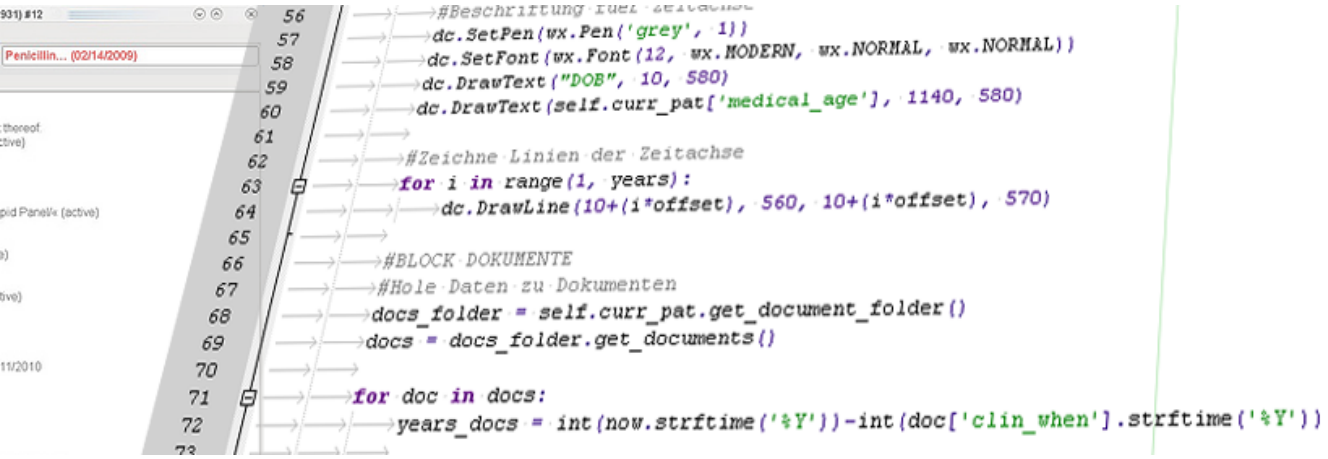
Fachhochschule Nordwestschweiz
Hochschule für Life Sciences

Spezialisiertes KIS für das LKGS-Zentrum am Universitätsspital Basel

Bachelorarbeit Nr. 294



Achim Bauer betreut durch Prof. Dr. Dominique Brodbeck

Autraggeber: Prof. Dr. Dominique Brodbeck

Institut für Medizinal- und Analysetechnologie
Hochschule für Life Sciences

Muttenz, Juli 2010

Vertraulich

Autor
Achim Bauer
Baselmattweg 53
CH-4123 Allschwil
+41 79 568 42 47
achim.bauer@students.fhnw.ch

Auftraggeber und Betreuer
Prof. Dr. Dominique Brodbeck
Gründenstrasse 40
CH-4132 Muttenz
+41 61 467 46 97
dominique.brodbeck@fhnw.ch

9. Juli 2010, Muttenz Schweiz

Zusammenfassung

Im Behandlungszentrum für Lippen-Kiefer-Gaumen-Spalten am Universitätsspital in Basel werden jährlich ca. 800 Patienten behandelt. Ihr Krankheitsbild erfordert eine sehr lange Behandlungszeit, welche bis zu 25 Jahren dauern kann; also von Geburt an bis zum Wachstumsabschluss. Dabei werden von vielen verschiedenen Spezialisten Eingriffe und Behandlungen vorgenommen, aber auch Fördermassnahmen im Bezug auf Sprache und Bewegung kommen zum Zug. Durch die vielen in die Behandlung involvierten Personen entsteht eine Verlaufsdokumentation, die einerseits sehr heterogen und vorwiegend papierbasiert ist. Ein weiteres Problem stellt die räumliche Trennung zwischen den behandelnden Stellen dar, welche es den Personen nicht ermöglicht, jederzeit auf die vorhandenen Daten zuzugreifen.

Daher bietet es sich an, Informationen über Patienten und deren Behandlung in Krankenhausinformationssystemen abzulegen. Diese sind vor allem darauf ausgelegt, textuelle Daten zu verarbeiten; Inhalte wie Bilder oder Videos werden oftmals nicht unterstützt. Gerade bei langen Zeiträumen wäre es aber hilfreich, diese mit der Krankengeschichte im KIS zu verknüpfen. Das Ziel dieser Arbeit ist es, den Aspekt der Visualisierung langfristiger Patientengeschichten mit der Unterstützung durch multimediale Inhalte zu verbinden und einen Prototypen zu entwicklen, welcher beide Problemstellungen löst.

Dazu wurden nach einer Sondierung bestehender Systeme Kriterien ermittelt, mit welchen sich diese Systeme in einer Gegenüberstellung bewerten lassen konnten. Nach dieser Evaluation wurde das am besten geeignete für eine Weiterentwicklung genauer untersucht. Durch der Aufgabenstellung entnommene Informationen und Gegebenheiten des vorhandenen Anwendungsfalles wurden Use Cases ermittelt, deren Grundidee in ein Interaktionskonzept eingeflossen ist. Dieses Konzept wiederum bildete die Basis für die Weiterentwicklung des Systems. Nicht alle Punkte des Interaktionskonzepts konnten umgesetzt werden. Trotzdem ist es möglich, die vorhandenen Dokumente einer Patientenakte auf einer Zeitachse anzuordnen und so die vorhandene Patientengeschichte zu visualisieren. Die Arbeit hat gezeigt, dass frei verfügbare Implementationen von KIS an spezielle Anwendungsfälle wie den des LKGS angepasst werden können und ist als Grundlage für eine weiterführende Adaptierung denkbar.

Summary

At the cleft lip and palate-treatment centre, located at the University Hospital in Basel, over 800 patients are medicated every year. Their clinical picture needs a very long time for treatment that lasts up to 25 years. So patients are treated by birth until being full-grown. In this long period, many specialists do surgery as well as dental treatments, but patients also get aid concerning speech and movement. Due to many people being involved in the treatment, documentation of progression becomes heterogenous and mainly paper-based. Another problem, which is regional separation, leads to not having access to available data at any time.

This requires storage of patient- and treatment-related data in an electronic health record system. Their main purpose is to handle mostly text-based documents, though files like pictures or videos are often not supported. But in case of long periods, it might be helpful to attach these documents with the events in the patients's history. This thesis aims to cover the aspect of visualising long-term patient history on one hand and attaching its data with multimedia-based content on the other and to develop a prototype which is able to solve both problems.

For this purpose, existing systems have been examinated to determine several criteria for a checkup. After this, the best of them has been analysed accurately for futher development. Based on few informations of the task, use caes were developped and a concept of interaction has been established. Not all parts of that concept could be implemented. However, you got the possibility to arrange existing documents of an electronic health record on a timeline and therefore visualise the patient's history. This thesis revealed that adaption of freely available open source health record systems to special applications is possible and might get considered for continuative development.

Inhaltsverzeichnis

1. Einleitung

Am Behandlungszentrum für Lippen-Kiefer-Gaumen-Spalten des Unispitals Basel werden jährlich ca. 800 Patienten behandelt. Deren Behandlung beginnt mit der Geburt und endet mit dem Wachtumsabschluss, was zu einer sehr langen Behandlungszeit von bis zu 25 Jahren führt. Während dieser langen Betreuung werden zum einen chirurgische Eingriffe sowie zahnärztliche und kieferorthopädische Behandlungen vorgenommen, anderseits finden spezielle Fördermassnahmen in Bezug auf Sprache und Bewegung statt. Die entstehende Verlaufsdokumentation wird vorwiegend papierbasiert durchgeführt und durch die grosse Anzahl der involvierten Personen sehr heterogen. Zudem können die externen Behandlungsstellen oftmals nicht auf die vorhandenen Daten zugreifen, da diese nicht von entfernt verfügbar sind. Allgemein sind die Daten in der aktuellen Form nicht übersichtlich dargestellt; eine Interpretation der Daten kann bisher nur ungenügend erfolgen.

Das Ziel dieser Arbeit ist es, ein Informationssystem zu realisieren, welche die langfristige Visualisierung der Patientengeschichten ermöglicht. Unterstützend sollen dabei Dokumente angezeigt werden, die im Verlauf der Behandlung erstellt worden sind. Die Dokumente erlauben dienen dabei der Einordnung der vorgenommenen Eingriffe.

Die Entwicklung dieses Informationssystems soll auf Basis eines bestehenden, frei verfügbaren und offenen Krankenhausinformationssystems erfolgen. Ein solches System sorgt vor allem für die Erfassung patientenbezogener Informationen und deren Bereitstellung am jeweils benötigten Ort. Das KIS unterstützt also die Versorgung des Patienten von der Aufnahme bis zur Entlassung.

In einem ersten Schritt werden verfügbare Systeme sondiert und mögliche Kriterien für eine Evaluation ermittelt. Aus der Aufgabenstellung hervorgehende Aspekte sowie weitere Informationen des Auftraggebers ermöglichen eine Auswahl und Gewichtung der ermittelten Kriterien. Mit diesen Parametern werden einige ausgewählte Systeme getestet und auf ihre Tauglichkeit überprüft. Das System, welches sich am Besten für eine Weiterentwicklung eignet, soll im darauffolgenden Kapitel näher analysiert werden. Da diese Arbeit auch dazu dienen kann, Erweiterungen für andere Anwendungsfälle zu erstellen, widmet sich ein Abschnitt des erwähnten Kapitels der Erstellung eines Plugins für das System. Dabei wird jeder Schritt erklärt und anhand von kleineren Codeabschnitten erläutert.

Der Umsetzung der Aufgabenstellung wird ein eigenes Kapitel gewidmet, welches ausgehend von der Zielsetzung ein von Use Cases[1] gestütztes Interaktionskonzept beschreibt. Diesem Konzept folgt eine nähere Analyse der Datenstrukturen, bevor schliesslich die Umsetzung anhand des erstellten Quellcodes erläutert wird. Am Ende werden in der Schlussfolgerung die Vorgehensweise noch einmal reflektiert und die Ergebnisse, welche die Arbeit hervorgebracht hat, zusammengefasst. Ein Ausblick auf kommende Anwendungen und Möglichkeiten schliesst die Arbeit ab.

[1] siehe Abschnitt 5.1

Auf das im Anwendungsfall beschriebene Krankheitsbild wird nicht eingegangen. Ebenso werden auf Ausführungen bezüglich Datensicherheit verzichtet. Dia Annahme, dass keines der untersuchten Systeme den Datenschutz missachtet, hat sich im Laufe der Arbeit bestätigt. Im Anhang finden sich detaillierte Informationen der Evaluation. Die anschliessend folgenden kompletten Listings erlauben es zusammen mit einer Installationsanleitung des ausgewählten Systems, den umgesetzten Prototyp testen zu können.

2. Bestehende Systeme

Aus der Aufgabenstellung geht hervor, dass bestehende, offene und zugleich freie Implementationen von Krankenhausinformationen miteinander verglichen und anhand diverser Kriterien bewertet werden müssen. Da in dieser Sparte fast keine Vorkenntnisse vorhanden waren, mussten bestehende Systeme zuallererst ermittelt werden. Eine Websuche mit dem Suchbegriff „free open source emr systems“ hat eine Auflistung[?] bei Wikipedia geliefert, welche eine gute Übersicht über bestehende Systeme bietet. Eine erste Eingrenzung wurde dadurch möglich.

Weitere Links in den Suchergebnissen führten in einem Fall zu einem in der Liste vorhanden Einträge, also zur Website des Systems. Ebenfalls auf der ersten Seite der Suchergebnisse war ein Link zu einem Blog, welcher sich mit biomedizinischer Informatik beschäftigt. Der gefundene Artikel[1] zeigte eine ähnliche Auflistung. Es galt nun, aus den bestehenden zwei Listen nur noch eine hervorzubringen. Da gewisse Systeme sowohl in der ersten als auch in der zweiten Auflistung vorkamen, galten diese als sichere Kandidaten für eine weitere Einstufung.

Um den nächsten Schritt durchführen zu können, sollten mit sogenannten „Killerkriterien“ im Vorfeld ungeeignete Implementationen von der Auswahl werden. Die in der Aufgabenstellung definierten Anforderungen stellen klar, dass nur Systeme, die bestimmte Eigenschaften aufweisen, in der Evaluation berücksichtigt werden. Dies sind die erwähnten Killerkriterien:

- kostenlos
- quellcodeoffen

Beide Auflistungen liefern annähernd 30 Einträge, wobei wie erwähnt viele Einträge doppelt vorhanden sind. Jeder Eintrag der beiden Listen besteht aus einer kurzen Beschreibung des Systems und einem Link zur Website des Entwicklers. Jede Website wurde besucht und die Informationen der Systeme begutachtet. Viele der verfügbaren Systeme konnten die oben genannten Kriterien nicht erfüllen und wurden somit aussortiert.

2.1. Auswahl der Systeme

Um die Auswahl der Systeme weiter eingrenzen zu können, wurden die Einträge, welche in beiden vorhandenen Auflistungen vorkamen, in die nächste Runde der Sondierung aufgenommen. Dieser Schritt sollte das Teilnehmerfeld zum einen weiter eingrenzen, zum anderen wird unter der Annahme, dass Systeme welche in beiden Listen auftauchen, einen höheren Bekanntheitsgrad aufweisen, gefiltert. Des Weiteren kann angenommen werden, dass sich die doppelt vorhandenen Systeme durch bessere Funktionalität hervorgetan haben.

Elf Systeme haben die Killerkriterien erfüllt und wurden in beiden Artikeln gelistet; doch auch diese elf haben sich nicht bis zur Evaluation durchgesetzt. Die nachfolgenden sieben Systeme wurden weiter geprüft:

- OpenEMR
- FreeMed
- GNUmed
- Medical
- OpenMRS
- Tolven
- TORCH

Die vier Systeme, die nicht weiter berücksichtigt wurden:

- OpenVista: Das in den USA am weitesten verbreitete System richtet sich an Grosskrankenhäuser, welche weit mehr Funktionalität benötigen als die Verwaltung von Patienten. Dabei haben sich viele Untersysteme entwickelt, welche es nicht einfach machen, Sparten von einander abzugrenzen. Der Aufwand, diese Teilsysteme genauer zu überprüfen, hätte zuviel Zeit gekostet.
- OSCAR: Ein auf das kanadische Gesundheitswesen zugeschnittenes System, welches neben der Verwaltung viele Funktionen für die Krankenkassenanbieter (z.B. Rechnungserstellung) bereitstellt. Dadurch ist die Applikation nicht wirklich für den Einsatz in Europa geeignet.
- Indivo: Dieses System erlaubt es Patienten, selbst eine eigene Patientenakte zu führen. Dies entspricht generell nicht den Anfordungen. Zudem beginnt die Behandlung bei Spaltpatienten bereits bei der Geburt, wo also noch niemand der Patienten in der Lage ist, eine elektronische Patientenakte zu führen.
- ClearHealth: Diese Software hat viele Features zu bieten, ist aber nicht kostenlos zu bekommen. VOn ihr wird deshalb abgesehen.

Im nächsten Kapitel wird erklärt, wie der Kriterienkatalog für die Evaluation zustande gekommen ist und welche Überlegungen bei den einzelnen Kriterien eine Rolle gespielt haben.

3. Evaluation EHR-Systeme

3.1. Ermittlung der Evaluationskriterien

3.1.1. Methoden zu Ermittlung

Um mögliche Bewertungskriterien ermitteln zu können, wurde vor allem auf einen explorativen Ansatz gesetzt. Zu jedem Eintrag der Liste der ausgewählten Systeme wurden nach dem Besuch der Herstellerwebsite Bemerkungen notiert. Viele der Systeme bieten direkt auf der Website eine Möglichkeit, das System in Aktion zu sehen und auszuprobieren. Dank dieser Live-Demos war es möglich einen guten Einblick in die einzelnen Systeme zu erlangen. Finden sich keine Demos, liefern Screenshots und/oder Videos Anhaltspunkte zum Funktionsumfang und der Bedienung. Sonstige bestehende Hinweise auf den Funktionsumfang sind vor allem als Aufstellungen oder in Texten vorhanden.

Einige Kriterien haben sich durch diesen Ansatz bereits hervorgetan. Hinweise auf weitere mögliche Kriterien ergaben sich aus einem Artikel[2], welcher Ratschläge bei der Wahl eines KIS erteilt. Eine enthaltene Checkliste möglicher Kriterien deckte sich mit selbst ermittelten Kriterien und erwies sich auch als Kontrolle. Im folgenden Abschnitt werden die Bewertungskriterien kurz beschrieben und auf die wichtigsten Punkte hingewiesen.

3.1.2. Bewertungskriterien

- Client-/Server-System: Der Aufgabenstellung entsprechend kommt nur eine Client/Server-Architektur in Frage, denn nur so ist die vorhandene räumliche Trennung zwischen den behandelnden Institutionen zu bewerkstelligen. Zudem muss es möglich sein, dass mehrere Anwender zur gleichen Zeit mit den vorhandenen Daten arbeiten.
- Support: Inwiefern kann der angebotene Support beurteilt werden? Hierbei geht um die Unterscheidung von kommerziellem Support und Community-basiertem, kostenlosen Support.
- ICD[1]-Anbindung: Obwohl vorgesehen ist, ein System für ein festgelegtes Krankheitsbild weiterzuentwickeln, soll die Möglichkeit bestehen, zu einem späteren Zeitpunkt durch Anpassungen auch andere Krankheitsbilder über längere Zeit zu beobachten und zu visualisieren. Eine Anbindung an den ICD-Standard verhindert dabei willkürlich vergebene Namen für Krankheitsbilder, indem diese vom Standard vorgegeben sind.
- HL7[2]-Anbindung: Um das angepasste System in eine bestehende Infrastruktur, bspw. ein KIS zu intergrieren, kann sich die Möglichkeit einer Anbindung an den HL7-Standard als sinnvoll erweisen.

[1] siehe Glossar Seite 41
[2] siehe Glossar Seite 41

- DICOM[3]-Schnittstelle: Diese Bedingung wurde von der Annahme gestützt, dass sich manche Systeme direkt mit bildgebenden Modalitäten in Verbindung setzen können. Im Laufe der Evaluation hat sich aber herausgestellt, dass die Programme - wenn überhaupt - nur einen DICOM-Viewer enthalten, welcher bereits vorhandene Daten bspw. eines PACS[4] abrufen kann.

