

Fachhochschule Vorarlberg GmbH



Bachelorarbeit im Fachhochschul-Bachelorstudiengang
Informatik - Software and Information Engineering

**Anbindung des Mythic22 Analysegerätes an das
Praxisprogramm *Elexis***

ausgeführt von

Christian Gruber

O810247001

Dornbirn, im Juli 2011

Betreuer Fachhochschule:	Prof. (FH) Dipl.-Inform. Thomas Feilhauer
Betreuer Fa. Medevit:	DI (FH) Marco Descher, MSc
	DI (FH) Thomas Huster

Eidesstattliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, Juli 2011

Christian Gruber

Zusammenfassung

Das Hauptaugenmerk dieser Arbeit liegt auf der Beschreibung der Vorgehensweise während der Implementierung einer Anbindung des Blutanalysegerätes *Mythic22* an das Praxisinformationssystem *Elexis*.

Diese Arbeit gibt einen Einblick in das Praxisinformationssystem *Elexis* und beschreibt näher, worum es sich dabei handelt. Des Weiteren wird das Blutanalysegerät *Mythic22* genauer beschrieben.

Es werden auch für diese Arbeit relevante Technologien näher beschrieben. Zu diesen Technologien zählen unter anderen die „Eclipse Rich Client Plattform“, „OSGI“ und „JUnit-Tests“. Im Zuge der Erklärung von „JUnit-Tests“ wird auch der Ansatz der „Test-Driven“ Entwicklung näher erläutert.

Im Laufe der genaueren Betrachtung der Implementierung der Anbindung des Blutanalysegerätes *Mythic22* an das Praxisinformationssystem *Elexis* werden vorangehende Planungsphasen und später folgende Probleme sowie deren Lösungen näher beschrieben.

Zuletzt beinhaltet diese Arbeit eine Erläuterung der persönlichen Erfahrungen, welche bei der Entwicklung dieser Anbindung gezogen werden konnten, einen Ausblick auf mögliche Weiterentwicklungen des Erarbeiteten sowie ein kurzes Kapitel über eine mögliche Realisierung von sogenannten „Callback Funktionen“ in Java.

Das Ergebnis dieser Arbeit besteht aus einem Plug-In für die Software *Elexis*, welches das Blutanalysegerät *Mythic22* anbindet und es ermöglicht, Daten von dem Gerät *Mythic22* zu empfangen und diese in die *Elexis* Datenbank einzufügen. Des Weiteren umfasst das Plug-In User Interface Komponenten, welche es ermöglichen, Einstellungen vorzunehmen und die automatische Verarbeitung eingehender Daten zu aktivieren beziehungsweise deaktivieren.

Abstract

The main focus of this bachelor thesis is the description of the implementation of a connection of the blood analyzer *Mythic22* to the doctor's office information system *Elexis*.

This bachelor thesis gives an insight into the doctor's office information system *Elexis*.

Furthermore, the blood analysis device *Mythic22* is described in more detail.

Also technologies related to this implementation are described. These technologies include, among others, "Eclipse Rich Client Platform", "OSGI" and "JUnit tests". As part of the explanation of "JUnit tests" also the approach of "test-driven" development is discussed.

Besides the implementation process this bachelor thesis describes encountered problems and their solutions in detail.

The last part of this thesis consists of personal drawn experiences, an outlook on possible further improvements and a short chapter describing a possible realization of so-called "callback functions" in Java.

The result of this work consists of a plug-in for the software *Elexis*, which connects the blood analyzer *Mythic22* and allows to automatically receive data from the device *Mythic22* and to insert it into the database of *Elexis*. The plug-in further contains user interface components, which make it possible to change settings and enable or disable automatic processing of incoming data.

Inhaltsverzeichnis

1.	Einleitung.....	6
2.	Elexis.....	7
3.	Analysegerät Mythic22.....	9
4.	Verwendete Technologien	10
4.1.	RCP.....	10
4.2.	Eclipse RCP.....	10
4.3.	OSGI	11
4.4.	JUnit Tests	12
5.	Implementierung.....	13
5.1.	Allgemeiner Entwicklungsansatz.....	13
5.2.	Allgemeine Vorgehensweise	14
5.3.	Anbindung des Analysegerätes Mythic22	15
5.3.1.	Ausgangsstandpunkt	15
5.3.2.	Wahl der Schnittstelle	15
5.3.3.	Vorgehensweise	16
5.3.4.	Aufgetretene Probleme und deren Lösungen.....	17
5.3.5.	Ergebnisse.....	17
5.4.	Einspeisen der Analysedaten in Elexis.....	21
5.4.1.	Ausgangsstandpunkt	21
5.4.2.	Vorgehensweise	21
5.4.3.	Ergebnisse.....	24
5.5.	Integrieren der Implementierung in Elexis.....	24
6.	Ergebnisse.....	27
6.1.	Fazit	27
6.2.	Anmerkungen	27
6.3.	Persönlich gezogene Erfahrungen.....	28
6.4.	Ausblick.....	28
7.	Anhang.....	32
7.1.	Callback in Java	32

1. Einleitung

Das Praxisinformationssystem *Elexis* baut auf der Grundlage von *Eclipse* auf und ist wie Eclipse selbst ebenfalls eine Open Source Software. Dies bedeutet, dass das Basisprogramm frei erhältlich und wie jede in Java geschriebene Software auf den meisten Betriebssystemen lauffähig ist. Diese Software ist zurzeit vor allem in der Schweiz bei diversen Ärzten in Verwendung und kann durch die Entwicklung von Plug-Ins verändert und erweitert werden.

Elexis ist von einem Hausarzt (Dr. med. Gerry Weirich) entwickelt und programmiert worden. Eine Gruppe interessierter Ärzte hat zur Entwicklung beigetragen und dafür gesorgt, dass *Elexis* benutzerfreundlich und für alle Ärztinnen und Ärzte geeignet ist. (vgl. Dr. Med. Zürcher, 2009, S.2)

Im Gegensatz zu *Elexis* sind bekannte Praxisprogramme schon vor Jahren konzipiert worden. Datenbanken und Programmiersysteme haben sich jedoch weiterentwickelt. Für *Elexis* wurden bewährte, robuste Datenbanken und die modernste Programmier Technologie *Eclipse* (Java) verwendet. Sie erlauben die Anpassung an die technologische Entwicklung und garantieren für Zukunftssicherheit. (vgl. Dr. Med. Zürcher, 2009, S.3)

Diese Arbeit behandelt die Anbindung des Blutanalysegerätes *Mythic22* an die Software *Elexis*. Das Gerät *Mythic22* sollte angebunden werden, da es in der Praxis von *Doktor Wolber* verwendet wird und Interesse an einer Anbindung bestand. Des Weiteren hatte ich die Möglichkeit, es dort im Einsatz zu beobachten.

