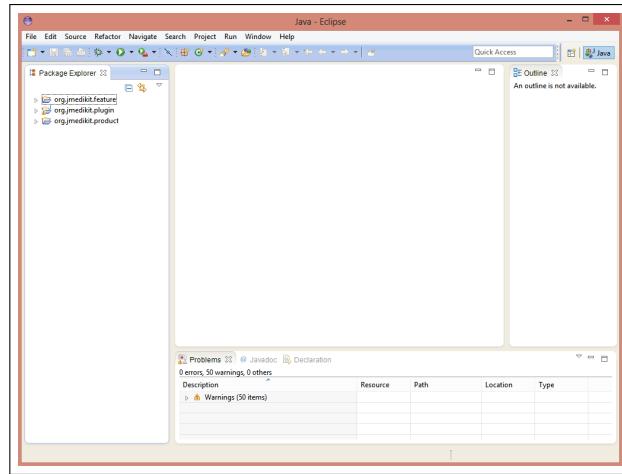


Abbildung B.8.: Der Package Explorer nach dem Importvorgang

Im Projekt *org.jmedikit.product* befindet sich die Datei *jmedikit.product*. Nach einem Doppelklick wird die Datei im Eclipse-Editor geöffnet und zeigt die Grundlegende Anwendungsdefinition von jMediKit und den zugehörigen Einstellungen. Unter dem Tab *Overview* im Bereich *Test* kann jMediKit mit einem Klick auf *Launch an Eclipse application* gestartet werden. Abbildung B.9 hebt die Schaltfläche hervor. Bei einem ersten Start wird jMediKit mit einer Fehlermeldung geschlossen, da von Eclipse noch nicht alle zum Start notwendigen Plug-ins geladen wurden, welche von der jMediKit benötigt werden.

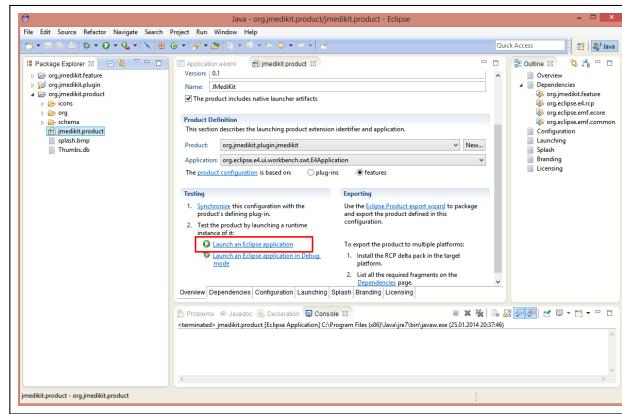


Abbildung B.9.: Der Package Explorer nach dem Importvorgang

Unter *Run → Run Configuration* können die Plug-ins zur Verfügung gestellt werden. Abbildung B.10 zeigt das Konfigurationsfenster. Im linken Teil muss die Product-Datei

## Installation der Eclipse e4 Umgebung

---

jmedikit.product ausgewählt sein. Unter dem Tab *Plug-ins* befindet sich auf der rechten Seite die Schaltfläche *Add Required Plug-ins*. Nach einem Klick auf *Apply* gefolgt von *Run* kann jMediKit gestartet werden.

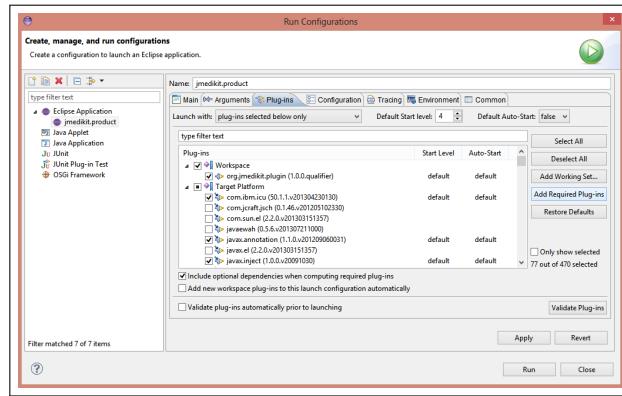


Abbildung B.10.: Konfigurationsfenster zu Anwendungseinstellungen und Anwendungsstart