- Bereits umgesetzte Lösungen: Ein breit abgestütztes Netzwerk mit bereits implementierten Lösungen hätten an dieser Stelle Parallelen gezogen werden könnnen. Viele Hersteller preisen ihre Lösung als die Beste an, jedoch finden sich bei den meisten keinerlei Hinweise, durch wen und wie oft die Systeme eingesetzt werden.

- Projekt ist „lebendig“: Wenn innerhalb eines gewissen Zeitraumes (höchstens ein Jahr) keine Fortschritte zu verzeichnen sind, kann man davon ausgehen, dass die Entwicklung nicht mehr weitergeführt wird oder gar eingestellt wurde.

- Import bestehender Daten: Dieser Punkt soll die Frage beantworten, ob und wie bestehende Daten, vor allem aus Datenbanken, importiert werden können.

- Spezieller Zugang für Patienten: Falls Patienten die Möglichkeit haben, über einen speziellen, für die behandelnden Personen nicht offenen Zugang die eigene Akte einzusehen, können beim entsprechenden System Punkte gegeben werden.

- Decision-Support-Tools: Unter DST versteht man Systeme, welche Informationen (Wissen) beinhalten und dieses dem Anwender auf eine solche Art und Weise präsentieren, dass dieser leichter eine Entscheidung treffen kann. Oftmals werden auch Lösungen präsentiert. Bietet ein untersuchtes System die Möglichkeit, die Arbeit des Anwenders mittels Meldungen und Erinnerungen zu unterstützen, kann bereits von einem DST gesprochen werden.

- Multimedia-Unterstützung: Eine der wichtigsten Voraussetzungen, da der Behandlungsverlauf mit Hilfe von vorhandenen Daten visualisiert werden soll. Es soll also möglich sein, Dateien wie etwa Bilder, Videos oder PDF-Dokumente einem Patienten und Fall zuordnen zu können.

- Zugriff über mobile Geräte: Gegenwärtig können durch immer leistungsfähigere Smartphones o.ä. viele Arbeiten auch unterwegs erledigt werden. Die vorhandenen Daten könnten auf diese Weise eingesehen und/oder bearbeitet werden. Hier gilt es zu unterscheiden, ob für das mobile Gerät ein eigener Client verfügbar ist oder ob über einen Browser auf die Daten zugegriffen werden kann.

- Erweiterbarkeit: Ein zentraler Punkt, soll doch das System später an eigene Anforderungen angepasst werden können. Die in Abschnit 2 vorgestellten Kriterien grenzen die vorliegenden Systeme bereits so weit ein, dass davon ausgegangen wird, dass jedes der Systeme erweiterbar ist. Je nachdem, welcher Aufwand betrieben werden muss, um Quellcode zu verändern, werden hier die Punkte verteilt. Eine Gewichtung könnte sich bspw. an der Anzahl zu verändernden Dateien richten.

- API/Schnittstelle: Dieses Kriterium soll darüber Auskunft geben, ob Module über eine Schnittstelle auf Funktionen des Grundsystems zugreifen können und ob man

[3] siehe Glossar Seite 41
[4] siehe Glossar Seite 41

diese Schnittstelle ansprechen kann. Das Grundsystem wird in Form vorhandener Bibliotheken zur Verfügung gestellt und lässt sich als „Blackbox“[5] nutzen.

- Lizenz: Bei den zu untersuchenden Systemen handelt es sich um Open-Source-Software, welche generell mit einer Lizenz verteilt wird, die es erlaubt, das System zu verändern und später weiterzugeben, inkl. einer kommerziellen Variante. Liegt das Modell LGPL vor, müssen die Quellcodes nach einer Anpassung nicht mit der Software mitgeliefert werden. Die GPL-Lizenz jedoch schriebt genau dies vor. Dieses Kriterium soll nur der Information dienen und erhält somit weder eine Bewertung noch eine Gewichtung.

Gut die Hälfte der Kriterien lassen sich durch eine einfache ja/nein-Antwort zuweisen. Bei der anderen Hälfte jedoch müssen die vorhandenen Gegebenheiten mit einer Mengen- oder Qualitätsangabe beurteilt werden. So hat bspw. die Frage nach dem angebotenen Support eine Wertung von 1 (wenig/nur kommerziell) bis 3 (viel/kommerziell und kostenlos durch Community) erbracht. Die Art der Gewichtung jedes einzelnen Kriteriums und seine Gewichtung für die Gesamtwertung sind als Tabelle in Anhang A.1 zu finden.

3.2. Evaluation der Systeme

Die Evaluation der Systeme wurde auf einem neu aufgesetzten PC mit Windows XP (SP3) durchgeführt. Wie in Abschnitt 3.1.1 beschrieben können viele Systeme direkt im Browser ausprobiert werden; bei den meisten anderen lassen sich jeweils ein Server und ein Client herunterladen. Die sieben Systeme, welche definitiv in die Evaluation aufgenommen wurden, sind nachfolgend kurz beschrieben. Details zu vorhandenen oder fehlenden Funktionen der Systeme sind in in Anhang A.1.2 zu finden.

[5] Abschnitt 4.6

3.2.1. OpenEMR

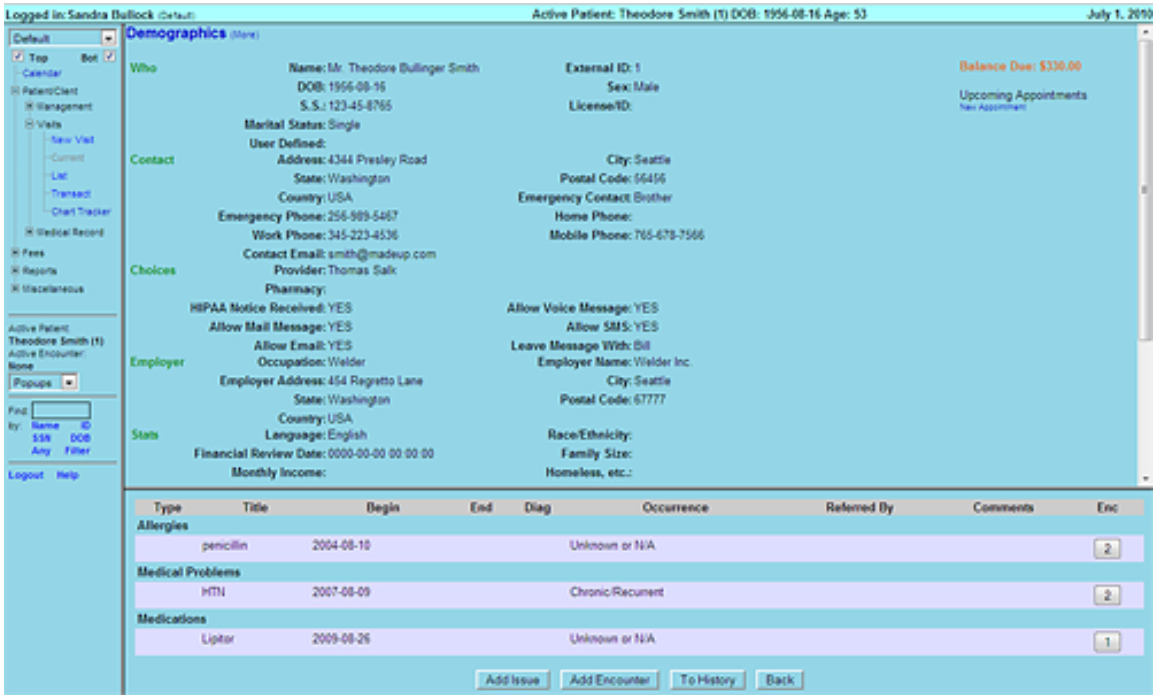


Abbildung 3.1.. OpenEMR: Patientenakte mit Übersicht über Allergien, Medikation und chronischen Problemen

OpenEMR ist eines der Systeme, welches den Browser als Client nutzt. Auf der Serverseite kommt das bekannte und oft eingesetzte [6]-Paket zum Zug. OpenEMR lässt sich zusammen mit diesem Paket herunterladen, was die Installation sehr einfach macht. Die Applikation ist in viele PHP-Skripts aufgeteilt, deren Inhalte und Daten über verschiedene Frames im Browser zu einer einheitlichen Oberfläche zusammengesetzt werden.

Das Hauptmenü setzt sich aus einer Baumstruktur zusammen, was leider nicht zur Übersicht beiträgt. Generell finden sich bei OpenEMR eher Funktionen administrativer Art. Die Patientendaten sind nur als Auflistungen vorhanden und verstärken dabei den Eindruck, dass die Darstellung nicht überschaubar erscheint. Ein Patientenfoto wird auch nicht angezeigt.

[6] siehe Glossar Kapitel 11

3.2.2. FreeMed

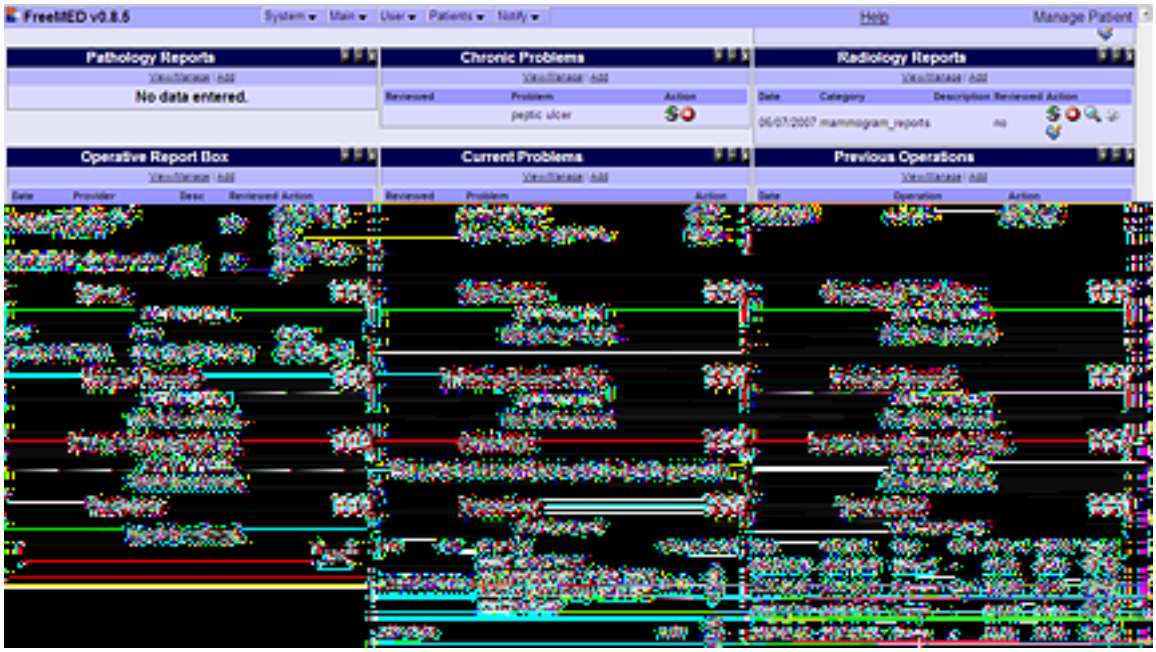


Abbildung 3.2.. FreeMed: Patientenakte mit allerlei Informationen

Auch FreeMed ist ein browserbasiertes System, welches die die XAMPP-Architektur nutzt. Ist eine XAMPP-Installation bereits vorhanden, wird der FreeMed-Ordner einfach in das /htdocs-Verzeichnis des Pakets gelegt. Danach wird mit einem Aufruf im Browser die Installation gestartet. In der Benutzung kann FreeMed nicht überzeugen. Wie bei OpenEMR werden die benötigten Information aus vielen kleinen Fenstern zusammengesetzt. Deren Anordnung lässt sich zwar verändern, aber sich nur die gewünschten Informationen anzeigen zu lassen ist nicht möglich.

Auch bei diesem System scheinen viele der Komponenten zusammengewürfelt. Zu jeder Sparte, welche in einem Fenster vertreten ist, lassen sich vorhandene Daten anzeigen/-verändern oder neue hinzufügen. Bei gewissen Details lassen sich Dokumente anhängen, bei anderen nicht. Auf einen Blick ist dies aber nicht ersichtlich. Eine Möglichkeit, Dokumente an eine Akte zu heften, bietet sich also nicht bei jeder Sparte.

3.2.3. GNUmed

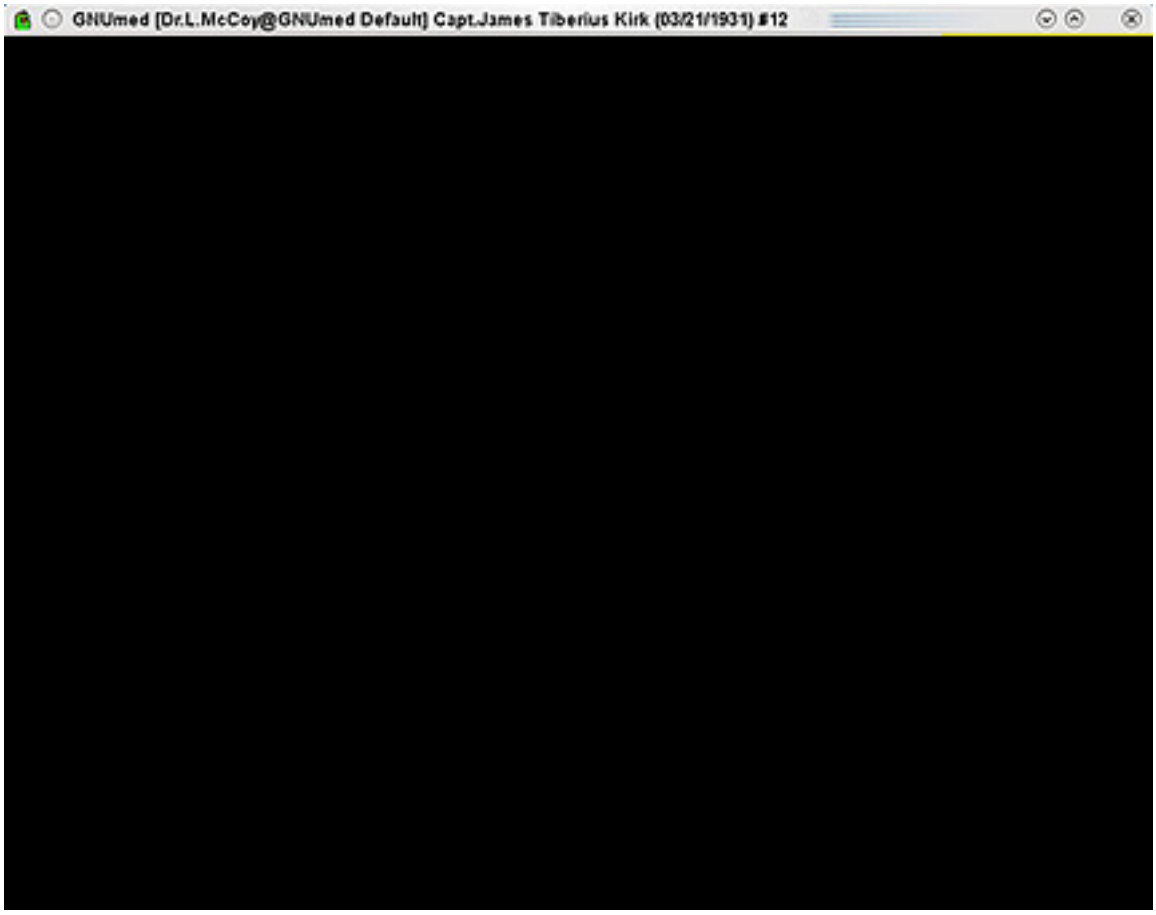


Abbildung 3.3.. GNUmed: Baumansicht einer elektronischen Patientenakte

GNUmed ist komplett in Python aufgebaut und stellt sich aus einem Client und einem Server zusammen, welche beide getrennt zum Download verfügbar sind. Beide Teile lassen sich ohne Voraussetzungen installieren, vorausgesetzt es sind die vollständigen Installationspakete[7]. Auch wenn GNUmed zuerst nicht klar strukturiert erscheint, lässt sich doch erahnen, über welchen Tab man zu welchen Informationen gelangt. Durch Profile und viele sonstige Einstellungsmöglichkeiten lässt sich die Applikation gut an die gewünschte Arbeitsumgebung anpassen.

Mit der standardmässigen deutschen Übersetzung findet man sich schnell zurecht; ausserdem werden an vielen Stellen der Applikation Tooltips verwendet. Das Erstellen neuer Patienten, zugehöriger Episoden und Dokumente funktioniert in dieser Applikation am Besten. Bereits hinzugefügte Dokumente werden mit jeweils systemeigenen Programmen geöffnet. GNUmed liefert also bspw. keine Bildbetrachtungs-Fähigkeiten.