Die Anbindung soll es ermöglichen Blutwerte, welche von *Mythic22* gesendet werden, automatisch entgegenzunehmen und direkt in *Elexis* einzuspielen, wo sie gespeichert und verwaltet werden können. Hierzu sollen auch GUI Elemente existieren, um Einstellungen für die Anbindung zu tätigen und das automatische Empfangen der Daten zu starten beziehungsweise zu stoppen.

2. Elexis

Das Praxisinformationssystem *Elexis* baut auf der Grundlage von *Eclipse* auf und ist wie Eclipse selbst ebenfalls eine Open Source Software. Dies bedeutet, dass das Basisprogramm frei erhältlich und wie jede in Java geschriebene Software auf den meisten Betriebssystemen lauffähig ist. Diese Software ist zurzeit vor allem in der Schweiz bei diversen Ärzten in Verwendung und kann durch die Entwicklung von Plug-Ins verändert und erweitert werden.

Elexis ist von einem Hausarzt (Dr. med. Gerry Weirich) entwickelt und programmiert worden. Eine Gruppe interessierter Ärzte hat zur Entwicklung beigetragen und dafür gesorgt, dass Elexis benutzerfreundlich und für alle Ärztinnen und Ärzte geeignet ist. (vgl. Dr. Med. Zürcher, 2009, S.2)

Zum Zeitpunkt der Verfassung dieser Arbeit ist vor allem die Firma *Medevit* an der Weiterentwicklung der Software *Elexis* beteiligt. Ein Hauptziel der Firma *Medevit* ist es, die Software für den österreichischen Markt tauglich zu machen, da sich das österreichische Gesundheitssystem doch signifikant von dem Gesundheitssystem der Schweiz unterscheidet.

Im Gegensatz zu *Elexis* sind bekannte Praxisprogramme schon vor Jahren konzipiert worden. Datenbanken und Programmiersysteme haben sich jedoch weiterentwickelt. Für *Elexis* wurden bewährte, robuste Datenbanken und die modernste Programmier Technologie *Eclipse* (Java) verwendet. Sie erlauben die Anpassung an die technologische Entwicklung und garantieren für Zukunftssicherheit. (vgl. Dr. Med. Zürcher, 2009, S.3)

Elexis dient dazu, Praxisinformationen und Krankengeschichten in einer Ärztepraxis zu verwalten. Elexis bietet gegenüber der herkömmlichen Krankengeschichte auf Papier viele Vorteile:

- Medizinische und administrative Daten der Patienten stehen sofort zur Verfügung
- Keine zeitraubende Vor- und Nachbereitung von Patientendossiers
- Zugriff auf Daten auch von zuhause oder von unterwegs
- Mitwirkung an der Softwareentwicklung - Anwenderwünsche fließen in Elexis ein
- Elexis kann sehr flexibel an die Bedürfnisse der Benutzer angepasst werden

- Geräteanbindungen und andere Funktionen können einfach als Plug-In in Elexis integriert werden
- Überregionales Supportnetz - Elexis wird von regionalen Partnern aus den Bereichen IT und Gesundheitsinformatik vertrieben, installiert und betreut. (vgl. Zur Rose Ärzte AG, 2010, S.2)

Die einzige Schwäche der Software Elexis ist der starke Fokus auf die Schweiz. Die Software ist nur in deutscher Sprache verfügbar und für das Gesundheitssystem der Schweiz ausgelegt. Jedoch arbeitet bereits die Firma *Medevit* daran, die Software für den österreichischen Markt tauglich zu machen.

3. Analysegerät Mythic22

Mythic22 ist ein Hämatologie System, welches Blut analysiert und 22 Werte liefern kann. Das Gerät stellt des Weiteren 3 Diagramme dar, welche sich auf diese Werte beziehen.

Die Auswertung der Blutprobe von *Mythic22*, im speziellen die 22 Blutwerte, können dem entsprechenden Patienten in dem Praxisinformationssystem *Elexis* zugeordnet und gespeichert werden. Die Eingabe der Daten in *Elexis* erfolgt ohne passendes Plug-In per Hand durch die Eingabe über eine Tastatur.

Das Gerät *Mythic22* verfügt sowohl über eine COM Port Schnittstelle als auch über eine Ethernet Schnittstelle. Über diese Schnittstellen lässt sich das Ergebnis einer Blutanalyse an andere Geräte übertragen. Die in dieser Arbeit behandelte Anbindung an das Gerät *Mythic22* verwendet die Ethernet Schnittstelle (siehe hierzu Kapitel „5.3.2 Wahl der Schnittstelle“).

Die Analysedaten, die von dem Gerät *Mythic22* übertragen werden, bestehen aus Zeichenketten (Strings) und folgen einem bestimmten Aufbau, welcher aus der Spezifikation des Gerätes hervorgeht (siehe auch Abbildung 1 - Ausschnitt gesendeter Analysedaten von *Mythic22*).

```
15 WBC;7.0...;;2.0...;4.0...;12.0...;15.0 CR
16 RBC;4.28...;;2.50...;4.00...;6.20...;7.00 CR
17 HGB;13.5...;;8.5...;11.0;17.0;19.0 CR
18 HCT;41.4...;;25.0;35.0;55.0;60.0 CR
19 PLT;261...;;70...;150...;400...;500... CR
20 LYM;1.0...;;0.7...;1.0...;5.0...;5.5... CR
21 MON;0.4...;;0.0...;0.1...;1.0...;1.1... CR
22 NEU;5.3...;;1.5...;2.0...;8.0...;9.0... CR
23 LYM%;14.5...;L;15.0;25.0;50.0;55.0 CR
24 MON%;6.1...;;1.0...;2.0...;10.0;12.0 CR
25 NEU%;75.3...;;45.0;50.0;80.0;85.0 CR
26 MCV;96.7...;;70.0...;80.0...;100.0;120.0 CR
```

Abbildung 1 - Ausschnitt gesendeter Analysedaten von *Mythic22*

4. Verwendete Technologien

4.1.RCP

Die Grundidee der Rich Clients war es, dem Endnutzer ein „reiches“ Nutzererlebnis durch die Software zu bieten.

Rich Clients fördern ein qualitativ hochwertiges Nutzererlebnis für den Endnutzer durch die Bereitstellung vielfältiger nativer Benutzeroberflächen sowie durch schnelle lokale Datenverarbeitung. Rich Client Benutzeroberflächen unterstützen native Funktionen wie Drag & Drop, Copy & Paste, etc. Ein gut umgesetzter Rich Client erlaubt es dem Endnutzer, sich auf seine Arbeit zu konzentrieren, ohne viele Gedanken an das System zu verschwenden.

(vgl. McAffer; Lemieux; Aniszczyk, 2010, S.3)

4.2.Eclipse RCP

Eclipse ist sehr gut geeignet, um Rich Client Applikationen zu erstellen, da es viele Vorteile früherer Rich und Thin Client Ansätze vereint und deren Nachteile berücksichtigt. Dies ist unter anderem in folgenden Punkten zu erkennen:

- **Komponenten** - *Eclipse* enthält ein robustes Komponentensystem. Auf *Eclipse* basierte Systeme bestehen aus Komponenten, welche als Plug-Ins bekannt sind. Diese Plug-Ins können von mehreren *Eclipse*-basierten Applikationen genutzt werden. Plug-Ins haben eine Versionsnummer, welche es ermöglicht, verschiedene Versionen desselben Plug-Ins gleichzeitig installiert zu haben. Dieser Ansatz ermöglicht es Applikationen, sich durch das Hinzufügen oder Ersetzen von Plug-Ins weiterzuentwickeln.
- **Middleware** – Das zuvor erwähnte Komponentensystem wird von Frameworks getragen, welche das Programmieren von Clients stark erleichtern. Kurz gesagt ist die

Eclipse Rich Client Plattform der Teil der Middleware, welche skalierbare UIs, Kontext sensitive Hilfe, Fehlerbehandlung, und vieles mehr beinhaltet, was nicht bei der Entwicklung eines Clients neu erfunden werden muss.

- **Natives Nutzererlebnis** – Das „Eclipse Standard Widget Toolkit“ (SWT) erlaubt es grafische Benutzeroberflächen zu erstellen, welche der nativen Oberfläche des Betriebssystems ähneln.
- **Übertragbarkeit** – Da *Eclipse Rich Clients* in Java programmiert werden, erben sie auch Javas berühmte Eigenschaft, auf allen Systemen zu funktionieren, so lange eine *Java Virtual Machine (JVM)* installiert ist.
- **Komponenten Bibliotheken** – Die Eclipse Community hat eine umfassende Reihe von Komponenten produziert, womit Applikationen entwickelt werden können.

(vgl. McAffer; Lemieux; Aniszczyk, 2010, S.5 - 6)

4.3.OSGI

OSGI ist eigentlich ein Adjektiv und bezieht sich auf die *OSGI Alliance*, eine Organisation, welche Spezifikationen erstellt, wie die „OSGI Service Plattform Specification R4“. Seit der *Eclipse* Version 3.0 basiert *Equinox*, die Laufzeit von *Eclipse*, auf diesen Spezifikationen. (vgl. The Eclipse Foundation, 2011, S.1)

Die *OSGI Alliance* wurde, etwa zur gleichen Zeit als das *Eclipse* Projekt startete, gebildet. Die ursprüngliche Aufgabe der *OSGI Alliance* war es, ein Java Komponenten- und Servicemodell für den Bau von Embedded-Geräten zu entwickeln. Durch den Fokus auf eine *Rich Client Plattform* während der Entwicklung von *Eclipse 3.0* entstand das *Equinox Projekt*. Dieses Projekt untersuchte neue Wege, um Eclipse dynamischer zu machen und die Installation von Plug-Ins zu unterstützen. Im Laufe der Zeit wurde *OSGI*, ähnlich wie *Eclipse*, zu einem Standard unter den dynamischen Frameworks. Als Ergebnis basiert *Eclipse* auf der *Equinox*-Implementierung der *OSGI-Framework*-Spezifikation. (vgl. McAffer; Lemieux; Aniszczyk, 2010, S.21)

4.4.JUnit Tests

Ein Unit Test prüft das Verhalten einer abgegrenzten Einheit einer Arbeit. Dies ist in einer Java Applikation meist eine einzelne Methode. (vgl. Tahchiev; Leme; Massol, Gregory, 2009, S.4)

Will man *JUnit* Tests optimal anwenden, sollte man in Betracht ziehen, die „Test-Driven“ Entwicklung zu verfolgen. Bei dieser Methode werden zuerst die Schnittstellen der Methoden definiert. Anschließend werden *JUnit* Tests für diese Methoden geschrieben und als letzter Schritt wird der Code verfasst, welcher diesen Test erfolgreich besteht. Durch diese Entwicklungsmethode verändert sich der Entwicklungszyklus wie folgt:

Der konventionelle Entwicklungszyklus beinhaltet folgende Schritte in dieser Reihenfolge:

- Programmieren
- Testen
- (vorherige Schritte wiederholen, wenn nötig)

Der Entwicklungszyklus bei der Benutzung der „Test-Driven“ Entwicklung sieht folgendermaßen aus:

- Testen
- Programmieren
- (vorherige Schritte wiederholen, wenn nötig)

(vgl. Tahchiev; Leme; Massol, Gregory, 2009, S.74)

Der Ansatz der „Test-Driven“ Entwicklung ist prinzipiell eine Vorgehensweise, die sehr viele Vorteile mit sich bringt:

- eine sehr hohe Wartbarkeit des geschriebenen Codes
- Fehlerquellen können bei Änderungen am Code schnell ausfindig gemacht werden
- Code wird vor dem Schreiben besser durchdacht

Ein Nachteil dieses Ansatzes ist, dass man in der Regel natürlich mehr Zeit benötigt, um zu einem ersten funktionierenden Code zu gelangen.

5. Implementierung

5.1. Allgemeiner Entwicklungsansatz

Ich habe während der Implementierung des Plug-Ins, für die Anbindung des Gerätes *Mythic22* an die Software *Elexis*, den Ansatz der „Test-Driven“ Entwicklung verfolgt. Dieser Ansatz wurde jedoch nicht durchgehend konsequent eingesetzt, mehr dazu in Kapitel „6.3 Persönlich gezogene Erfahrungen“.