[7] Die Pakete enthalten „full“ in ihrem Dateinamen. Details siehe Anhang mit Installationsanleitung

3.2.4. Medical

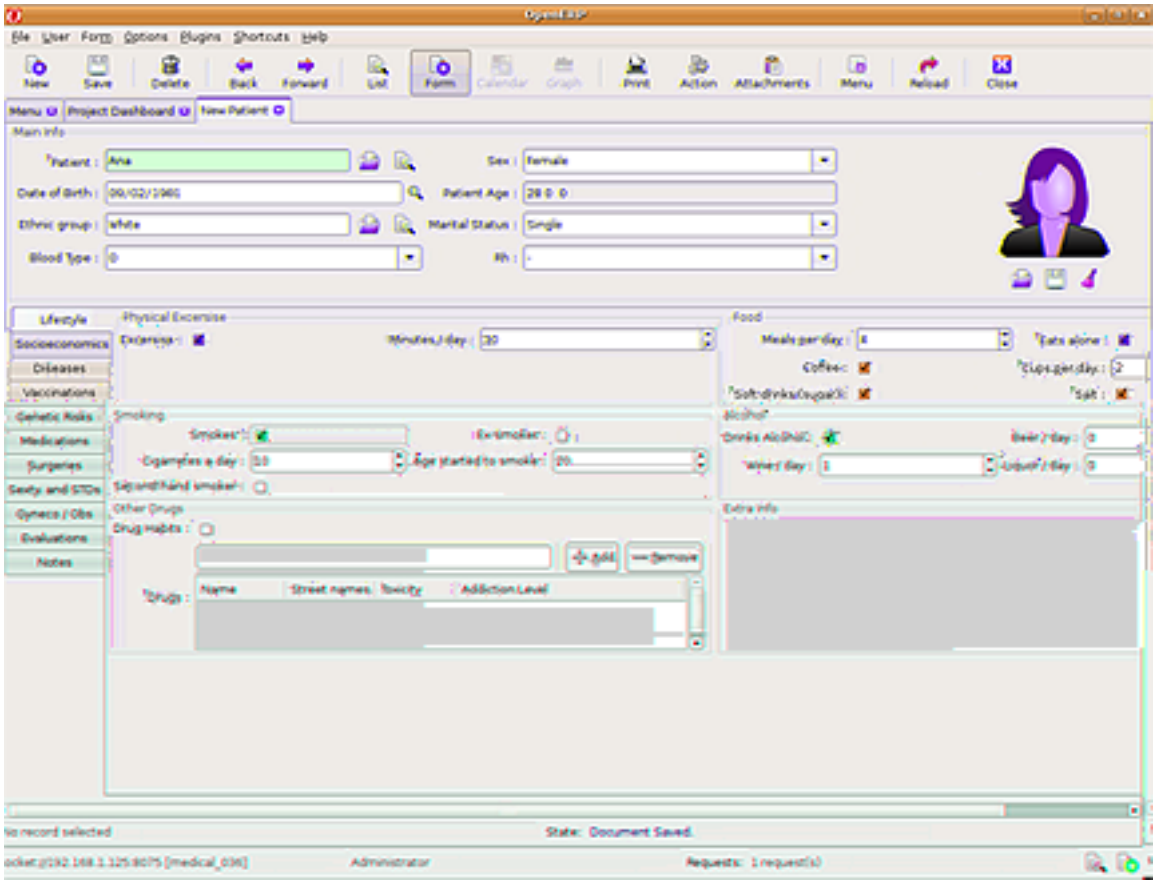


Abbildung 3.4.. Medical: Maske zum Erstellen eines neuen Patienten

Diese Applikation erscheint als Plugin für ein sehr bekanntes und verbreitetes ERP-System, Open ERP. Es gilt also zunächst, diesen doch ziemlich grossen Brocken zu installieren um anschliessend die Medical-Module in das System einzubauen. Der Vorgang ist sehr gut dokumentiert und hat gut funktioniert. Trotzdem nimmt das zugrundeliegende System noch immer die Überhand ein. Das Medical-Modul wirkt versteckt und seine Handhabung erweist sich als nicht so leicht wie bei anderen. Auch nach längerem Versuchen ist nicht klar, wie man einem Patienten Dokumente zuweisen kann.

3.2.5. OpenMRS

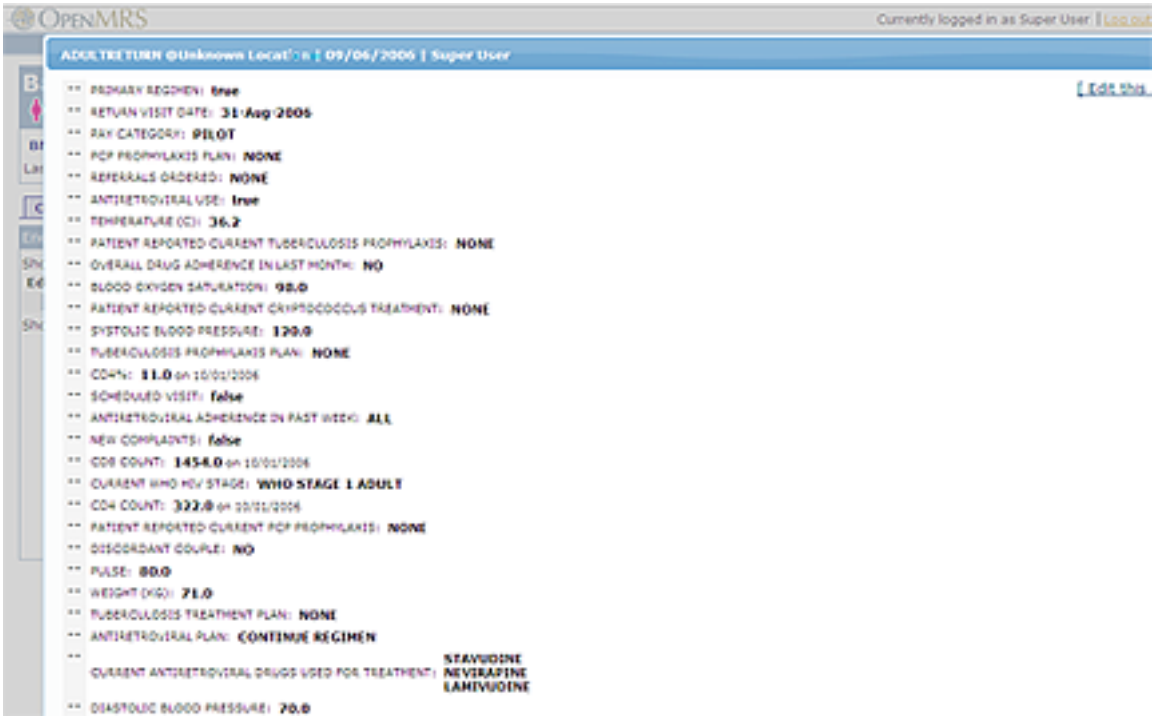


Abbildung 3.5.. OpenMRS: Detailansicht eines Episodenberichtes

Dieses KIS ist eines der wenigen, welche in einem wissenschaftlichen Artikel auftauchen. In ihrem Paper[4] beschreiben Anokwa et al., wie OpenMRS in einer kleinen Krankenstation in einem ländlichen Gebiet in Ruanda eingesetzt wird. OpenMRS ist speziell auf Gesundheitssysteme in der dritten Welt ausgelegt, wo Gelder für die Entwicklung und Support von KIS schlicht fehlen. Bis heute wurde das System in über 20 Ländern implementiert. Vor allem eine spezielle, unabhängige Datenbankstruktur soll dabei helfen, Informationen rasch anzuzeigen und analysieren zu können.

Als Java-basierte Webapplikation nutzt auch OpenMRS den Browser als Client. Auch wenn sich die Benutzeroberfläche hier übersichtlicher zeigt als bei den anderen getesteten Browser-Systemen, scheint die Verwendung von OpenMRS auch eher auf eine Nutzung von rein textuellen Daten abzuzielen. Es besteht zwar die Möglichkeit, Graphen aus vorhandenen Daten zu erstellen, nur wird dabei eine Verwaltung von Dokumenten nicht ermöglicht.

3.2.6. Tolven

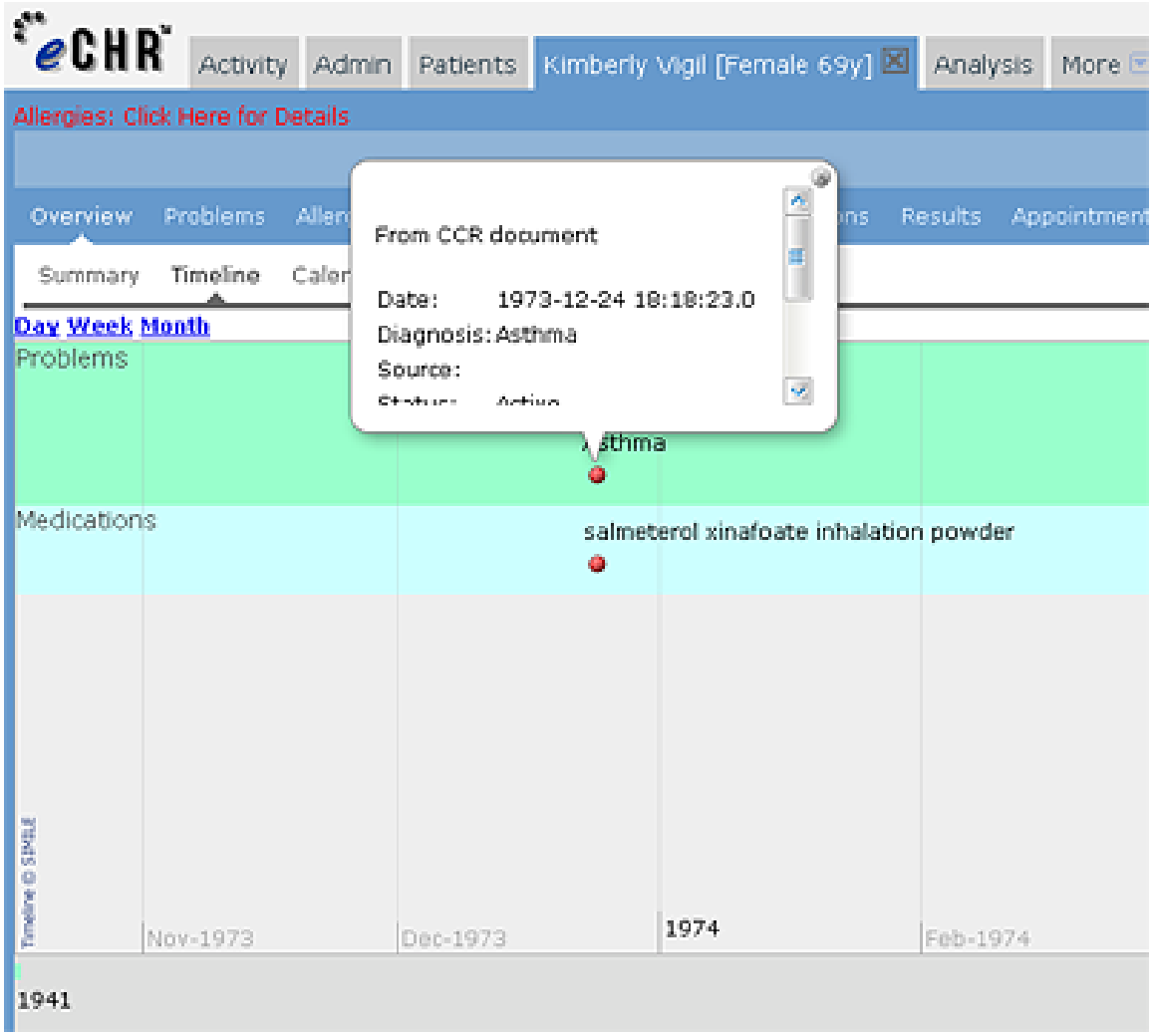


Abbildung 3.6.. Tolven: Patientenakte mit Timeline

Vom Aufbau her gleicht das ebenfalls java-basierte Tolven stark dem zuletzt vorgestellten OpenMRS. Über Tabs werden einzelne Bereiche der Applikation aufgerufen. Auch hier finden sich Übersichten welche durch viele und anpassbare Graphen gestützt werden. Sehr interessant ist die Möglichkeit, sich eine Timeline anzeigen zu lassen. Über den ganzen Behandlungszeitraum verteilt werden dabei vorgenommene Untersuchungen und deren Ergebnisse als Punkte angezeigt. Beim Anwählen eines solchen Punktes werden die Ergebnisse angezeigt.

Eine solche Funktion deckt sich in weiten Teilen mit Vorgaben aus der Aufgabenstellung, deswegen gilt diesem System ein besonderer Augenmerk. Eine weitere Besonderheit: Tolven ist die einzige Applikation welche zwischen klinischen und persönlichen Krankenakten unterscheidet. Wer davon allerdings Nutzen ziehen soll, ist nicht klar.

3.2.7. TORCH

Der Entscheid für dieses System für die Evaluation hat sich im Nachhinein als grosser Fehler erwiesen. Die Website des Projekts, welche mittlerweile [Stand: 06.07.2010] nicht mehr online ist, hat zwar Informationen liefern können. Dennoch ist es dem Autor entgangen, dass bestehende Versionen der Software bereits seit mehreren Jahren kein Update mehr erhalten haben. Beim Versuch, dennoch prüfen zu können, welcher Funktionsumfang erwartet werden kann, wurden diese Umstände bewusst. Da bereits grosse Teile der Evaluation abgeschlossen waren, wurde das System trotz allem in der Liste beibehalten.

3.3. Anwendungsfall

Auf einen Anwendungsfall wurde in dieser Phase verzichtet, da nur der reine Funktionsumfang ermittelt werden sollte und nicht etwa, wie einfach und schnell man bspw. einen Patienten erstellen kann. Im Nachhinein betrachtet hätte eine kleine Aufgabenstellung sicher mehr systematisches Vorgehen in die Evaluation gebracht. Aufgrund der Heterogenität der Systeme aber wäre es schwierig gewesen, genau eine solche Aufgabe zu stellen. Dies sollte in Anbetracht des explorativen Ansatzes berücksichtigt werden, welcher die Ermittlung der Kriterien und die Evaluation vereint hat.

3.4. Ergebnisse der Evaluation

Die Evaluation zeigt leider kein eindeutiges Ergebnis. Gleich drei Systeme erhalten 55 von 63 möglichen Punkten, und auch die Plätze 4 und 5 sind mit einer Punktzahl von über 50 vertreten.

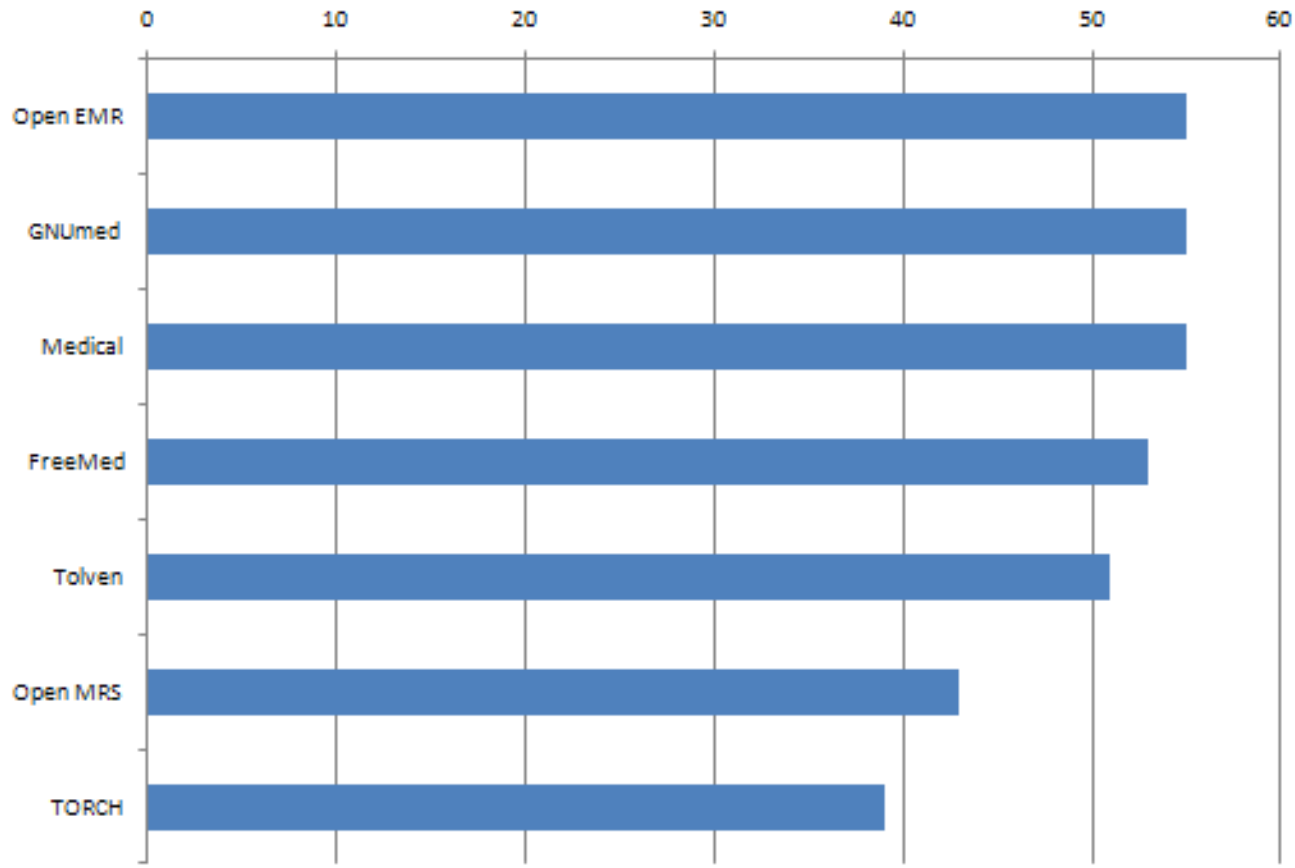


Abbildung 3.7.. Diagramm der Punkteverteilung

Tabelle 3.1.. Punkteverteilung der Systeme

System	Punkte (total)
OpenEMR	55
GNUmed	55
Medical	55
Freemed	53
Tolven	51
Open MRS	43
TORCH	39

In den Top 3 ist ausser den beiden Python-basierten Systemen Medical und GNUmed noch ein KIS (OpenEMR) enthalten, welches auf der XAMPP-Architektur aufgebaut ist. Bei der Entscheidung zeigt sich aber hier im Hinblick auf die Aufgabenstellung und die spätere Umsetzung eine Hürde: Da mitunter grafische Elemente oder sogar Animationen vonnöten sind und zusätzlich Bedienelemente zum Einsatz kommen, welche der Navigation dienen, ist man mit den nativ unterstützen Möglichkeiten eines Browsers schnell an einem Punkt angelangt, an dem Erweiterungen nötig sind. Flash-Komponenten oder JavaScript-Zusätze könnten hier Abhilfe schaffen. Die Unterstützung für jede Art von

Endgerät für diese Technologien ist aber mangelhaft. Um auf einen Einsatz diese Erweiterungstechnologien zu verzichten, macht eine ganzheitliche Lösung Sinn. Zwar werden bei Python viele Aufgaben im grafischen Bereich auch von zusätzlichen Bibliotheken übernommen, welche aber einfacher einzubinden sind. Zudem lässt sich dann alles direkt in Python programmieren. Die Entscheidung fällt schlussendlich auch mit der Präferenz sowohl des Auftraggebers als auch des Autors.