Ein Hauptaugenmerk während der Entwicklung dieses Plug-Ins lag darauf, den Weg der Daten von dem Gerät *Mythic22* bis hin zur Datenbank der Software *Elexis* in einzelne Teile einzuteilen, welche dann später zusammenarbeiten sollten. Diese Teile sind:

- **Netzwerkverbindung zum Blutanalysegerät *Mythic22*** – dieser Teil übergibt die empfangenen Daten in ihrer unverarbeiteten Form an den nächsten Teil
- **Parser** – dieser Teil verarbeitet die empfangenen Daten weiter zu Objekten, welche komfortabel weiterverarbeitet werden können
- **Persistenz** – dieser Teil nimmt die nötigen Daten aus den vom vorherigen Teil erstellen Objekten und speichert diese in die Datenbank

Dieses Vorgehen soll es dem Programmierer erleichtern, den „Test-Driven“ Entwicklungsansatz besser verfolgen zu können, da für jede dieser Unterteilungen ein Unit Test verfasst werden kann, der sicher stellt, dass dieser Teil funktioniert. Dies kommt natürlich späterem Debuggen zugute, da klar gesehen werden kann, welche Teile ihre Unit Tests nicht bestehen.

Des Weiteren soll das Unterteilen des Programms in Abschnitte es dem Programmierer erleichtern, sich auf diese kleinen Probleme zu konzentrieren ohne von der Größe des ganzen Projekts abgelenkt zu werden.

5.2. Allgemeine Vorgehensweise

Da das zu entwickelnde Plug-In ein technisches Gerät an eine bereits bestehende Software anbinden soll, ist ein erster Schritt der Vorgehensweise das Kennenlernen dieses Gerätes. Ich konnte dieses Gerät dank der Kontakte meiner Betreuer in der Praxis von Doktor *Wolber* im Einsatz beobachten. Dies ermöglichte nicht nur ein sehr viel besseres Verständnis des Blutanalysegerätes *Mythic22*, sondern demonstrierte auch einen typischen Anwendungsfall („Use-Case“) für die Verwendung dieses Gerätes. Dieser Anwendungsfall umfasste folgende Schritte:

- Eingabe der Patienten ID am Gerät *Mythic22*
- Einlegen der Blutprobe
- Start der Analyse
- Auswahl des entsprechenden Patienten in der Software *Elexis*
- Ablesen und Übertragen der Ergebnisse von *Mythic22* in *Elexis*

Die Analyse dieses Anwendungsfalls gab Aufschluss darüber, in welcher Art und Weise das zu programmierende Plug-In funktionieren sollte. So ergab die Analyse unter anderem, dass die ID des Patienten am Gerät *Mythic22* angegeben wird, um zusammen mit den Ergebnissen der Blutanalyse des Gerätes per Ethernet versendet zu werden. Dies wiederum beeinflusste die Planung des Plug-Ins deutlich, da es anders als ich zuvor angenommen hätte, möglich war die Ergebnisse der Blutanalyse ohne Eingreifen des Endnutzers dem richtigen Patienten zuzuordnen.

5.3. Anbindung des Analysegerätes Mythic22

5.3.1. Ausgangsstandpunkt

Vor Beginn meiner Arbeit bestand keine Anbindung zwischen dem Praxisinformationssystem *Elexis* und dem Blutanalysegerät *Mythic22*. Es existierte jedoch eine Anbindung für ein ähnliches Gerät namens *Mythic18*. Die Anbindung des Gerätes *Mythic18* sendet ihre Daten über eine COM Port Schnittstelle.

Die grafische Oberfläche (Knöpfe und Einstellungsseite), der in dieser Arbeit beschriebenen Implementierung der Anbindung für das Gerät *Mythic22*, orientiert sich an der Implementierung der Anbindung des Gerätes *Mythic18*.

Vor Beginn meiner Arbeit bestand bereits eine passende Datenstruktur in *Elexis*, welche Blutwerte speichern kann. Diese Datenstruktur wird als Laborwerte bezeichnet und wird nicht nur für Blutwerte, sondern viel mehr für alle Analysedaten eines Patienten verwendet.

5.3.2. Wahl der Schnittstelle

Wie im Kapitel „3 Analysegerät Mythic22“ bereits erwähnt wurde, verfügt das Gerät *Mythic22* sowohl über eine Ethernet als auch eine COM Port Schnittstelle. Da mehrere Schnittstellen zur Verfügung stehen, ist es nötig schon früh in der Planungsphase zu entscheiden, welche Schnittstelle von der Anbindung benutzt werden soll oder ob die Anbindung beide Schnittstellen unterstützen soll. Um das Programm vielen Benutzern zugänglich zu machen, wäre es natürlich am besten, alle Schnittstellen zu unterstützen; für eine erste Implementierung würde aber auch die Umsetzung nur einer Schnittstelle reichen mit Rücksichtnahme auf eine mögliche spätere Implementierung der anderen Schnittstelle. In dem konkreten Fall, der in dieser Arbeit behandelten Anbindung, habe ich mich unter Absprache mit meinen Betreuern für die Implementierung der Anbindung per Ethernet entschieden, mit der Option auf eine mögliche Implementierung der Anbindung per COM Port in der Zukunft.

Diese Entscheidung wurde aus folgenden Gründen getroffen:

- Ein Beispiel aus der Praxis, aus welchem auch ein für das Projekt relevanter Anwendungsfall hervorgeht, benutzt die Ethernet Schnittstelle.
- Eine Anbindung über Ethernet lässt sich in der Regel leicht realisieren, da meist schon ein Netzwerk vorhanden ist und das Gerät *Mythic22* nur noch ins Netzwerk integriert werden muss.
- Die Ethernet Anbindung ermöglicht es, die Daten des Gerätes *Mythic22* an jedes im Netzwerk vertretene Gerät zu senden. Die COM Port Anbindung erfolgt hingegen meist direkt von genau einem Gerät zu einem anderen Gerät.
- Die COM Port Schnittstelle ist veraltet und wird in der Regel nur noch sehr selten benutzt. Diese Schnittstelle wurde Großteils durch USB ersetzt, während Ethernet immer noch eine wichtige und nicht wegzudenkende Schnittstelle ist.

5.3.3. Vorgehensweise

Neben den im Kapitel „5.2 Allgemeine Vorgehensweise“ erwähnten Schritten und der in Kapitel „5.3.2 Wahl der Schnittstelle“ erläuterten Wahl der Schnittstelle, war es nötig, entsprechenden Code zu schreiben, welcher eine Verbindung mit dem Gerät *Mythic22* aufbaut und dessen Ergebnisse entgegennimmt. Um diese Verbindung aufzubauen, ist es nötig, den Port zu kennen auf den das Gerät *Mythic22* seine Daten schickt. Da dieser nicht fest definiert ist und geändert werden kann, muss dieser auch in der Software *Elexis* eingestellt werden können, was über ein Einstellungsfenster geschieht (siehe Kapitel „5.5 Integrieren der Implementierung in Elexis“).

Die durch diesen ersten Teil des Programmes erhaltenen Daten bestanden aus einer langen Kette von Zeichen (einem String). Diese sollten in einem folgenden Teil des Plug-Ins in Objekte umgewandelt werden, damit diese komfortabel weiterverarbeitet werden können. Um dies zu erreichen, war es nötig sowohl die Objekte zu erstellen, in welchen die erhaltenen Daten gespeichert werden, als auch einen Parser zu schreiben, welcher die Daten in Objekte umwandelt.

Um sowohl Objekte als auch Parser zu erstellen, war es vorerst nötig, die Spezifikation der empfangenen Daten genauer zu analysieren. Durch diese Analyse konnte festgestellt werden, welche Daten gesendet werden und wie sich diese Daten aus dem erhaltenen String filtern lassen. Mit diesen Erkenntnissen konnte ein Parser geschrieben werden, der einzelne Daten aus dem empfangenen String filtert und diese in einem Objekt ablegt.

Diese erstellten Objekte vom Typ „*Mythic22Result*“ (siehe Abbildung 2- *Mythic22Result* Objekt) sind die Grundlage für den nächsten Schritt, das Einspeisen der Daten in die Software *Elexis*.

5.3.4. Aufgetretene Probleme und deren Lösungen

Ein aufgetretenes Problem entstand durch eine fehlerhafte Annahme, den Netzwerkverkehr zwischen *Mythic22* und *Elexis* betreffend. Ich habe angenommen, dass das Gerät *Mythic22* erst beginnt Daten zu senden, nachdem sich ein Client verbunden hat, der die Daten entgegennehmen kann. Stattdessen verhält es sich jedoch so, dass das Gerät *Mythic22* auf einen Knopfdruck des Benutzers hin versucht, die Daten an eine zuvor eingestellte Adresse zu senden.

Durch diese fehlerhafte Annahme musste ich einiges an geschriebenem Code wieder verwerfen und diesen neu verfassen. Dieser Code war jedoch nicht unnütz, da ich dadurch erfahren habe, wie man Callback Funktionen in Java realisiert (siehe hierzu Anhang „Callback in Java“).

5.3.5. Ergebnisse

Die Ergebnisse dieses ersten Teils des Plug-Ins sind folgende:

- Eine Klasse „NetListener“, welche die Verbindung zu dem Gerät *Mythic22* handhabt und Daten entgegennimmt (siehe Abbildung 3 - Klasse NetListener und Klasse InputHandler)
- Eine Klasse „InputHandler“, welche die empfangenen Daten verarbeitet und in Objekten ablegt (siehe Abbildung 3 - Klasse NetListener und Klasse InputHandler)
- Ein Objekt vom Typ „Mythic22Result“, welches die Ergebnisse der Blutanalyse des Gerätes *Mythic22* enthält (siehe Abbildung 2- Mythic22Result Objekt)
- Ein Objekt vom Typ „HaematologicalValue“, welches ein Teil von dem Objekt des Typs „Mythic22Result“ ist und ebenfalls Ergebnisse der Blutanalyse enthält (siehe Abbildung 2- Mythic22Result Objekt)