3.4.1. Entscheid System für Weiterentwicklung

Die letzten verbliebenen Kandidaten waren also Medical und GNUmed. Bei typischen Arbeitsschritten wie neue Patienten anlegen, neue Episode hinzufügen oder Dokumente hinzufügen zeigt sich GNUmed von der besseren Seite als Medical. Da Medical auf einer ERP(Enterprise Resource Planing)-Software aufsetzt, werden viele Gegebenheiten dieses ERP-Systems übernommen.

Normalerweise dient eine solche Software dazu, Geschäftsprozesse wie z.B. Personalwirtschaft und Produktion abzubilden. So werden z.B. behandelnde Ärzte als Geschäftspartner angesehen, was nicht allzu selbstverständlich ist. Screenshots auf der Medical-Website zeigen zwar eine Unterstützung für Dokumente, aber auch nach längerem testen war nicht herauszufinden, wie diese Funktion genutzt werden kann. Das eher nicht geeignete Grundsystem ist Medical deutlich anzumerken. Das von Grund auf als EMR-Software ausgelegte GNUmed zeigt sich hier als besserer Kandidat.

Bei den unterstützen Formaten zeigt sich bei den geforderten (Bilder, Videos, PDF) keine Einschränkung, auch grössere Dateien werden in das System geladen. Zudem können papierbasierte Dokumente durch eine Installation eines TWAIN[8]-Moduls leicht eingescannt werden; ein im Hinblick auf den Import bereits vorhandener Daten wichtiger Punkt. Auch bestehende xDT-Dateien können eingelesen werden. GNUmed ist in allen Belangen sehr gut dokumentiert und bietet einen Java-ähnliche API-Doc. Die Formen des Supports reichen von professionellem, kommerziellem Support einer externen Firma bis hin zu einem von den Entwicklern betriebenen Forum. Neben generellen FAQs finden sich auch Themen speziell zur Weiterentwicklung von GNUmed, die Einstiegshürde ist hier nicht so hoch wie bei Medical, wo Vorkenntnisse des Hauptsystems OpenERP vonnöten sind. Am Ende hat auch die Handhabung des Programms eine Rolle gespielt, wobei wie erwähnt die Vorteile bei GNUmed liegen.

[8] Technology Without An Interesting Name, Standard zum Austausch von Daten zwischen Bildeingabegeräten

4. Beschreibung des gewählten Systems

Das Projekt GNUmed soll zur qualitativen Verbesserung der Versorgung des Patienten beitragen. So wird die GNUmed-Software von einer Gruppe, welche sich vor allem aus Ärzten und Programmieren zusammensetzt, entwickelt und herausgegeben. Der Name GNUmed zeigt zum Einen die Anlehnung an das GNU-Projekt, der Zusatz „med“ liefert den Bezug zur medizinischen Branche. Die Software wird vor allem für die Verwaltung von Krankenakten genutzt; durch die laufende Entwicklung aber flossen immer mehr Teile einer medizinischen Institution ein, so dass mittlerweile die gesamte Behandlung eines Patienten mit der Software administriert werden kann. Die Entwickler legen dabei grossen Wert auf die Trennung zwischen primär patientenbezogenen Prozessen und solchen, welche sich im Hintergrund abspielen, wie etwa Administratives oder die Übernahme der Kosten durch Versicherungen. Eine vollständige Übersicht des Funktionsumfanges ist auf der GNUmed-Website[3] gelistet.

4.1. Eingesetzte Technologien

Die GNUmed-Module sind zum grössten Teil in Python umgesetzt; die Daten liefert eine PostgreSQL-Datenbank. Die Applikation wird über eine grafische Benutzeroberfläche gesteuert. Diese Oberfläche basiert auf WxPython, einem GUI-Toolkit, analog zu Swing(Java). Alle hier genannten Technologien sind quellcodeoffen und plattformunabhängig.

4.2. Client/Server

Um mit GNUmed arbeiten zu können, benötigt man sowohl den GNUmed-Client als auch den dazugehörigen Server. Bei dessen Installation wird grundsätzlich eine PostgreSQL-Datenbank installiert. Um die Verarbeitung der Daten und deren Anzeige in einer grafischen Benutzeroberfläche kümmert sich der GNUmed-Client, welcher also einen Rich Client darstellt. Zusätzlich weist die Erweiterbarkeit mit Modulen/Plugins auf diesen Umstand hin. Die Entwicklung eines Moduls wird in Abschnitt 4.7 erläutert.

4.3. Vorteile

Dass sich die Applikation für eine Weiterentwicklung eignet, hat sich in der Evaluation gezeigt. Weitere Vorzüge der Applikation werden aber erst in deren Handhabung sichtbar. So kann man zügig neue Patienten in das System aufzunehmen und - dem gewünschten Einsatzbereich entsprechend - Dokumente in die elektronische Akte eines Patienten hinzuzufügen. Die Grösse und Art der Dokumente spielt hierbei keine Rolle, wobei grosse Dokumente natürlich eine längere Ladezeit bedeuten, sowohl beim Upload auf den Server als auch beim Betrachten. Nicht digitalisierte Daten lassen über eine Scanner-Schnittstelle an eine Akte „anheften“. Aber auch eine Import-Möglichkeit für das bei niedergelassenen Ärzten weit verbreitete xDT-Format ist vorhanden. So ist die Möglichkeit

gegeben, vorhandene Daten zu importieren oder auf diese auf leichte Weise einzusehen.

Mit dem vorhanden Funktionsumfang kann also eine vollständige Behandlung eines Patienten abgebildet werden, von dessen Aufnahme in der Institution bis hin zu aktuellen Behandlungsmassnahmen. Informationen bezüglich Medikation und Allergien sind hiervon nicht ausgeschlossen. Viele der vorhandenen Funktionlitäten sind aber nicht für alle der behandelnden Personen notwendig. So ist es möglich, für verschiedene Benutzegruppen verschiedene Module anzuzeigen. In GNUmed wird dieses Problem mit Profilen gelöst. In jedem Profil kann genau ausgewählt werden, welche der vorhandenen Module angezeigt wird und welche nicht. Da diese Profile selbst angepasst werden können, schützen diese nicht vor unerlaubter Autorisierung; erst ein dediziertes Benutzerkonto kann hier eine klare Trennung schaffen.

4.4. Login-Verfahren

Nach dem Starten der Applikation erscheint zuerst eine Anmeldefenster, bei welchem der Server angegeben werden muss, mit welchem man sich verbinden möchte. Standardmässig stehen hier zwei Server zur Auswahl, zum Einen der öffentlich zugänglich Server des Entwicklerteams und der andere lokal installierte. Diesen kann man auf jeden Fall auswählen, auch wenn er noch nicht installiert wurde. Im Rahmen dieser Arbeit wird von einer lokalen Installation des Servers ausgegangen. Den zu Testzwecken vorgegebenen Benutzeraccount kann mit dem Benutzernamen „any-doc“ und dem genau gleich lautenden Passwort nutzen.

Abbildung 4.1.. Das GNUmed-Anmeldefenster mit Serverauswahl

4.5. Oberfläche

GNUmed wird über eine grafische Benutzeroberfläche gesteuert. Das GUI-Toolkit wxPython liefert hierfür die Basis. Die Applikation wird durch Panels unterteilt, um dann in jedem der Panels verschiedene Informationen anzuzeigen. Im Top-Panel von GNUmed wird neben des vollen Namens der aktuell gewählten Person auch deren Foto und Alter angezeigt. Zudem wird in roter Schrift auf eventuell vorhandene Allergien hingewiesen.

Im unteren Teil werden die einzelnen Module als „Tabs“ angezeigt, wobei je nach gewähltem Tab andere Informationen geladen werden. Diese Reihe von Tabs deckt sich zum grossen Teil mit der vorhandenen Menüleiste im Top-Panel. Die Oberfläche ist im Grossen und Ganzen also ziemlich einfach gehalten. Auf den Einsatz von Symbolen und Grafiken wird grösstenteils verzichtet. Wer mit den Symbolen auch deren Unterstützung

zur Bedienung vermisst, dem wird mit konsequent eingesetzten Tooltips geholfen. Bleibt der Mauszeiger also länger in einem Feld oder über einem Eingabefeld, erscheinen wichtige Hilfestellungen zur bspw. zur Art der Daten, die eingegeben werden müssen.

4.6. Architektur

Wie erwähnt ist der grösste Teil des GNUmed-Clients in Python umgesetzt. Das bei der Installation angegebene Verzeichnis beinhaltet neben Konfigurations- und Sprachdateien einen so genannten „Launcher“ welcher für den Start des Clients nötig ist. Die eigentlichen Module - und somit für eine Weiterentwicklung interessanten Dateien - befinden sich in der Bibliothek, welche von der installierten Python-Distribution genutzt wird. Um Änderungen an der Applikation vorzunehmen, muss also hier angesetzt werden. Bei einer Installation des vollständigen Clients werden folgende Verzeichnisse in die Python-Bilbiothek kopiert:

- business
- exporters
- pycommon
- wxGladeWidgets
- wxpython mit Unterverzeichnis /gui

Eine eingehende Analyse aller Verzeichnisse und deren Inhalt würde den Rahmen dieser Übersicht sprengen. Zudem werden viele Dateien und Module als „Blackboxes“ eingesetzt, das heisst die Module funktionieren und werden eingesetzt, ohne genau zu wissen wie sie im Inneren funktionieren. Als Vergleich sei ein Motor in einem Auto genannt. Der Fahrer weiss, wie er das Auto bedienen muss, um damit fahren zu können. Die genaue Funktionsweise des Motors ist ihm dabei aber nicht bekannt, obwohl er diesen nutzt. Im nächsten Abschnitt wird der Fokus auf die beiden letzten Verzeichnisse der obigen Liste gelegt und Schritt für Schritt erklärt, wie ein Plugin für GNUmed erstellt werden kann.

4.7. Plugins

Die vorrangig erwähnten Tabs werden von Entwicklern (und somit auch in der Dokumentation) Plugins genannt. Ein solches Plugin deckt einen bestimmten Aufgabenbereich ab oder erlaubt die Anzeige bestimmter Daten. Um ein neues Plugin zu entwickeln, bedient man sich am Besten bestehender Plugins und nimmt die entsprechenden Anpassungen vor.

4.7.1. Anzupassende Dateien

Ausgehend von einer bestehenden Datei sollen die Änderungen vorgenommen werden. Welche der vorhandenen Dateien dabei ausgewählt wird, spielt eine untergeordnete Rolle. Es kann aber hilfreich sein, nicht allzu komplexe und grosse Dateien für die Anpassungen zu kopieren. Um mit einem konkreten Beispiel weiterfahren zu können, soll ein

der Aufgabenstellung angepasster Name für die Bezeichnung des Plugins verwendet werden - Timeline. Natürlich kann hierfür auch ein anderer Name verwendet werden. Die folgenden Abschnitte sollen eine Anleitung geben, wie man generell Plugins für GNUmed erstellen kann. Der Bezeichnug Timeline soll für diesen Teil betrachtet also nicht allzu viel Beachtung geschenkt werden. Erst im Kontext mit dem späteren Kapitel, welches mehr Bezug auf die Umsetzung der Aufgabenstellung nimmt, kann an der Bezeichnung Timeline festgehalten werden.

Die im Folgenden gelisteten Dateien nutzen nicht immer gleichartige (d.h. sie besitzen den gleichen Namen) Plugins als Basis, da sich diese im Umfang deutlich unterscheiden. Auch die Entwickler legen sich hier nicht auf eine bestimmte Datei fest. Die vorgeschlagenen Dateien haben sich durch einen Vergleich vorhandener Pluginelemente hervorgetan, da sie sich bezüglich Umfang in Grenzen halten und nur die nötigsten Dinge abdecken. Ein wechselnder Name der anzupassenden Datei soll also die Datei zeigen, welche mit dem geringsten Aufwand angepasst wird. Zwecks Übersicht werden an dieser Stelle nur die Teile des Quellcodes gezeigt, welche wichtig für die Weiterentwicklung sind.

Komplette Listings mit Versionsdokumentation werden weggelassen. Auch Zeilennummern sind in den manchmal einzeiligen Anweisungen nicht vorhanden. Bei mehrzeiligen Listings stimmen zudem die Zeilennummern nicht genau mit dem Sourcecode der Dateien überein, deswegen wird auch darauf verzichtet. Für das Übersichtsdiagramm wird für den Pluginnamen jeweils „NAME“ eingesetzt.

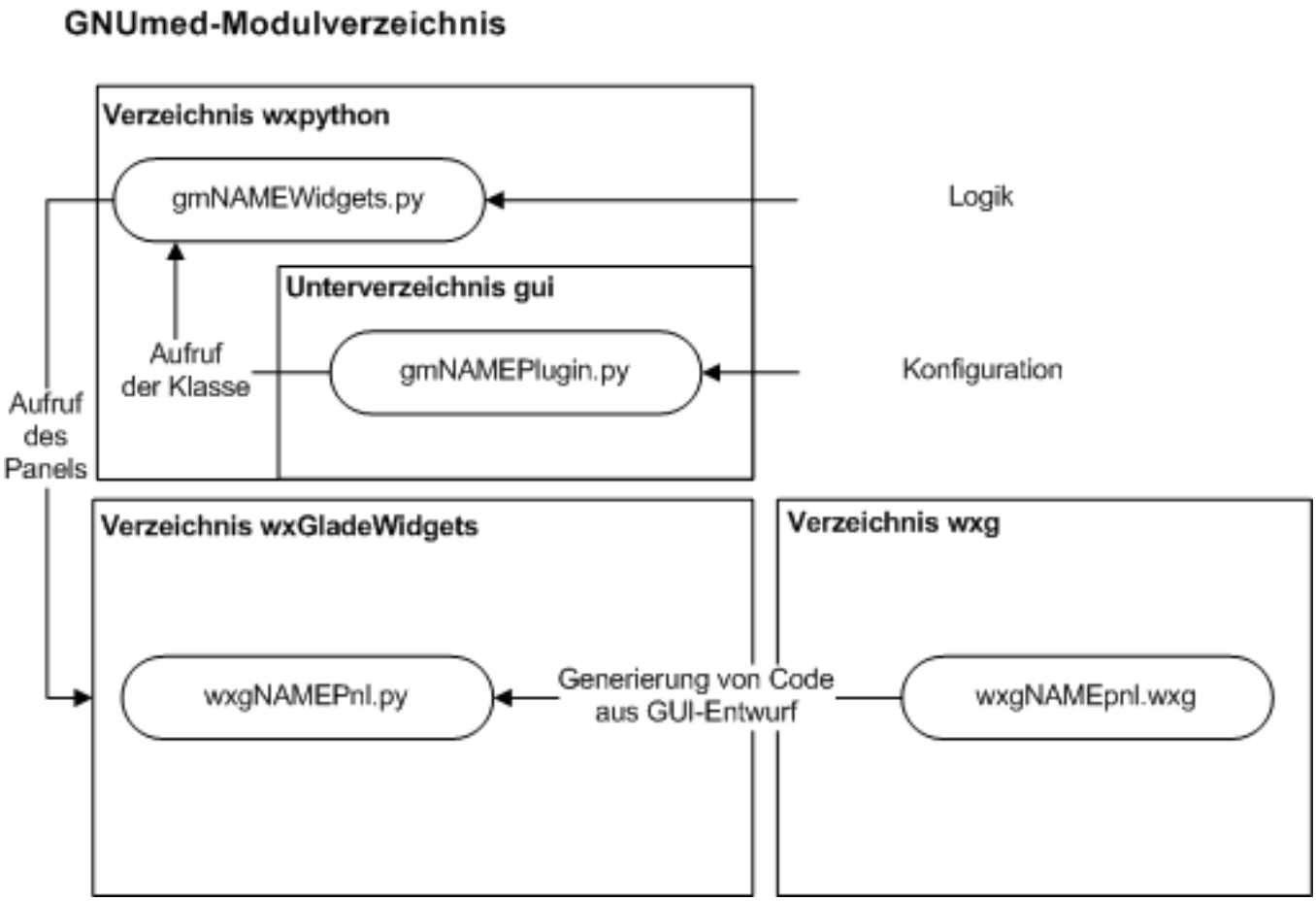


Abbildung 4.2.. Übersicht der abzuändernden Dateien und deren Zusammenhang

gmDataMiningPlugin.py

Diese Datei gilt als eine Art Konfigurationsdatei und findet sich im Ordner `wxpython/gui`. Grundlegende Dinge wie etwa der Name des Tabs und des Menüeintrags werden hier angepasst. Zudem wird festgelegt, welche der Oberflächen erscheint, wenn dieser Tab angewählt wird.

1. Import ändern: Die Datei `gmPlugin` sorgt dafür, dass alle vorhandenen Plugins konfiguriert und geladen werden. Die zweite zu importierende Datei beinhaltet die Logik. Auch wenn diese Datei noch nicht besteht, wird an dieser Stelle schon auf sie verwiesen. Nach dem Ändern des Namens muss die Zeile so aussehen:

   Listing 4.1. Ändern der Importanweisung

   ```
   from Gnumed.wxpython import gmPlugin, gmTimelineWidgets
   ```

2. Name der Klasse ändern: Damit die Datei richtig initialisiert wird, muss der Klassenname an den Dateinamen angepasst werden. Die Klasse leitet sich von der importierten Datei `gmPlugin` ab. Die komplette Zeile lautet wie folgt:

Listing 4.2. Anpassen des Klassennamens an den Dateinamen

```
class gmTimelinePlugin(gmPlugin.cNotebookPlugin):
```

3. Anzeigenamen des Tabs ändern: Mit dem Ändern dieses Attributes wird der Namen des Tabs im Programm geändert:

Listing 4.3. Anpassen des Tab-Namens

```
tab_name = _('Timeline')
```

4. Laden des Anzeigenamens: Damit der oben festgelegte Namen auch geladen wird, muss folgende Änderung der Funktion `n me()` erfolgen:

Listing 4.4. Laden des Anzeigenamens

```
def name(self):
 return gmTimelinePlugin.tab_name
```

5. Funktion `GetWidget()` anpassen: Diese Funktion legt fest, welches der vorhandenen Oberflächen effektiv geladen wird. Auch hier wird auf noch nicht bestehende Inhalte verwiesen.