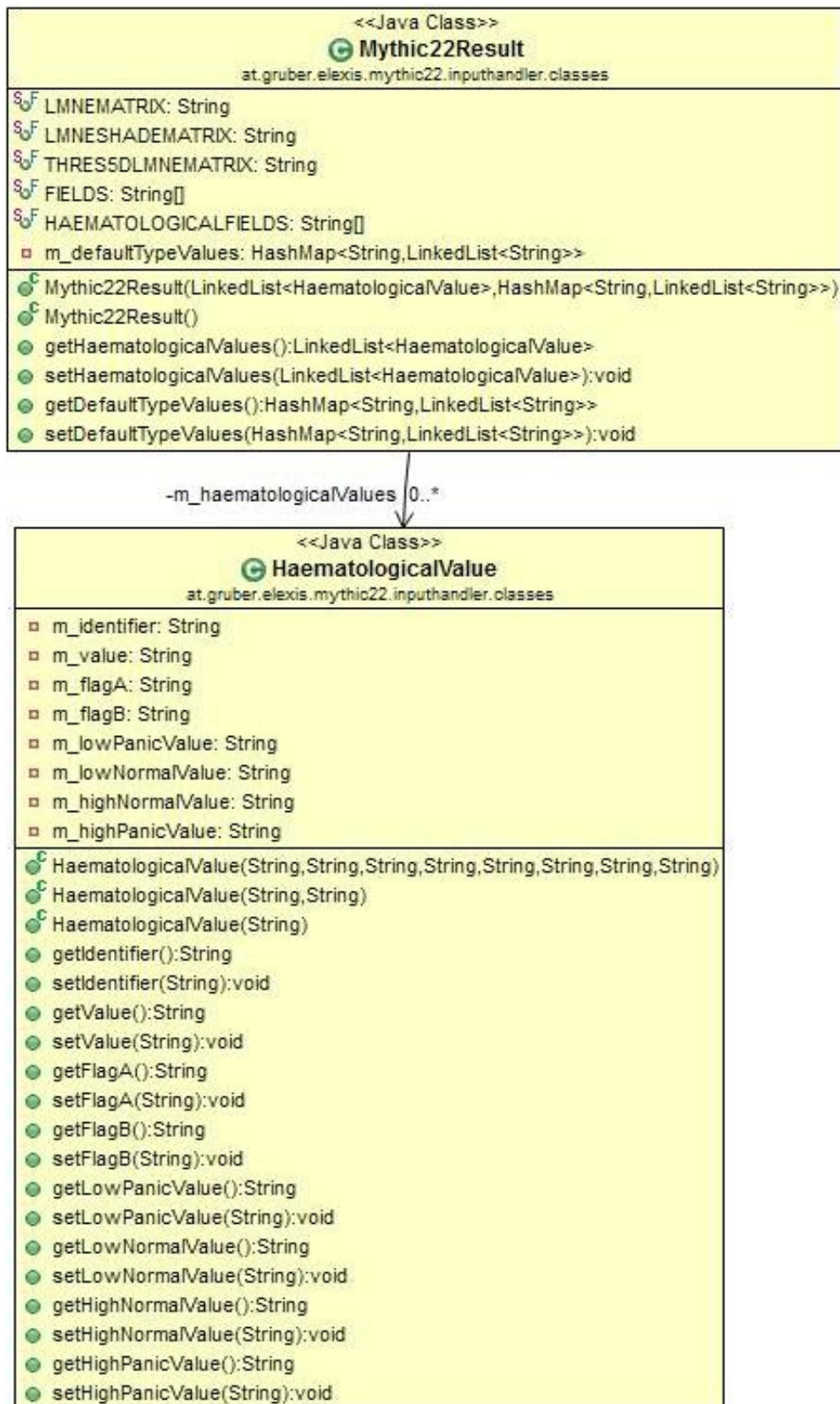


Abbildung 2- Mythic22Result Objekt

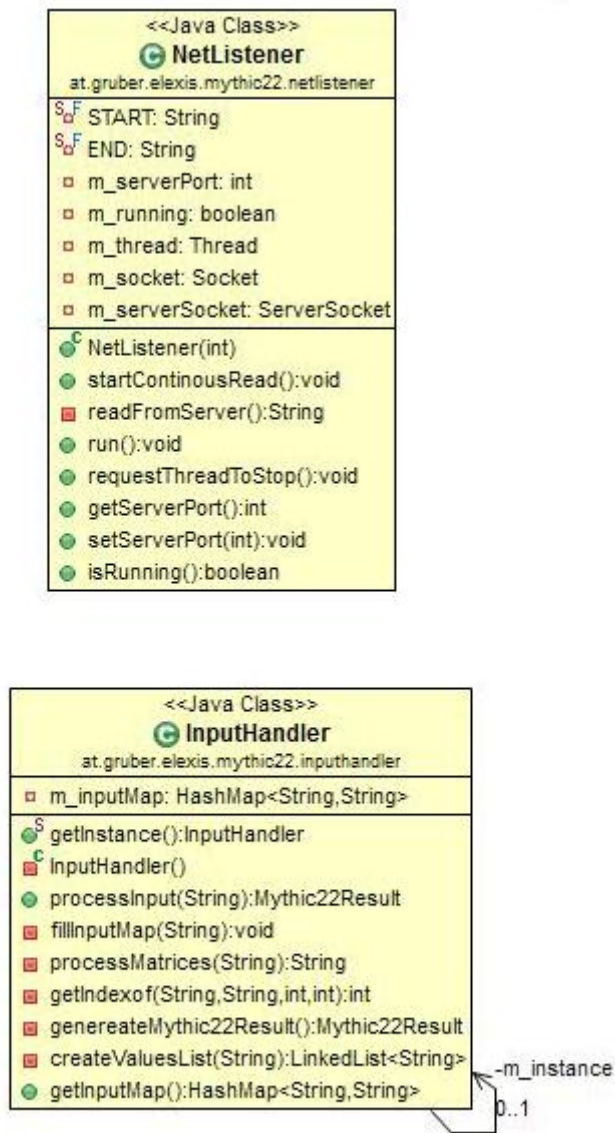


Abbildung 3 - Klasse NetListener und Klasse InputHandler

5.4. Einspeisen der Analysedaten in Elexis

5.4.1. Ausgangsstandpunkt

Das Praxisinformationssystem *Elexis* bot bereits vor Beginn meiner Arbeit die Möglichkeit, Laborwerte, welche unter anderem auch Blutwerte umfassen, zu speichern und zu pflegen. Das Einspeisen der von dem Gerät *Mythic22* erstellten Daten musste von Hand erfolgen. Es war also nötig, die Daten vom Gerät abzulesen und diese in *Elexis* wieder einzugeben und dem richtigen Patienten zuzuordnen.

5.4.2. Vorgehensweise

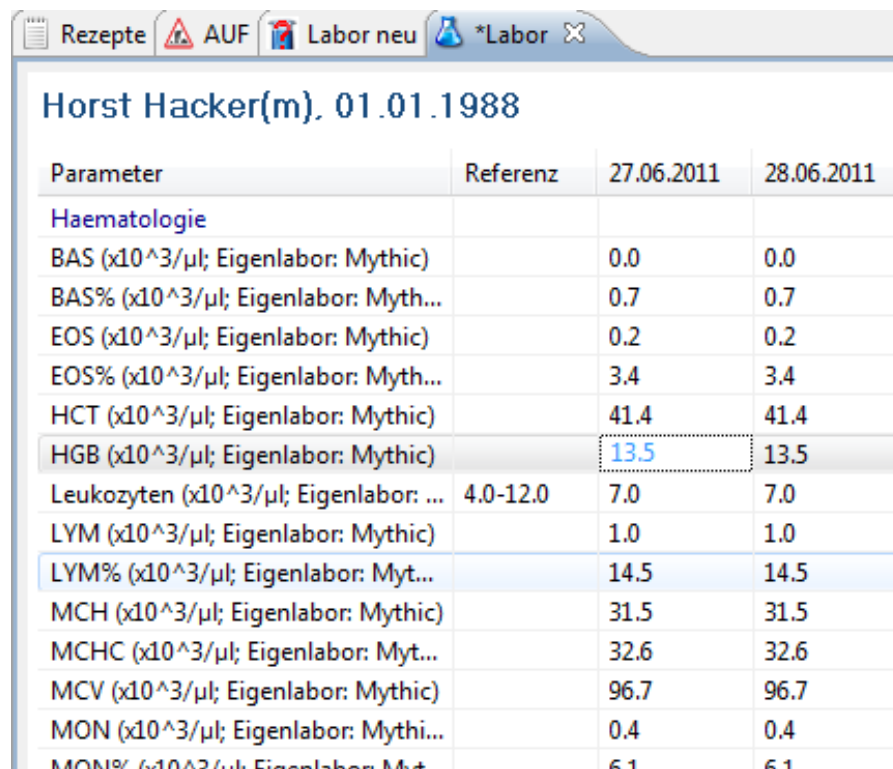
Da durch den in Kapitel „5.3 Anbindung des Analysegerätes Mythic22“ beschriebenen Code bereits ein Objekt besteht, welches alle Ergebnisse der Blutanalyse beinhaltet, ist es nötig, diese Ergebnisse in *Elexis* einzuspielen.

Um dies zu bewerkstelligen, muss aus jedem Blutanalyse-Ergebnis ein passender Laborwert erstellt werden. Dieser Laborwert repräsentiert einen Laboreintrag eines Patienten in *Elexis* (siehe Abbildung 4 - Laborwerte in *Elexis*). Einem Laborwert werden ein Patient, ein Datum und ein Laboritem zugeordnet.

Ein Laboritem definiert den Typ eines Laborwertes näher. So hat als Beispiel ein Blutwert eine Einheit, in der er gemessen wird, eine Gruppe der er zugeordnet ist, einen Referenzwert in dem er sich aufhalten sollte, etc. (siehe Abbildung 5 - Eingabefeld für ein Laboritem).

Damit ein Laborwert erfolgreich einem Laboritem zugeordnet werden kann, wird eine sogenannte „Mapping-Tabelle“ benötigt, welche dem Kürzel eines Laborwertes die Datenbank ID des entsprechenden Laboritems gegenüberstellt (siehe Abbildung 6 - Beispiel einer Mapping Tabelle). Diese Mapping-Tabelle muss einmal vor der Inbetriebnahme des Plug-Ins von Hand angelegt werden.

Nachdem ein neuer Laborwert angelegt wurde mit entsprechenden Zuordnungen zu Patient, Datum und Laboritem, übernimmt *Elexis* selbst die Speicherung dieses Laborwertes in die Datenbank.



Parameter	Referenz	27.06.2011	28.06.2011
Haematologie			
BAS ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		0.0	0.0
BAS% ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		0.7	0.7
EOS ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		0.2	0.2
EOS% ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		3.4	3.4
HCT ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		41.4	41.4
HGB ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		13.5	13.5
Leukozyten ($\times 10^3/\mu\text{l}$; Eigenlabor: ...)	4.0-12.0	7.0	7.0
LYM ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		1.0	1.0
LYM% ($\times 10^3/\mu\text{l}$; Eigenlabor: Myt...		14.5	14.5
MCH ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		31.5	31.5
MCHC ($\times 10^3/\mu\text{l}$; Eigenlabor: Myt...		32.6	32.6
MCV ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythic)		96.7	96.7
MON ($\times 10^3/\mu\text{l}$; Eigenlabor: Mythi...		0.4	0.4
MON% ($\times 10^3/\mu\text{l}$; Eigenlabor: Mst...		6.1	6.1

Abbildung 4 - Laborwerte in Elexis

Laborparameter

Neuen Laborparameter eingeben

Bitte editieren Sie den Parameter und klicken Sie OK.

Eigenlabor: Mythic Labor, ,

Kürzel: WBC Titel: Leukozyten

Typ: ☒ Zahl ☐ Text ☐ Absolut ☐ Formel ☐ Dokument

Referenz M: 4.0-12.0 Referenz W: 4.0-12.0

Einheit: $\times 10^3/\mu\text{l}$

Gruppe: a Haematologie Sequenz-Nr.: L

Export Tags:

OK Cancel

Abbildung 5 - Eingabefeld für ein Laboritem

	A	B	C
1	Kürzel	LaborItemID	
2	WBC	p67ace1c47e35f40209	
3	NEU	jff12bd0289741780016	
4	RBC	m69436a64ca3ec9bf025	
5	HGB	E12a825fcfb39ad26030	
6	HCT	V3f28c759413f5339035	
7	MCV	H50c2fbb1a1dfa83e040	
8	MCH	lb8e6ecffbde1020045	
9	MCHC	q292b9badd2bea2d3050	
10	RDW	z38a67f99653ef2f5055	
11	PLT	a6ba916fab95631eb060	
12	MPV	h267a231f4f4db366065	
13	PCT	x5a375ba33435218b070	
14	PDW	m239ecd82f0527c64075	
15	LYM	E6231c9c4a59a5f5e080	
16	MON	V632465cf43795ab7085	
17	LYM%	H3b072f43885db190090	
18	MON%	lc0556081e3ab8af095	
19	NEU%	qa902cabe35b9a0e0100	
20	EOS	i21cec06be98ea0c30108	
21	BAS	f1b3674cf046e43cd0113	

Abbildung 6 - Beispiel einer Mapping Tabelle

5.4.3. Ergebnisse

Das Ergebnis dieses zweiten Teils des Plug-Ins ist die Klasse „PersistenceHandler“, welche die Blutwerte aus einem Objekt vom Typ „Mythic22Result“ in *Elexis* einspeist und somit persistent macht (siehe Abbildung 7 - Klasse PersistenceHandler)

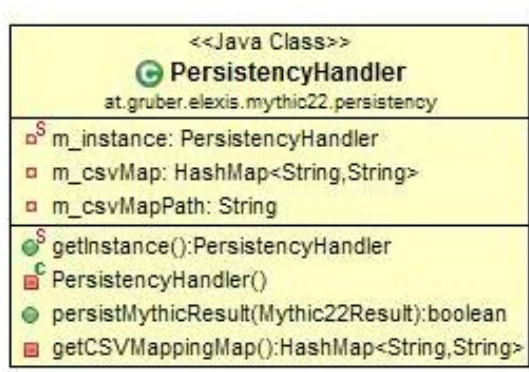


Abbildung 7 - Klasse PersistenceHandler

5.5. Integrieren der Implementierung in Elexis

Da die Grundfunktionen des Plug-Ins bereits fertig gestellt waren, war es noch nötig, eine Möglichkeit für den Endnutzer zu liefern, um auf diese Funktionalitäten zuzugreifen. Da die meisten Vorgänge ohne Eingreifen des Benutzers stattfinden können, war es nur nötig folgendes einzufügen:

- Eine eigene Einstellungsseite in der bereits vorhandenen Einstellungsruhrük „Datenaustausch“. Auf dieser Einstellungsseite lässt sich sowohl der Port angeben, auf den das Gerät *Mythic22* sendet als auch der Pfad der sogenannten „Mapping-Tabelle“ (siehe Abbildung 8 - Einstellungsseite für Mythic22)

- Ein Button in der Toolbar der Laboransicht, welcher einen Hintergrund Thread startet (beziehungsweise stoppt), um ankommende Resultate des Gerätes *Mythic22* zu verarbeiten (siehe Abbildung 9 - Mythic22-Listener Start/Stop Button)