Listing 4.5. Anpassen der GetWidget-Funktion

```
def GetWidget(self, parent):
 self._widget = gmTimelineWidgets.cTimelinePnl(parent, -1)
 return self._widget
```

6. Menüeintrag ändern: Die letzte Änderung in dieser Datei legt den Menüleisten-Eintrag fest. Der erste Parameter steht für den Menüpunkt in der Leiste, der zweite legt den Unterpunkt fest. Der Unterpunkt Timeline wird also in der Sparte Werkzeuge angezeigt werden. Die Benennung ergibt sich durch die deutsche Übersetzung um lautet im Quelltext `tools`.

Listing 4.6. Anpassen Menüleisten-Eintrags

```
def MenuInfo(self):
 return ('tools', _('&Timeline'))
```

Damit sind die Änderungen an dieser Datei abgeschlossen. Damit das neue Plugin angezeigt werden kann, sind aber noch weitere Änderungen vonnöten.

gmDeviceWidgets.py

Alle Dateien bei denen der Begriff `Widgets` im Dateinamen vorkommt, können verschiedenen Plugins Daten liefern, die Widgets selber holen sich dabei Daten von vielen Ressourcen. Es sei an dieser Stelle darauf hingewiesen, dass sich die Bezeichnung Widget

in diesem Kontext nicht mit der eigentlichen Definition eines Widgets[1] deckt.

Die ausgewählte Datei `gmDeviceWidgets.py` ist dafür vorbereitet, Daten von Geräten zu verarbeiten. Der sonstige Inhalt wird nicht weiter besprochen, da die Datei nur als Vorlage benutzt werden soll. Auch bei dieser Datei werden Schritt für Schritt Anpassungen vorgenommen. Vorerst sollen aber nur die Änderungen beleuchtet werden, die dazu beitragen, das Plugin anzeigen zu können. Erweiterte Änderungen bezüglich Logik und Erweiterungen im Bezug auf die Aufgabenstellung folgen in einem späteren Kapitel dieser Arbeit.

1. Import ändern: Grosse Änderungen müssen hier nicht mehr vorgenommen werden. Grundlegende Module werden bereits importiert. Zudem lassen sich diese zu einem späteren Zeitpunkt bei allfälligen Anpassungen noch verändern. Einzig die letzte Zeile erfährt eine Änderung und verweist auf das Panel, welches es zu einem Zeitpunkt noch zu erstellen gilt. Vorerst soll die Zeile aber so aussehen:

   Listing 4.7. Import des Panels

   ```
   from Gnumed.wxGladeWidgets import wxgTimelinePnl
   ```

2. Unnötigen Code entfernen: Da im neuen Plugin schliesslich kein Code dupliziert werden soll, werden ganze Codeblöcke aus der Datei entfernt. Zudem ist die Datei sonst einfach unübersichtlich. Alles, was in der "Rohversion"noch vorhanden sein soll, ist die Definition der Klasse und deren Initialisierung. Von der doch recht umfangreichen Datei bleibt nur relativ wenig übrig.

Nach Imports von Systembibliotheken in der ersten Zeile folgen im weiteren Abschnitt die bereits angesprochenen Imports von verschiedenen Grundmodulen von GNUmed. Anschliessend wird die Klasse `cTimelinePnl` erstellt, welche sich von der Datei `wxgTimelinePnl.py` ableitet. Die Klasse enthält vorerst nur eine Funktion `__init__`, welche wiederum das vorhandene Panel `wxgTimelinePnl` initialisiert.

Listing 4.8. Die Datei gmTimelineWidgets in Rohform

```
import sys, logging, datetime as pyDT, decimal, StringIO
import wx

if __name__ == '__main__':
 sys.path.insert(0, '../../')

from Gnumed.business import gmPerson, gmDevices, gmDocuments,
gmDemographicRecord
from Gnumed.pycommon import gmDispatcher, gmMatchProvider
from Gnumed.wxpython import gmRegetMixin, gmGuiHelpers,
gmPatSearchWidgets
from Gnumed.wxGladeWidgets import wxgTimelinePnl

class cTimelinePnl(wxgTimelinePnl.wxgTimelinePnl):

 def __init__(self, *args, **kwargs):
  wxgTimelinePnl.wxgTimelinePnl.__init__(self, *args, **kwargs)
```

[1] Kleine eigenständige Komponente eines grafischen Fenstersystems, welche auf Userinteraktion reagiert und simple Aufgaben erfüllt

Der letzte Schritt, der nun vorgenommen werden muss, ist die Erstellung der Oberfläche im GUI-Editor wxGlade[7]. Auch hier bietet sich an, vorhandene Dateien an die entsprechenden Bedürfnisse anzupassen. Wichtig ist hierbei, dass die grafischen Elemente - namentlich die Panels - den Name tragen welchen sie im Sourcecode erhalten haben. Bezogen auf die Namensgebung der Datei `gmTimelineWidgets.py` (siehe oben) muss die Datei also `wxgTimelinePnl.py` heissen und ein gleich lautendes Panel enthalten.

Um auf bereits vorhandene Oberflächen zurückgreifen zu können, bedarf es allerdings eines grösseren Zwischenschrittes. Hierfür müssen die neuesten Dateien aus dem öffentlich zugänglichen Repository geholt werden. Für diesen Vorgang findet sich auf der GNUmed-Website eine Anleitung für Windows[6]. Für andere Betriebssysteme bestehen natürlich auch Anleitungen.

Hat man die bestehenden Oberflächen geladen, wird nun eine von den Dateien geöffnet. Nebst der Anzeige der tatsächlichen Oberfläche wird ein Überblick über die Hierarchie der verwendeten Elemente gezeigt.

Die Datei `wxgW itingListPnl.wxg` liefert hierfür eine gute Vorlage. Gleich das erste Unterelement der Hierarchie ist ein Panel. Mit einem Doppelklick darauf werden die Panel-Eigenschaften geladen. Im Eigenschaftenfenster auf der linken Seite erscheint beim Punkt Class nun der Klassenname. Also muss der Name hier angepasst werden. Das Hinzufügen weiterer Elemente wird hier nicht weiter erläutert. Weitere Details hierzu finden sich in Kapitel 5.4 sowie in Form eines Tutorials[8] auf der wxGlade-Website.

Um aus der Oberfläche Code zu generieren, muss in der Baum-Ansicht das oberste Element (Application) gewählt werden. Wieder zurück im Eigenschaftenfenster muss der Ausgabepfad des Codes angepasst werden. Als Verzeichnis muss `wxGl deWidgets` ausgewählt werden, und die Datei muss der oben genannten Konvention entsprechen, in diesem Fall `wxgTimelinePnl`. Mit dem Betätigen des 'Generate Code'-Buttons wird die Datei erstellt. Alle Voraussetzungen für die Anzeige des Plugins sind nun erfüllt. Da GNUmed mit unterschiedlichen Profilen arbeiten kann, muss das neu erstellte Plugin noch diesem Profil hinzugefügt werden.

Hierfür wird der GNUmed-Client gestartet. Unter 'GNUmed' -> 'Stammdaten' -> 'Arbeitsplatz-Profile' lasen sich die bestehenden Profile anpassen. Beim Bearbeiten erscheinen alle verfügbaren Plugins - so auch das erstellte gmTimelinePlugin. Dieses wird angewählt, der Dialog bestätigt und die Applikation dann neu gestartet. Das Plugin wird nun unten rechts als letzter Tab angezeigt; der Vorgang ist somit abgeschlossen.

5. Weiterentwicklung des Systems

Die bestehende Applikation übernimmt bereits viele Aufgaben wie etwa Patienten- und Dokumentverwaltung; visuelle Elemente sind aber bei keinem der Plugins zu finden.

5.1. Zielsetzung

Mit der bestehenden Applikation lassen sich bereits Patienten in einer Datenbank erfassen. Jeder Episode können Dokumente hinzugefügt werden, diese sind bisher aber nur als Auflistungen zu sehen. Die meisten Plugins bedienen sich einfachen Listen oder - als Erweiterung davon - Baummenüs. Das Plugin Timeline soll nicht zur Verwaltung der Dokumente dienen, vielmehr soll es den in die Behandlung und Pflege involvierten Personen ermöglichen, sich die ganze Patientengeschichte abbilden zu lassen. Mit dem zu erstellenden Plugin sollen zudem die bestehenden Dokumente auf einer Art Zeitleiste angezeigt werden, welche sich über die ganze Lebensdauer eines Patienten erstreckt.

Aus der oben formulierten Zielsetzung wurden Use Cases definiert. Mit Use Cases sollen genaue Abläufe festgelegt werden, welche das Verhalten des Systems beschreiben und zu einem Ziel führen, welches für den Benutzer einen Nutzen hat. Die folgenden Use Cases beschreiben, wie man vorgehen kann, um die oben genannte Zielsetzung zu erreichen. Sie gehen dabei nur auf die Funktion der Zeitleiste ein und behandeln nicht die Verwaltung der Patienten oder deren Dokumente. Diese Aufgaben können mit dem bestehenden System bereits erfüllt werden.

1. Zeitleiste eines Patienten anzeigen
2. In die Zeitleiste eines Patienten zoomen
3. Details an einer bestimmten Stelle der Zeitleiste anzeigen, wie etwa zugehörige Episode, Datum
4. Vorschau des Dokuments direkt in der Zeitleiste anzeigen

Im nächsten Abschnitt sollen die in den Use Cases definierten Abläufe in einem Interaktionskonzept beschrieben werden.

5.2. Interaktionskonzept

Für diese visuell unterstützte Zeitabfolge kann eine ähnliche Form der Darstellung verwendet werden, wie sie die Applikation Tolven[1] nutzt. Auf einer Zeitachse, welche alle Jahre vom Behandlungsbeginn bis zum aktuellen Zeitpunkt abdeckt, werden aufgetretene Episoden mit Strichen angezeigt. In einer detaillierten Ansicht werden aus den Strichen bereits Punkte. Wird nun der Mauszeiger über den Punkt bewegt, wird aus dem normalen Mauszeiger eine Hand, wie dies bei Links in einem Browser üblich ist. Die

[1] siehe Evaluation Abschnitt 3.2.6

Punkte weisen also auf mehr Informationen hin. Eine ähnliche Zeitleiste soll nun auch in GNUmed erscheinen. Da vor allem Wert auf multimediale Inhalte gelegt wird, soll bereits vor dem Anwählen eines Punktes bzw. eines Dokumentes ersichtlich sein, welche Art von Datei sich dahinter verbirgt. Bekanntlicherweise kann GNUmed mit jedem Dateiformat umgehen, wirklich sinnvoll scheint aber nur die Unterscheidung von Bild- und Videoformaten sowie Dokumenten in textueller Form. Selbstverständlich kann jede dieser Gruppen mehrere Dateitypen umfassen. Die Einteilung der vorhandenen Dokumente erfolgt anhand kleiner Symbole. Diese Symbole ersetzen die Punkte aus dem Tolven-Ansatz und geben bereits Aufschluss über die Art des zugehörigen Dokuments einer Episode.

Als weiterer Unterschied ist die Zeitachse zu nennen, welche - so ist es bei Tolven geregelt - bei der ersten vorgenommenen Behandlung beginnt. In dieser Umsetzung soll der Start der Achse sogleich dem Geburtsdatum entsprechen. Dies erklärt sich im Hinblick auf die Patienten des Spaltzentrums, welche bereits mit dieser Behinderung zur Welt kommen und nicht plötzlich erkranken.

Nun ist es aber nötig, die Übersicht über den ganzen Krankheitsverlauf von detaillierteren Informationen zu trennen. Die Anfangs erwähnte Zeitachse soll dabei die ganzheitliche Übersicht liefern und durch zusätzliche Panels mit kleineren Zeiteinheiten wie etwa Monate ergänzt werden. Ein Schieberegler legt dabei fest, welcher Abschnitt der Übersichts-Zeitliste angezeigt wird. Dieser Regler erhält durch die Möglichkeit, ihn in der horizontalen zu skalieren, eine weitere Funktion. Mit der Wahl des Bereichs wird dieser automatisch in den Detailpanels abgebildet.

Da die Dokumente mit Gegebenheiten wie Krankenhausaufenthalten, vorgenommenen Operationen und Medikation in Zusammenhang gebracht werden sollen, erweist sich eine Aufteilung in genau diese drei Sparten als sinnvoll. Zu den vorhandenen Krankenhausaufenthalten und Operationen werden dann die erwähnten Symbole zugewiesen. Das Medikation-Panel zeigt nur die vorhandenen Daten zu den Medikationen an und soll nicht zusätzlich mit Dokumenten versehen werden. Die untenstehende Abbildung, welche als grafischer Entwurf für dieses Konzept dient, veranschaulicht die Überlegungen noch einmal.

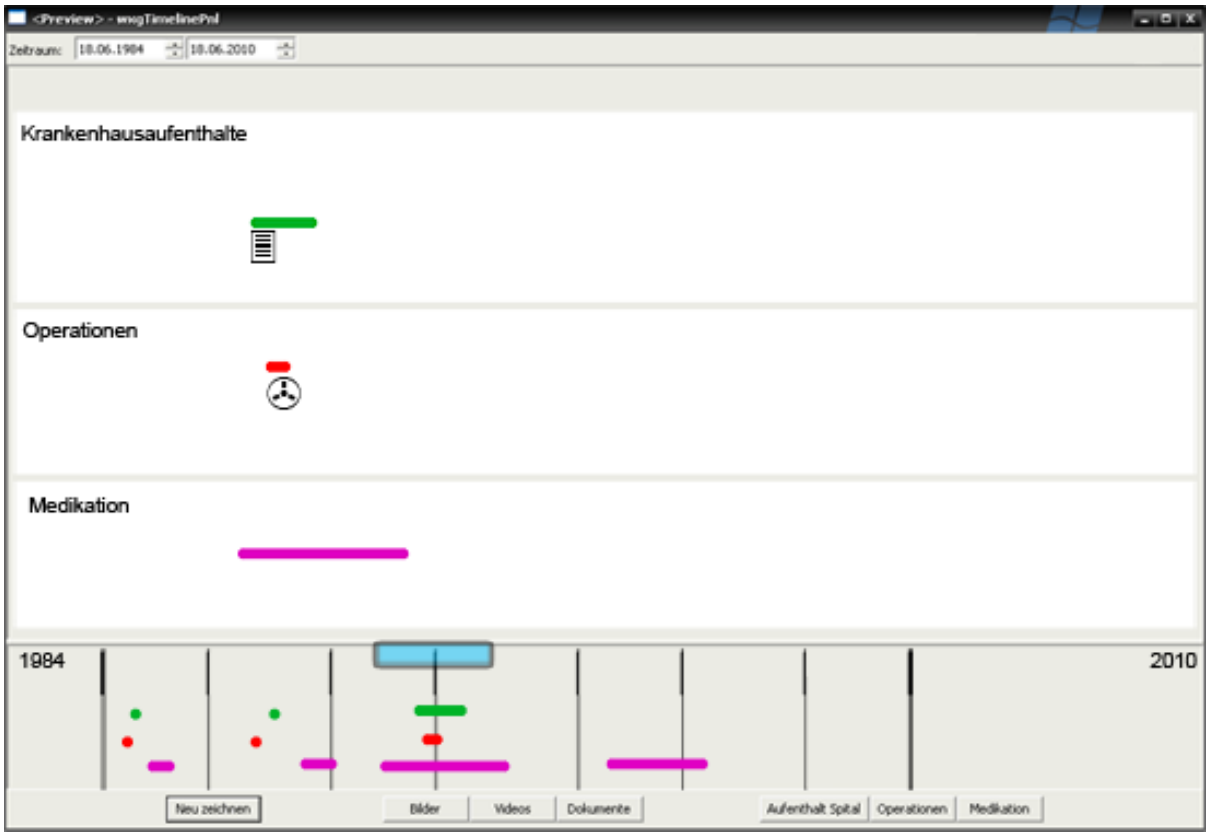


Abbildung 5.1.. Der Schieberegler über der Übersichtleiste legt fest, welche Bereiche in den drei Detailpanels angezeigt werden

Denkbar ist zudem, dass bei verschiedenen Maus-Events, wie etwa das Bewegen der Maus über das Symbol oder das Klicken derselben noch weitere Informationen, wie etwa Kommentar des Dokuments und dessen Grösse angezeigt werden.

5.3. Ermittlung der Datenstrukturen

Um herauszufinden, woher sich die benötigten Daten bekommen lassen, ist ein Blick in die GNUmed-API hilfreich. Nach näherer Betrachtung des Pakets `business` wird klar, das die darin enthaltenen Dateien Objekte zur Verfügung stellen. Der Entwickler muss sich also nicht um die Herkunft der Daten kümmern, sondern kann mit vorhandenen Objekten arbeiten. Die Datenbankabfrage wird an anderer Stelle erledigt und braucht ihn nicht zu interessieren. Er kann die vorhandenen Objekte also als Blackbox[2] nutzen.

Es gilt nun, für vorerst mindestens zwei Aufgaben eine Lösung zu finden: Erstens muss auf die Stammdaten eines Patienten zugegriffen werden, um Dinge wie Geburtsdatum, aktuelles Alter und Daten zur Medikation ermitteln zu können. Zweitens sollen die dem aktiven Patienten zugeordneten Dokumente erfassbar gemacht werden. In der API-Dokumentation hat sich das `gmPerson`-Objekt als dasjenige hervorgetan, welches Daten des Patienten liefert. Da die umfangreiche API aber natürlich keine fertigen Rezepte liefert, sind Nach-

[2] siehe Abschnitt 4.6

forschungen an anderen Stellen vonnöten.

Um herauszufinden, wie das aktuelle Alter des Patienten ermittelt wird, sei auf einen Bereich der Applikation verwiesen, beim welchem diese Information schon angezeigt wird. Im oberen Teil des Programms befindet sich das Top-Panel, welches nebst dem Foto des Patienten auch dessen Geschlecht sowie Geburtsdatum anzeigt. Eine Untersuchung der Datei `gmTopP nel` sollte also den erhofften Codeabschnitt liefern, und tatsächlich wird man auf Zeile 305 fündig. Allerdings wird nicht das Geburtsdatum, sondern das aktuelle Alter des Patienten in Form von x Jahre, y Monate zurückgegeben.

Listing 5.1. Zugriff auf das medizinische Alter

```
self.curr_pat['medical_age']
```

Das Objekt wird im Konstruktor initialisiert:

Listing 5.2. Zugriff auf die Patientendaten Zeile 52

```
self.curr_pat = gmPerson.gmCurrentPatient()
```

Diese Lösung für das erste Problem wird vorerst bei Seite gelassen. Beim späteren, genaueren Erläutern der Erweiterung wird dieser Ansatz genauer beleuchtet, dabei werden auch die Daten zur Medikation angesprochen.

Die zweite zu untersuchende Datei ist diejenige, welche Daten für dokumentenbezogene Plugins liefert. Hier kommen die im Kapitel 4.7.1, erwähnten Widget-Dateien zum Zug. Diese stellen Daten und Funktionen für verschiedene Aufgabenbereiche der Applikation zur Verfügung. Unter den genannten Dateien (zu finden im Verzeichnis `wxpython`) ist auch eine mit dem Namen `gmDocumentWidgets` vorhanden. Die Datei ist mit beinahe 2000 Zeilen Code ziemlich umfangreich, dennoch ist ja bekannt, nach welchem Objekt gesucht werden muss. Der folgende Codeabschnitt liefert das gewünschte Resultat:

Listing 5.3. Die Dokumente eines Patienten werden ermittelt

```
# read documents from database
curr_pat = gmPerson.gmCurrentPatient()
docs_folder = curr_pat.get_document_folder()
docs = docs_folder.get_documents()
```

Mit diesen Informationen sind die Grundlagen vorhanden, um sich der Umsetzung des Interaktions-Entwurfs zu widmen. Die Erstellung eines neuen Plugins wurde bereits behandelt und ist in Kapitel 4.7 beschrieben. Nun geht es darum, die Oberfläche für das Plugin umzusetzen und die Funktionalität zu implementieren.

5.4. Umsetzung des Entwurfs

5.4.1. Umsetzung des GUI-Entwurfs

Hier wird auf den bereits verwendeten GUI-Editor wxGlade zurückgegriffen. Wiederum wird die Datei `wxgW itingListPnl` geladen und sogleich unter einem anderen Namen gespeichert. Dieser muss bekanntlicherweise `wxgTimelinePnl` lauten. Und auch das Panel, welches als erstes Unterelement der Hierarchie erscheint, muss diesen Name erhalten. Das Panel enthält natürlich noch weitere Unterelemente, welche mit einem Klick auf

die jeweiligen Pluszeichen angezeigt werden können. Die Ansicht ist so auszuklappen, dass sie unterstehender Abbildung entpricht:

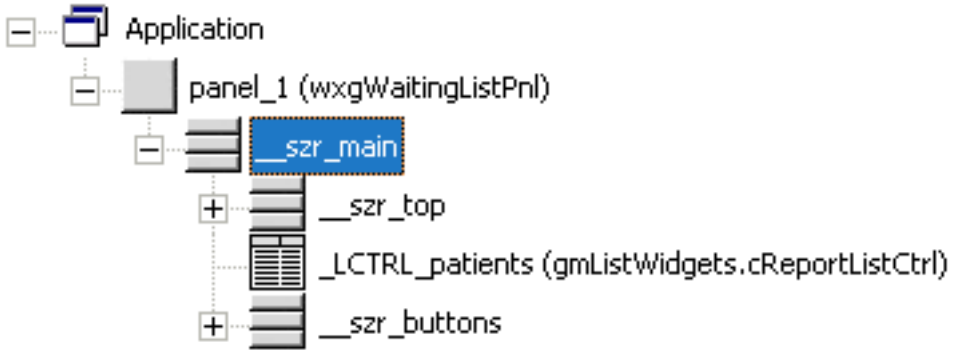


Abbildung 5.2.. Die Baumansicht der Datei wxgWaitingListPnl

Das nun in der Mitte stehende Element wird gelöscht, um Platz für ein Panel zu schaffen. Dazu wird im das Panel-Werkzeug ausgewählt und im nun leerstehenden Bereich (grau schraffiert) an beliebiger Stelle geklickt. Das nun platzierte Panel passt sich dem zur Verfügung stehenden Bereich an, soll an dieser Stelle aber nur als Platzhalter dienen.

Die bereits vorhandenen Buttons im unteren Bereich werden zu „Toggle-Buttons“ umgewandelt, welche nicht wie normale Buttons nach Loslassen der Maus in ihre urpsrüngliche Form zurückspringen, sondern ähnlich einem Schalter beim Klicken ihren Status bewahren. Die Funktionsweise kann mit Checkboxes verglichen werden. Mit diesen Buttons sollen später einzelne Elemente der Zeitleite ein- und ausgeblendet werden.

Das Slider-Element zur Navigation in der Übersichts-Zeitleiste wird wegen Unklarheiten bei der Bereichsaufteilung vorerst im oberen Bereich platziert. Die Arbeit im GUI-Editor ist nun beendet. Die Oberfläche sieht nun so aus:

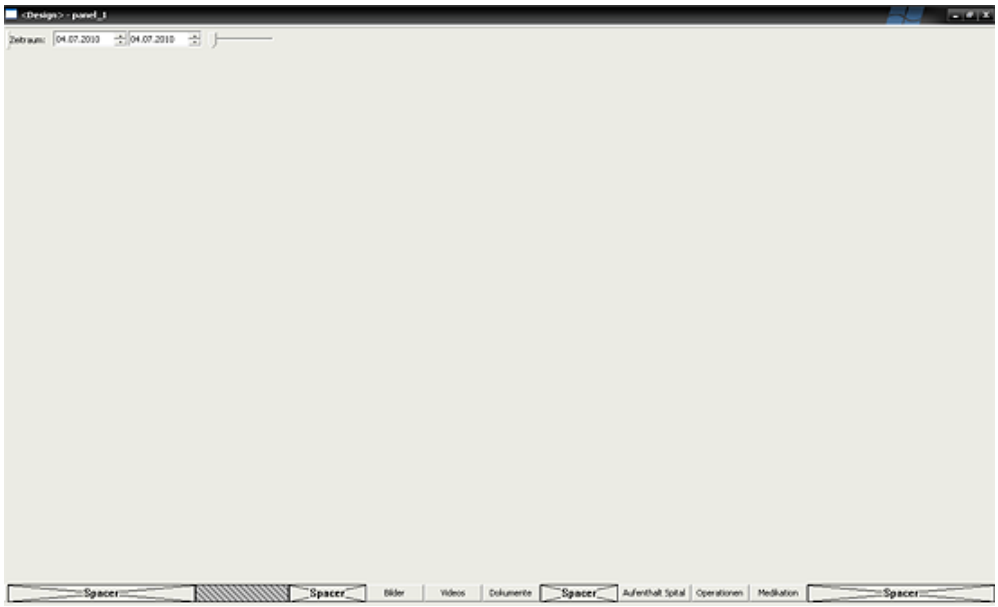


Abbildung 5.3.. Die fertige, leer anmutende Oberfläche (Ansicht in wxGlade)

Da das in der Mitte platzierte Panel noch keine Elemente beinhält, erscheint es noch ziemlich leer. Im nächsten Schritt soll es also mit Daten des Patienten gefüllt werden. Vorher muss aber aus der grafischen Vorlage in wxGlade noch Python-Code erstellt werden. Hierfür wählt man in der Baumansicht das oberste Element (Application) und betätigt nach der Auswahl der zu erstellenden Datei den Generate Code-Button. Somit wird die soeben erstelle Oberfläche auch wirklich angezeigt.

5.4.2. Erweiterung des Plugins

Die Erweiterung des Plugins wird sich nur auf die Datei `gmTimelineWidgets` beziehen. Es gilt nun, das vorhandene Wissen um die Daten des Patienten und die der Dokumente im vorhandenen Plugin abzubilden. Ausgehend von der in Abschnitt 4.7 erstellten Rohversion der Datei `gmTimelineWidgets` erfolgt die Erweiterung. Der erste Codeblock zeigt die benötigten Imports. Einzig das Paket `d tetime` wird zusätzlich importiert, um später mit Jahren rechnen zu können.

Listing 5.4. Imports der fertigen Datei

```
import sys, logging, datetime as pyDT, decimal, StringIO
import wx       #, wx.grid

if __name__ == '__main__':
 sys.path.insert(0, '../../')

#Imports fuer Rechnung mit Jahren
from datetime import datetime

#GNUmed-spezifische Imports
from Gnumed.business import gmPerson, gmDevices, gmDocuments,
gmDemographicRecord
from Gnumed.pycommon import gmDispatcher, gmMatchProvider
from Gnumed.wxpython import gmRegetMixin, gmGuiHelpers,
gmPatSearchWidgets
```

```
from Gnumed.wxGladeWidgets import wxgTimelinePnl
```

In einem zweiten Schritt wird die Klasse `cTimelinePnl` erstellt und in der `__init__`-Funktion eine Instanz erstellt, die Funktion gilt also als Konstruktor. Danach wird das Panel, auf welchem gezeichnet werden soll, erstellt. Die Bildschirmgrösse wird nicht der jeweiligen angepasst, sondern als Konstante festgelegt. Mit genau diesem Panel erfolgt mittels der Funktion `Bind()` eine Verknüpfung an einen „Event“, in diesem Fall der Event `wx.EVT_PAINT`. Wird also das Panel gezeichnet bzw. `wx.EVT_PAINT` ausgeführt, wird zugleich auch die Funktion aufgerufen. Die letzte Zeile des Konstruktors besorgt die Daten des Patienten.

Listing 5.5. Definition und Instanzierung Klasse cTimelinePnl

```
class cTimelinePnl(wxgTimelinePnl.wxgTimelinePnl):

 def __init__(self, *args, **kwargs):
  wxgTimelinePnl.wxgTimelinePnl.__init__(self, *args, **kwargs)

  #Setze Breite der Panels auf ungefaehre Bildschirmgroesse
  self.PANEL_WIDTH = 1200

  #Zeichne das Panel
  self.panel = wx.Panel(self, size=(self.PANEL_WIDTH, 800))
  self.Fit()

  #Die Funktion on_paint wird an den Event EVT_PAINT gekoppelt
  self.panel.Bind(wx.EVT_PAINT, self.on_paint)

  #Hole Patientendaten
  self.curr_pat = gmPerson.gmCurrentPatient()
```

Nachfolgend wird die `on_p int()`-Funktion aufgeführt, welche die sich um die Aufbereitung der Daten und deren Darstellung kümmert. Zuerst soll nach dem Funktionskopf das Zeichnen des Übersichts- sowie der Detailpanels übernommen werden:

Listing 5.6. Zeichnen der Panels auf das im zuvor erstellte Panel

```
def on_paint(self, event):
 dc = wx.PaintDC(self.panel)
 dc.SetPen(wx.Pen('white', 4))

 #Zeichnen der weissen Panels
 dc.DrawRectangle(10, 40, self.PANEL_WIDTH, 150)
 dc.DrawRectangle(10, 200, self.PANEL_WIDTH, 150)
 dc.DrawRectangle(10, 360, self.PANEL_WIDTH, 150)
 dc.DrawRectangle(10, 520, self.PANEL_WIDTH, 50)
```

Nun soll zum ersten Mal mit Daten des Patienten gerechnet werden. Die Funktion `get_form tted_dob()` wird mit dem Parameter format = '%Y' aufgerufen, um nur das Geburtsjahr auszugeben. Die Differenz von aktuellem Jahr und Geburtsjahr ergibt die Anzahl Jahre, welche abgedeckt werden müssen. Mit dieser Information lässt sich die Breite der Zeitachseneinheiten bestimmen, indem man die Breite des Panels durch die Anzahl Jahre dividiert. Nach der Beschriftung der Zeitachse folgt die Unterteilung in die einzelnen Jahre. Dieser Vorgang wird mit einer for-Schleife durchgeführt, welche bei Jahr eins beginnen soll.

Listing 5.7. Zeichnen der Detailpanels auf das im zuvor erstellte Panel

```
#Ermittle das aktuelle Datum
```

```
now = datetime.now()

#Ziehe Geburtsjahr vom aktuellen Jahr ab und ermittle Jahre
years = int(now.strftime('\%Y'))
-int(self.curr_pat.get_formatted_dob(format = '\%Y'))

# Ermittle Breite, die ein Jahr im Panel haben darf
offset = (self.PANEL_WIDTH / years)

#Beschriftung fuer Zeitachse
dc.SetPen(wx.Pen('grey', 1))
dc.SetFont(wx.Font(12, wx.MODERN, wx.NORMAL, wx.NORMAL))
dc.DrawText("DOB", 10, 580)
dc.DrawText(self.curr_pat['medical_age'], 1140, 580)

#Zeichne Linien der Zeitachse
for i in range(1, years):
 dc.DrawLine(10+(i*offset), 560, 10+(i*offset), 570)
```

Im nächsten Schritt werden die Daten bezüglich der Dokumente ins Spiel gebracht. Mit der Funktion `get_document_folder()` werden die Informationen in das Objekt docs geladen. Für alle enthaltenen Einträge wird über dieses Objekt wird mit einer for-Schleife iteriert. Analog zum Geburtsdatum wird das Erstellungdatum des Dokumentes vom aktuellen Datum abgezogen, um die Anzahl der Jahre und somit die Platzierung der Symbole herauszufinden. Da die Jahreszahlen als Strings ausgegeben werden, müssen sie zuerst in den Ganzzahl-Datentyp `int` konvertiert werden, um mit ihnen rechnen zu können. Welches Symbol dabei ausgewählt wird, hängt vom Dateityp der Dokumente ab. Entspricht der Typ nicht einem der definierten, wird ein generell gültiges Symbol gewählt. Unter das Symbol werden der Kommentar und das Erstellungsjahr der Dokumente angezeigt, um diese besser einordnen zu können.

Listing 5.8. Codeblock zur Ermittlung der Dokumentendaten und Platzierung der Symbole

```
{lst_block_dokumente}
#BLOCK DOKUMENTE
#Hole Daten zu Dokumenten
docs_folder = self.curr_pat.get_document_folder()
docs = docs_folder.get_documents()

for doc in docs:
 years_docs = int(now.strftime('\%Y'))
 -int(doc['clin_when'].strftime('\%Y'))

 #Pruefen welcher Dateityp und entsprechendes Bild laden
 if str(doc['l10n_type']) == 'Dokument':
  imageFile = 'C:\\Programme\\GNUmed-client\\bin\\bitmaps\\
␣␣page_white_text.png'
  offset_height = 100

 elif str(doc['l10n_type']) == 'sonstiger␣Arztbrief':
  imageFile = 'C:\\Programme\\GNUmed-client\\bin\\bitmaps\\
␣␣page_white_text.png'
  offset_height = 100

 elif str(doc['l10n_type']) == 'Bild':
  imageFile = 'C:\\Programme\\GNUmed-client\\bin\\bitmaps\\
␣␣camera.png'
```

```
    offset_height = 260

  elif str(doc['l10n_type']) == 'Video':
    imageFile = 'C:\\Programme\\GNUmed-client\\bin\\bitmaps\\
    webcam.png'
    offset_height = 420

  else:
    imageFile = 'C:\\Programme\\GNUmed-client\\bin\\bitmaps\\
    box.png'
    offset_height = 450

  image = wx.Image(imageFile, wx.BITMAP_TYPE_ANY).ConvertToBitmap()
  dc.DrawBitmap(image, self.PANEL_WIDTH-(years_docs*offset),
  offset_height, True)

  dc.DrawText(str(doc['comment']), self.PANEL_WIDTH-(years_docs*offset),
  offset_height+15)
  dc.DrawText(str(doc['clin_when'].strftime('\%Y')), self.PANEL_WIDTH-
  (years_docs*offset), offset_height+25)
```

Der abschliessende Abschnitt zeigt die Daten zu den vom Patienten eingenommenen Medikamenten und zeichnet diese in der Übersichtsleiste. Hierfür wird in der ersten Zeile mit get_emr() die ganze Elektronische Akte geladen, um anschliessend daraus die Daten zu den Medikamenten zu beziehen (`get_current_subst nce_int ke()`). Der zurückgegebene String muss nun gekürzt werden, um daraus die Jahreszahl ermitteln zu können.

Die Umrechnung in Jahre muss an dieser Stelle etwas umständlich erfolgen, da die Ausgabe auf Tagen basiert und nicht auf Jahren und Monaten wie beim medizinischen Alter. Mit dem Startjahr der Einnahme wird auch der Startpunkt für die Linie festgelegt, welche die Medikation visualisieren soll. Um auch die Medikation einordnen zu können, wird der Name der Episode hinzugefügt.

Listing 5.9. Codeblock Zeitraum Medikamenteneinnahme

```
#BLOCK MEDIKAMENTE
emr = self.curr_pat.get_emr()
meds = emr.get_current_substance_intake()

#Hole Daten wie lange Patient Medis nimmt (nur Jahre)
s = str(meds[0]['duration'])
s = s.partition(" ")

#length_med steht fuer die Laenge der Medikamenteneinnahme
length_med = int(s[0])/365

#Ermittle Startjahr der Einnahme
year_med = int(now.strftime('\%Y'))
-int(meds[0]['started'].strftime('\%Y'))

#Zeichnen der Medis
dc.SetPen(wx.Pen('blue', 2))

#Linie vom Beginn bis Ende
dc.DrawLine(self.PANEL_WIDTH-(year_med*offset), 530,
self.PANEL_WIDTH-(year_med*offset)+(length_med*offset), 530)

dc.DrawText(str(meds[0]['episode']), self.PANEL_WIDTH-(year_med*offset), 535)
```

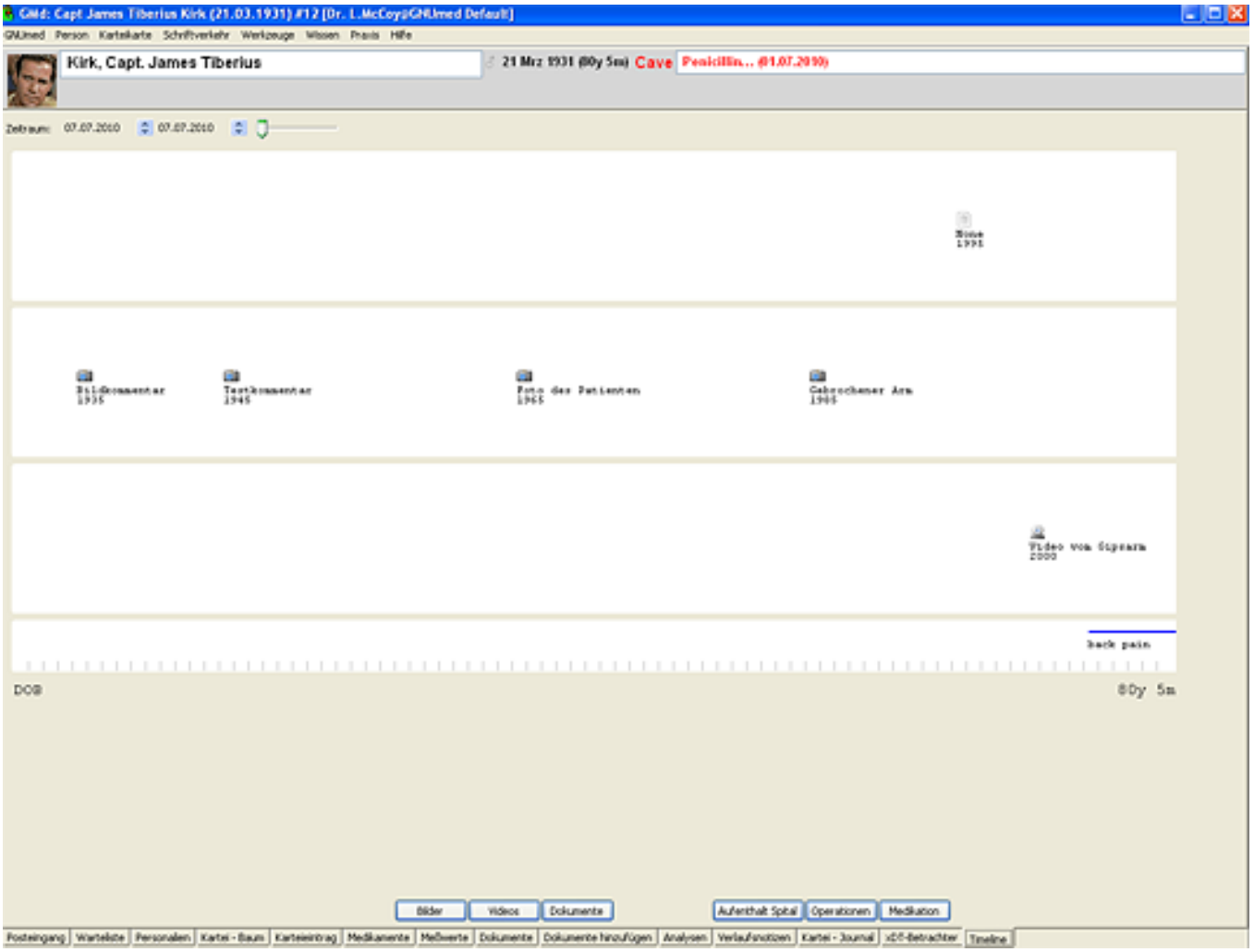


Abbildung 5.4.. Das weiterentwickelte Panel mit Anzeige der Dokumente und der Medikation

6. Schlussfolgerung

Das vorhergehende Kapitel hat sich der Weiterentwicklung eines Plugins für das GNUmed-System gewidmet. Diese hat unter einigen anderen das Rennen machen können und sich für diese Aufgabenstellung als am besten geeignet erwiesen. Viele der untersuchten Systeme erfüllten gewisse Kriterien - einige davon als Killerkriterien definiert - nicht. Die Applikationen mögen wohl viele der grundlegenden Aufgaben wie das Verwalten von Patienten und deren Krankheitsfälle beherrschen, doch nur wenige können sich im Bereich der patientenbezogenen Dokumentenverwaltung hervortun. Dies bildet schliesslich die Grundlage für eine multimedial gestützte, visualisierte Patientengeschichte. Das Thema Visualisierung scheint bei den Entwicklern der bestehenden Systeme keine oder nur eine sehr geringe Rolle gespielt zu haben. Beide Anforderungen nun zudem zu vereinen - diesen Ansatz scheint es noch nicht zu geben. Die Möglichkeit, die Anforderungen umzusetzen, besteht jedoch. Die vorliegende Arbeit zeigt, dass basierend auf freien, offenen Krankenhausverwaltungsystemen Erweiterungen umgesetzt werden können, die in Richtung Visualisierung weisen.

Eine Sondierung möglicher Kriterien für eine Evaluation erwies sich anfangs als schwierig und nur wenig fassbar. So haben sich Ermittlung der Kriterien und Evaluation der Systeme rückblickend als *ein* Prozess entpuppt. Der gewählte, experimentelle Ansatz hat sich als wenig effizient erwiesen, anderseits ist eine Trennung der beiden Prozesse in einer fast unbekannten Softwaresparte auch nicht leicht zu vollziehen. Zudem wurde der Thematik der elektronischen Patientenakten und deren Vorteile gegenüber papierbasierten Dokumenten zu viel Bedeutung zugeordnet, ein der Problemlösung nicht dienlicher Irrtum. Wahrscheinlich wäre auch beim Bewertungssystem ein Überdenken nötig; die Gewichtung mit Prozenten könnte hier eine ernstzunehmende Alternative darstellen. Auf jeden Fall scheint eine systematischere Vorgehensweise sinnvoller. Die Evaluation hat hierdurch sehr viel mehr Zeit gekostet, als dafür eingeplant gewesen wäre. Um den Evaluationsprozess klarer zu strukturieren, ist es sicher auch angebracht, vorangehend ein Szenario zu definieren, welches es durchzuspielen gilt.

Trotz allem hat sich die Entscheidung für GNUmed als die richtige erwiesen. Nebst einer sehr guten Dokumentation und Wegleitungen zur Entwicklung neuer Module bietet sich die Möglichkeit, direkt mit Entwicklern in Kontakt zu treten, welche selbst eifrig am Werk sind. Die vielen Updates allein im Zeitraum dieser Arbeit haben gezeigt, dass GNUmed eine langfristige Lösung darstellen soll und dabei viele Personen, auch Ärzte, am Entwicklungsprozess beteiligt sind. Nebst den guten Erweiterungsmöglichkeiten würde sich auch eine Verteilung der Erweiterung als einfach erweisen. So könnten die involvierten Personen den Referenzclient herunterladen, und durch einfaches Hinzufügen einiger Dateien wären sie bereits in der Lage, die Erweiterung zu nutzen. Die Möglichkeiten der Anpassung durch Profile und Erweiterung durch die Verwendung eines Open-Source-Systems scheinen jeder Anforderung gewachsen. Zudem ist die zu Grunde liegende Sprache Python weit verbreitet und einfach zu lernen. Ein Umstand, der dazu führen könnte, dass sich Ärzte und Entwickler einander annähern könnten - so wie dies beim

Hauptentwicklerteam der Fall ist.

Die vorliegende Arbeit zeigt die Schritte, welche für eine Weiterentwicklung nötig sind und wo Hilfestellungen gegeben werden. Auch wenn nur Teile des vorgesehenen Prototyps umgesetzt werden konnten, sind für weitere Projekte Grundlagen vorhanden, welche es ermöglichen, sich ganzheitlich mit der Entwicklung zu befassen. Auch im Hinblick auf andere spezifische Anwendungsfälle kann sich eine Verwendung von GNUmed daher als sinnvoll erweisen.

7. Literaturverzeichnis

wikiemr Wikipedia (2010): *List of open source healthcare software* URL: http://en.wikipedi .org/wiki/List_of_open_source_he lthc re_softw re [Stand: 20.04.2010]

[0] [1] Goomedic.com (2010): *Open source EMR* URL: http://www.goomedic.com/open-source-emr-list [Stand: 23.04.2010]

[2] Adler, Kenneth (2005): *How to Select an Electronic Health Record System* URL: http://www. fp.org/fpm/2005/0200/p55.html [Stand: 01.05.2010]

[3] Hilbert, Sebastian (2010): *What Can I Actually DO With GNUmed Today?* URL: http://wiki.gnumed.de/bin/view/Gnumed/Wh tC nIActu llyDOWithGNUmedTod y [Stand: 31.05.2010]

[4] Anokwa, Yaw/Allen, Christian/Tapan, Parikh (2008): *Deploying a Medical Record System in Rural Rwanda*, HCI4CID.

[5] Hilbert, Sebastian (2009): *Sources from CVS Windows* URL: http://wiki.gnumed.de/bin/view/Gnumed/SourcesFromCVSWindows [Stand: 04.06.2010]

[6] Busser, James (2009): *Installation Sources and Guides for GNUmed* URL: http://wiki.gnumed.de/bin/view/Gnumed/Inst llerGuideHome [Stand: 03.06.2010]

[7] Griggio, Alberto (2009): *wxGlade: a GUI builder for wxWidgets/wxPython* URL: http://wxgl de.sourceforge.net/ [Stand: 15.06.2010]

[8] Griggio, Alberto (2009): *wxGlade: a GUI builder for wxWidgets/wxPython* URL: http://wxgl de.sourceforge.net/tutori l.php [Stand: 15.06.2010]

[9] Thomson, Richard (unbekannt): *Open Clinical. Electronic medical records* URL: http://www.openclinic l.org/emr.html [Stand: 25.04.2010]

[10] Lynn, John (unbekannt): *Benefits of EMR or EHR Over Paper Charts | EMR and HIPAA* URL: http://www.emr ndhip .com/benefits-of-emr-or-ehr-over-p per-ch rts/ [Stand: 25.04.2010]

[11] Dugas, Martin/Schmidt, Karin (2002): *Medizinische Informatik und Bioinformatik: Ein Kompendium für Studium und Praxis* 1. Aufl. Springer, Berlin.

[12] Ernesti, Johannes (2008): *Python: das umfassende Handbuch. Einführung, Praxis, Referenz* 1. Aufl. Galileo Press, Bonn.

8. Tabellenverzeichnis

9. Abbildungsverzeichnis

10. Listings

11. Glossar

ICD International Statistical Classification of Diseases and Related Health Problems, das wichtigste, weltweit anerkannte Diagnoseklassifikationssystem der Medizin. Die aktuelle, international gültige Ausgabe ist ICD-10 und stammt aus dem Jahr 2006. Die ICD wurde von der Weltegsundeheitsorganisation WHO erstellt und dient der Diagnosen- und Prozedurenverschlüsselung medizinischer Leistungen. Anstoss für dieses System war ursprünglich eine Todesursachenstatistik. Für besondere Einsatzgebiete bestehen Spezialausgaben, wie z.B. ICD-O für die Onkologie.

HL7 HL7 bezeichnet zum einen eine seit 1987 bestehende NPO, welche Standards im Gesundheitswesen zum Austausch und Integration von Daten entwickelt. Zum andern wird genau dieser Standard ebenfalls mit HL7 bezeichnet. Er bietet standardisierte Formate und Protokolle zum Austausch von Nachrichten zwischen Computersystemen im Gesundheitswesen. Dadurch wird der Aufwand für Schnittstellen reduziert. Wird zum Beispiel ein Patient im Patientenverwaltungssystem erfasst, wird eine HL7-Nachricht von diesem Verwaltungssystem an bspw. das Laborsystem gesendet. So können die Daten des Patienten elektronisch in das Labor übermittelt und dort weiter verwendet werden.

DICOM Digital Imaging and Communications in Medicine, ein „weit verbreiteter, internationaler Standard für elektronische Bildkommunikation in der Medzine“. Der Standard befasst sich mit dem Austausch digitaler Bilder mit erweiterten Informationen. Im Detail werden Datenformate für medizinische Bilderserien und zahlreiche Annotationen wie etwa Name des Patienten, Art und Geräteparameter der Aufnahmen beschrieben. Damit soll der möglichste reibungslose Datenaustausch zwischen bildgebenden Modalitäten verschiedener Hersteller sichergestellt werden. Die Umsetzung in der Praxis lässt oftmals zu Wünschen übrig, oftmals ist dies auf wirtschaftliche Gründe aber auch auf die Komplexität des DICOM-Standards zurückzuführen.

PACS Picture Archiving and Communication System, dient zum Management grosser Mengen von Bilddaten und wird vor allem in der Radiologie eingesetzt. Die wichtigsten Aufgaben eines PACS sind die Bildarchivierung und die Bildkommunikation. Zudem sollen die Bilder schnell am jeweils benötigten Ort zur Verfügung gestellt werden. Der Speicherbedarf der Bilder stellt dabei für die Speicher- und Netzwerkkapazität die grösste Herausforderung dar.

XAMPP Bei Web-Anwendungen oder dynamischen Webseiten werden oft gleiche Architekturen eingesetzt. Das XAMPP-Paket vereint diese am häufigsten verwendeten Teilsysteme. Die Hauptbestandteile sind ein Webserver (Apache), eine Datenbank (MySQl) sowie die Skriptsprachen PHP und Perl. Das X steht dabei für die verschiedenen Betriebssysteme, für die die Distribution eingesetzt werden kann. Im Vordergrund steht vor allem eine einfache Installation der benötigten Umgebung, damit Entwickler schnell eine Testumgebung nutzen können. Aufgrund der dabei entstehenden Einschränkungen im Sicherheitsbereich wird daher von einem Einsatz als Produktivsystem abgeraten.

Erklärung

Hiermit erkläre ich, die vorliegende Bachelor-Thesis selbständig, ohne Mithilfe Dritter und unter Benutzung nur der angegebenen Quellen verfasst zu haben.

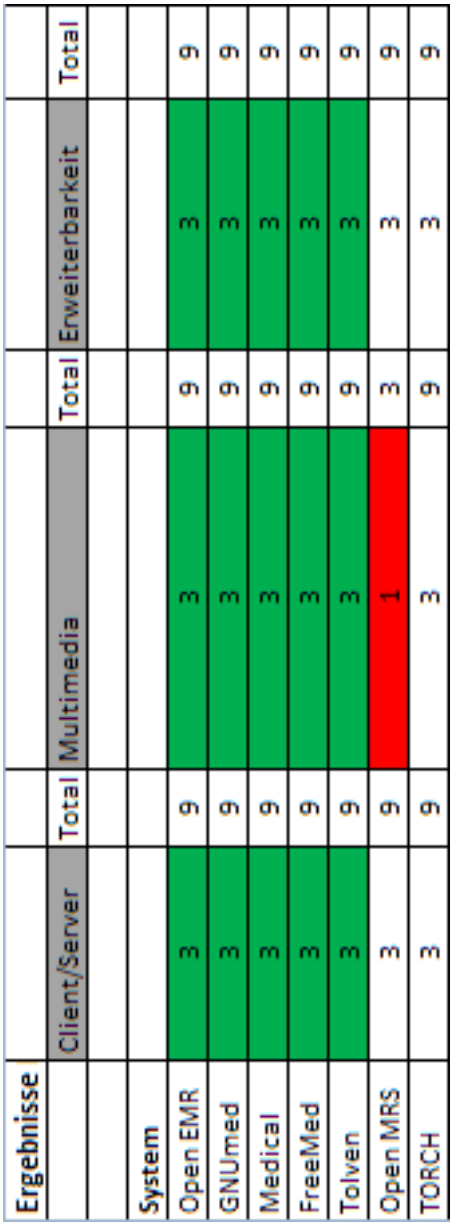
Datum

Unterschrift

A. Anhang

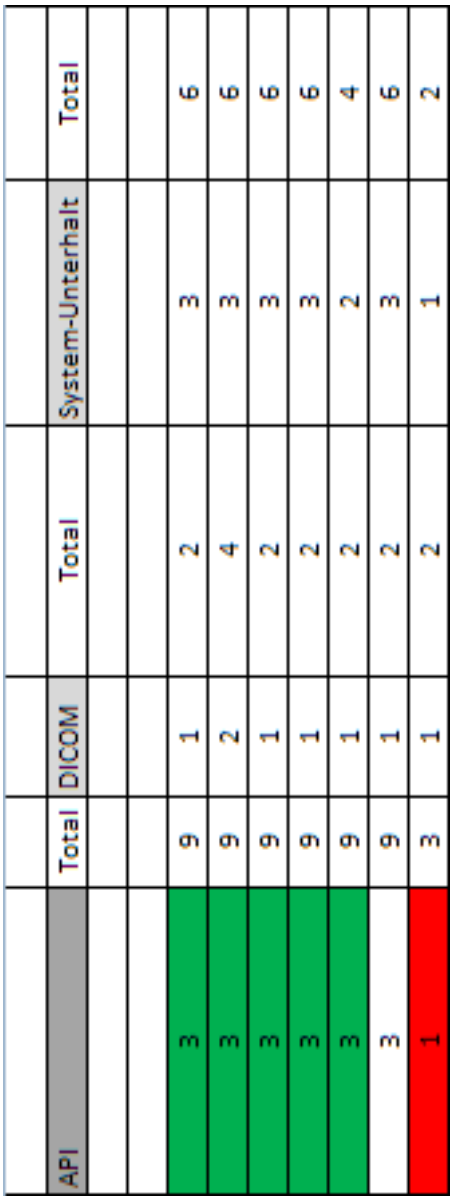
A.1. Tabellen zur Evaluation

A.1.1. Ergebnisse mit Punktetotal



Ergebnisse		Client/Server	Total	Multimedia	Total	Erweiterbarkeit	Total
	System						
	Open EMR	3	9	3	9	3	9
	GNUmed	3	9	3	9	3	9
	Medical	3	9	3	9	3	9
	FreeMed	3	9	3	9	3	9
	Tolven	3	9	3	9	3	9
	Open MRS	3	9	1	3	3	9
	TORCH	3	9	3	9	3	9

Abbildung 1.1.. Tabelle Ergebnisse mit Punktetotal Teil 1 von 4



	API	Total	DICOM	Total	System-Unterhalt	Total
	3	9	1	2	3	6
	3	9	2	4	3	6
	3	9	1	2	3	6
	3	9	1	2	3	6
	3	9	1	2	2	4
	3	9	1	2	3	6
	1	3	1	2	1	2

Abbildung 1.2.. Tabelle Ergebnisse mit Punktetotal Teil 2 von 4

	Support	Total	Lizenz	Total	ICD	Total
	3	6	GPL		3	3
	3	6	GPL		1	1
	3	6	GPL		3	3
	3	6	GPL		1	1
	2	4	LGPL		1	1
	1	2	Public License (Mozilla)		1	1
	1	2	GPL		1	1
Punkte						
ja = 2 Punkte						
nein = 0 Punkte						
Sonst: Gewichtung entspricht Punkten (siehe Evaluation)						

Abbildung 1.3.. Tabelle Ergebnisse mit Punktetotal Teil 3 von 4

Web-Zugriff Patienten	Total	DST	Total	GESAMT	
1	1	1	1	55	Open EMR
1	1	1	1	55	GNUmed
1	1	1	1	55	Medical
1	1	1	1	53	FreeMed
3	3	1	1	51	Tolven
1	1	1	1	43	Open MRS
1	1	1	1	39	TORCH

Abbildung 1.4.. Tabelle Ergebnisse mit Punktetotal Teil 4 von 4

A.1.2. Details zur Evaluation

System	Kriterium	Ergebnis	Bemerkung	Gewichtung
OpenEMR	Client/Server	ja	Web-Client (XAMPP)	3
	Support	3	Professioneller Support durch viele Firmen gegeben (kommerziell), sonst viele Manuals	2
	ICD-Anbindung	ja		1
	HL7-Schnittstelle	3	Hinweis in Dokumentation gegeben	2
	DICOM-Schnittstelle	1		2
	Anzahl Lösungen	1	Erwähnte angehörige Projekte: 2; sonstige Zahl unbekannt; angeblich 3400 Downloads pro Monat	2
	System-Unterhalt	3	Im Mai des Jahres bereits 4. Patch	2
	Import bestehender Daten	1	schlecht/nicht möglich	2
	Web-Zugriff für Patienten	nein		1
	Decision-Support-Tools	1		1
	Multimedia/PDF	ja	Upload von Bildern und Dokumenten möglich	3
	Zugriff über mobile Geräte	nein	keine mobile Version (Zugriff Webinterface möglich)	2
	Erweiterbarkeit	ja	PHP(XAMPP)	3
	API	ja	Möglichkeiten direkt auf Patientendaten zuzugreifen, z.B. getPatientData(); Datenbank	3
	Lizenz		GPL	

Abbildung 1.5.. Detail Evaluation Open EMR

FreeMed	Client/Server	ja	Web-Client	3
	Support	3	Community und kommerziell	2
	ICD-Anbindung	nein		1
	HL7-Schnittstelle	2		2
	DICOM-Schnittstelle	1		2
	Anzahl Lösungen	1	nicht bekannt	2
	System-Unterhalt	3	Laufende Entwicklung, Roadmap	2
	Import bestehender Daten	1	schlecht/nicht möglich	2
	Web-Zugriff für Patienten	nein		1
	Decision-Support-Tools	1		1
	Multimedia/PDF	ja	Bilder als Anhang möglich (FCK-Editor)	3
	Zugriff über mobile Geräte	nein	keine mobile Version (Zugriff Webinterface möglich)	2
	Erweiterbarkeit	ja	PHP	3
	API	ja	nur Zugriff auf DB, viele Funktionen in Skripts gekapselt	3
	Lizenz		GPL	

Abbildung 1.6.. Detail Evaluation Freemed

TORCH	Client/Server	ja	Web-Client	3
	Support	1	Mailing-List (Website nicht mehr aktiv)	2
	ICD-Anbindung	nein		1
	HL7-Schnittstelle	1		2
	DICOM-Schnittstelle	1		2
	Anzahl Lösungen	1	Keine Angaben	2
	System-Unterhalt	1	Letztes Update im Jahr 2004	2
	[illegible]	1	[illegible]	2
	Web-Zugriff für Patienten	nein		1
	Decision-Support-Tools	nein		1
	Multimedia/PDF	ja	Nur Bild-Upload möglich	3
	Zugriff über mobile Geräte	nein	keine mobile Version (Zugriff Webinterface möglich)	2
	Enweiterbarkeit	ja	Python	3
	API	nein	keine Angaben	3
	Lizenz		GPL	

Abbildung 1.7.. Detail Evaluation TORCH

Tolven	Client/Server	ja	Web Client	3
	Support	2	nicht bekannt, aber scheint eher kommerziell zu sein	2
	ICD-Anbindung	nein		1
	HL7-Schnittstelle	3	Sogar Unterstützung für HL7 V3	2
	DICOM-Schnittstelle	1		2
	[illegible]	1	[illegible]	
2	[illegible]		2 keine Angaben, News von 2009, Wiki-Updates von 2009	
2	[illegible]		1 schlecht/nicht möglich	
	[illegible]		ja Möglichkeit, für sich selbst eine Akte anzulegen und Rechte	
1	[illegible]		nein	
3	[illegible]		ja Alle Dateitypen möglich	
2	[illegible]		ja	
3	[illegible]		ja Java	
3	[illegible]		ja ähnlich Java-API	
	[illegible]		[illegible]	

Abbildung 1.8.. Detail Evaluation Tolven

GNUmed	Client/Server	ja	Rich Client	3
	Support	3	Community/Kommerziell	2
	ICD-Anbindung	nein	andere Anbindungen (Free Diams, Gelbe Liste)	1
	HL7-Schnittstelle	1		2
	DICOM-Schnittstelle	2	DICOM-Viewer eingebaut	2
	Anzahl Lösungen	2	3 Success Stories, wahrscheinlich viele weitere realisierte Lösungen	2
	System-Unterhalt	3	Laufende Entwicklung, Roadmap	2
	Import bestehender Daten	2	Bestehende Daten aus alten GNUmed-Installationen, "German Medical Cards"	2
	Web-Zugriff für Patienten	nein		1
	Decision-Support-Tools	nein		1
	Multimedia/PDF	ja	Alle Dateitypen möglich, auch Daten von Kameras/Scannern	3
	Zugriff über mobile Geräte	nein		2
	Erweiterbarkeit	ja	Python	3
	API	ja	Epydoc	3
	Lizenz		GPL	

Abbildung 1.9.. Detail Evaluation GNUmed

Medical	Client/Server	ja	Rich Client (basiert auf Open ERP, ist ein Plugin) und Web Client	3
	Support	3		2
	ICD-Anbindung	ja		1
	HL7-Schnittstelle	3		2
	DICOM-Schnittstelle	1		2
	Anzahl Lösungen			2
	System-Unterhalt	3	Laufende Entwicklung, neueste Version April 2010; viele Mitwirkende	2
	Import bestehender Daten	1		2
	Web-Zugriff für Patienten	nein		1
	Decision-Support-Tools	nein		1
	Multimedia/PDF	ja	Bilder/Dokumente können "attached" werden	3
	Zugriff über mobile Geräte	nein		2
	Erweiterbarkeit	ja	Python	3
	API		zur Zeit keine Angaben	3
	Lizenz		GPL	

Abbildung 1.10.. Detail Evaluation Medical

A.2. Komplette Listings

Listing A.1. Datei gmTimelinePlugin.py

```
from Gnumed.wxpython import gmPlugin, gmTimelineWidgets

#============================================================================

#Klassenname geaendert
class gmTimelinePlugin(gmPlugin.cNotebookPlugin):

        #Tabname geaendert
        tab_name = _('Timeline')
        #------------------------------------------------------------------
        def __init__(self):
                gmPlugin.cNotebookPlugin.__init__(self)
        #------------------------------------------------------------------
        #Klassennamen hier eingesetzt
        def name(self):
                return gmTimelinePlugin.tab_name
        #------------------------------------------------------------------
        #Hier muss noch der Panelname eingesetzt werden
        def GetWidget(self, parent):
                self._widget = gmTimelineWidgets.cTimelinePnl(parent, -1)
                return self._widget
        #------------------------------------------------------------------
        #Menueeintrag geaendert (Tools bleibt)
        def MenuInfo(self):
                return ('tools', _('&Timeline'))
        #------------------------------------------------------------------
        def can_receive_focus(self):
                return True
```

Listing A.2. Datei gmTimelineWidgets.py

```
import sys, logging, datetime as pyDT, decimal, StringIO
import wx       #, wx.grid

if __name__ == '__main__':
        sys.path.insert(0, '../../')

#Imports fuer Rechnung mit Jahren
from datetime import datetime

#GNUmed-spezifische Imports
from Gnumed.business import gmPerson, gmDevices, gmDocuments,
gmDemographicRecord
from Gnumed.pycommon import gmDispatcher, gmMatchProvider
from Gnumed.wxpython import gmRegetMixin, gmGuiHelpers,
gmPatSearchWidgets
from Gnumed.wxGladeWidgets import wxgTimelinePnl

#============================================================
class cTimelinePnl(wxgTimelinePnl.wxgTimelinePnl):

        def __init__(self, *args, **kwargs):
                wxgTimelinePnl.wxgTimelinePnl.__init__(self, *args, **kwargs)

                #Setze Breite der Panels auf ungefaehre Bildschirmgroesse
                self.PANEL_WIDTH = 1200

                #Zeichne das Panel
                self.panel = wx.Panel(self, size=(self.PANEL_WIDTH, 800))
                self.Fit()

                #Die Funktion on_paint wird an den Event EVT_PAINT gekoppelt
                self.panel.Bind(wx.EVT_PAINT, self.on_paint)

                #Hole Patientendaten
                self.curr_pat = gmPerson.gmCurrentPatient()

        def on_paint(self, event):
                dc = wx.PaintDC(self.panel)
                dc.SetPen(wx.Pen('white', 4))

                #Zeichnen der weissen Panels
                dc.DrawRectangle(10, 40, self.PANEL_WIDTH, 150)
                dc.DrawRectangle(10, 200, self.PANEL_WIDTH, 150)
                dc.DrawRectangle(10, 360, self.PANEL_WIDTH, 150)
                dc.DrawRectangle(10, 520, self.PANEL_WIDTH, 50)

                #Ermittle das aktuelle Datum
                now = datetime.now()

                #Ziehe Geburtsjahr vom aktuellen Jahr ab und ermittle Jahre
                years = int(now.strftime('%Y'))-int(self.curr_pat.
                    get_formatted_dob(format = '%Y'))

                # Ermittle Breite, die ein Jahr im Panel haben darf
                offset = (self.PANEL_WIDTH / years)

                #Beschriftung fuer Zeitachse
                dc.SetPen(wx.Pen('grey', 1))
```

```
dc.SetFont(wx.Font(12, wx.MODERN, wx.NORMAL, wx.NORMAL))
dc.DrawText("DOB", 10, 580)
dc.DrawText(self.curr_pat['medical_age'], 1140, 580)

#Zeichne Linien der Zeitachse
for i in range(1, years):
        dc.DrawLine(10+(i*offset), 560, 10+(i*offset), 570)

#BLOCK DOKUMENTE
#Hole Daten zu Dokumenten
docs_folder = self.curr_pat.get_document_folder()
docs = docs_folder.get_documents()

for doc in docs:
        years_docs = int(now.strftime('%Y'))-int(doc['clin_when
            '].strftime('%Y'))

        #Pruefen welcher Dateityp und entsprechendes Bild laden
        if str(doc['l10n_type']) == 'Dokument':
                imageFile = 'C:\\Programme\\GNUmed-client\\bin
                    \\bitmaps\\page_white_text.png'
                offset_height = 100
        elif str(doc['l10n_type']) == 'sonstiger␣Arztbrief':
                imageFile = 'C:\\Programme\\GNUmed-client\\bin
                    \\bitmaps\\page_white_text.png'
                offset_height = 100
        elif str(doc['l10n_type']) == 'Bild':
                imageFile = 'C:\\Programme\\GNUmed-client\\bin
                    \\bitmaps\\camera.png'
                offset_height = 260
        elif str(doc['l10n_type']) == 'Video':
                imageFile = 'C:\\Programme\\GNUmed-client\\bin
                    \\bitmaps\\webcam.png'
                offset_height = 420
        else:
                imageFile = 'C:\\Programme\\GNUmed-client\\bin
                    \\bitmaps\\box.png'
                offset_height = 450
        image = wx.Image(imageFile, wx.BITMAP_TYPE_ANY).
            ConvertToBitmap()
        dc.DrawBitmap(image, self.PANEL_WIDTH-(years_docs*
            offset), offset_height, True)

        dc.DrawText(str(doc['comment']), self.PANEL_WIDTH-(
            years_docs*offset), offset_height+15)
        dc.DrawText(str(doc['clin_when'].strftime('%Y')), self.
            PANEL_WIDTH-(years_docs*offset), offset_height+25)

#BLOCK MEDIKAMENTE
emr = self.curr_pat.get_emr()
meds = emr.get_current_substance_intake()

#Hole Daten wie lange Patient Medis nimmt (nur Jahre)
s = str(meds[0]['duration'])
s = s.partition("␣")

sys.stdout.write(s[0])

#length_med steht fuer die Laenge der Medikamenteneinnahme
```

```
length_med = int(s[0])/365

#Ermittle Startjahr der Einnahme
year_med = int(now.strftime('%Y'))-int(meds[0]['started'].
    strftime('%Y'))

#Zeichnen der Medis
dc.SetPen(wx.Pen('blue', 2))

#Linie vom Beginn bis Ende
dc.DrawLine(self.PANEL_WIDTH-(year_med*offset), 530, self.
    PANEL_WIDTH-(year_med*offset)J(e)g9(+6(3zP96ld Q[5T5h6J01.)-4e)-40(a)-39(r)-39(_)-40(m)-
```

Listing A.3. Datei wxgTimelinePnl.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# generated by wxGlade 0.6.3 on Mon Jun 21 13:44:31 2010

import wx

# begin wxGlade: extracode
# end wxGlade



class wxgTimelinePnl(wx.ScrolledWindow):
    def __init__(self, *args, **kwds):
        # begin wxGlade: wxgTimelinePnl.__init__
        kwds["style"] = wx.NO_BORDER|wx.TAB_TRAVERSAL
        wx.ScrolledWindow.__init__(self, *args, **kwds)
        self.datepicker_ctrl_1 = wx.DatePickerCtrl(self, -1)
        self.datepicker_ctrl_2 = wx.DatePickerCtrl(self, -1)
        self.anzeige = wx.TextCtrl(self, -1, "")
        self.panel_2 = wx.Panel(self, -1)
        self._BTN_neuzeichnen = wx.Button(self, -1, _("Neu zeichnen"), style=wx
            .BU_EXACTFIT)
        self.button_1 = wx.ToggleButton(self, -1, _("Bilder"))
        self.button_2 = wx.ToggleButton(self, -1, _("Videos"))
        self.button_3 = wx.ToggleButton(self, -1, _("Dokumente"))
        self.button_4 = wx.ToggleButton(self, -1, _("Aufenthalt Spital"))
        self.button_5 = wx.ToggleButton(self, -1, _("Operationen"))
        self.button_6 = wx.ToggleButton(self, -1, _("Medikation"))

        self.__set_properties()
        self.__do_layout()

        self.Bind(wx.EVT_BUTTON, self._on_neuzeichnen_button_pressed, self.
            _BTN_neuzeichnen)
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: wxgTimelinePnl.__set_properties
        self.SetScrollRate(10, 10)
        self._BTN_neuzeichnen.SetDefault()
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: wxgTimelinePnl.__do_layout
        __szr_main = wx.BoxSizer(wx.VERTICAL)
        __szr_buttons = wx.BoxSizer(wx.HORIZONTAL)
        __szr_top = wx.BoxSizer(wx.HORIZONTAL)
        sizer_1 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_2 = wx.BoxSizer(wx.HORIZONTAL)
        __lbl_filter = wx.StaticText(self, -1, _("Zeitraum:"))
        __szr_top.Add(__lbl_filter, 0, wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 10)
        __szr_top.Add(self.datepicker_ctrl_1, 0, 0, 0)
        __szr_top.Add(self.datepicker_ctrl_2, 0, 0, 0)
        __szr_top.Add(self.anzeige, 0, 0, 0)
        sizer_2.Add((600, 20), 0, 0, 0)
        sizer_1.Add(sizer_2, 1, wx.EXPAND, 0)
        __szr_top.Add(sizer_1, 1, wx.EXPAND, 0)
        __szr_main.Add(__szr_top, 0, wx.LEFT|wx.RIGHT|wx.TOP|wx.EXPAND, 3)
```

```
        __szr_main.Add(self.panel_2, 1, wx.EXPAND, 0)
        __szr_buttons.Add((20, 20), 2, wx.EXPAND, 0)
        __szr_buttons.Add(self._BTN_neuzeichnen, 0, wx.RIGHT|wx.
            ALIGN_CENTER_VERTICAL, 5)
        __szr_buttons.Add((100, 20), 0, wx.EXPAND, 0)
        __szr_buttons.Add(self.button_1, 0, 0, 0)
        __szr_buttons.Add(self.button_2, 0, 0, 0)
        __szr_buttons.Add(self.button_3, 0, 0, 0)
        __szr_buttons.Add((100, 20), 0, wx.EXPAND, 0)
        __szr_buttons.Add(self.button_4, 0, 0, 0)
        __szr_buttons.Add(self.button_5, 0, 0, 0)
        __szr_buttons.Add(self.button_6, 0, 0, 0)
        __szr_buttons.Add((20, 20), 2, wx.EXPAND, 0)
        __szr_main.Add(__szr_buttons, 0, wx.TOP|wx.BOTTOM|wx.EXPAND, 5)
        self.SetSizer(__szr_main)
        __szr_main.Fit(self)
        # end wxGlade

    def _on_activate_button_pressed(self, event): # wxGlade: wxgTimelinePnl.<
        event_handler>
        print "Event␣handler␣`_on_activate_button_pressed'␣not
```