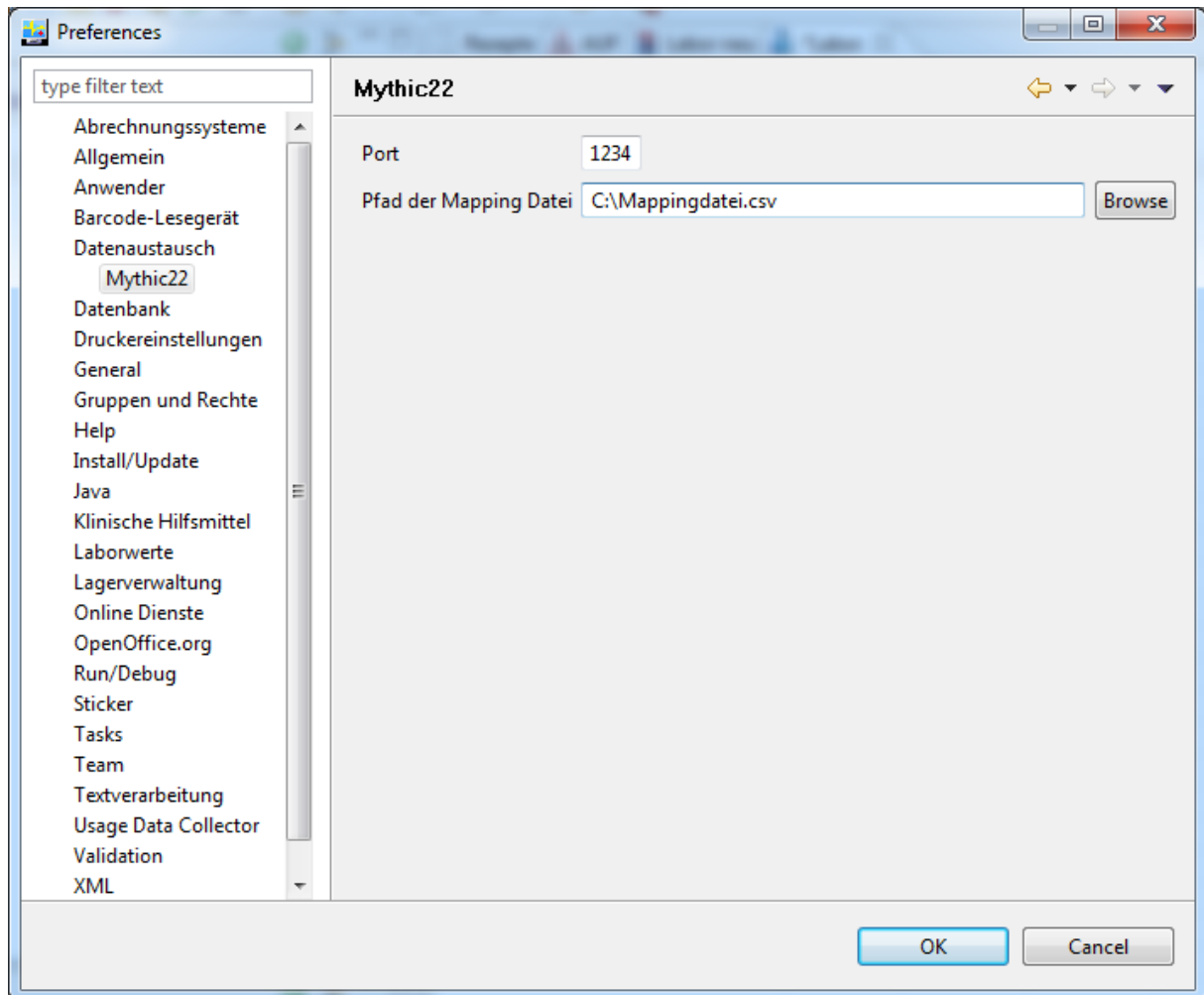


Abbildung 8 - Einstellungsseite für Mythic22

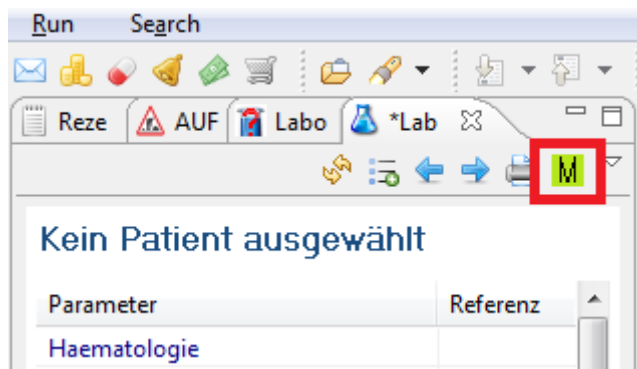


Abbildung 9 - Mythic22-Listener Start/Stopp Button

6. Ergebnisse

6.1.Fazit

Das Programmieren mit der *Eclipse Rich Client Plattform* ist anfangs sehr ungewohnt und vieles ist nicht leicht zu verstehen. Hat man sich aber eingearbeitet, erkennt man den genialen Aufbau des Frameworks und die Stärken, die Plug-In basierte Applikationen mit sich bringen.

Elexis braucht im Moment sicher noch einige Erweiterungen, bevor es für den österreichischen Markt fit ist, aber das ist schließlich nichts, was sich nicht durch das ein oder andere Plug-In lösen lässt. Da *Elexis* in seinem Grundgerüst ein Open Source Programm ist und Plug-Ins in der Regel erschwinglich sind, ist *Elexis* auf jeden Fall eine ernstzunehmende Verwaltungssoftware für Ärzte.

6.2.Anmerkungen

Hier möchte ich noch genauer auf die in Kapitel „5.4.2 Vorgehensweise“ erwähnte Mapping Tabelle eingehen (siehe auch Abbildung 6 - Beispiel einer Mapping Tabelle). Ähnliche Mapping Tabellen sind bereits in anderen Plug-Ins für die Software *Elexis* vertreten und werden meiner Meinung nach sehr viele Probleme verursachen. Das Anlegen dieser Tabelle kann nicht von einem Laien durchgeführt werden und wird auf jeden Fall eine Fachkraft benötigen. Das Anlegen dieser Tabelle wird auch jedes Mal nötig sein, wenn neue Arten von Werten hinzugefügt werden sollen. Es wäre auch durchaus denkbar, dass ein Nutzer diese Mapping Tabelle aus Versehen löscht.

Meiner Meinung nach werden Mapping Tabellen dieser Art einen inakzeptablen Supportaufwand verursachen und sollten vermieden werden.

6.3. Persönlich gezogene Erfahrungen

Bei der Ausarbeitung dieses Projekts konnte ich vor allem in Erfahrung bringen, wie der Ansatz der „Test-Driven“ Entwicklung funktioniert. Dies ist prinzipiell eine Vorgehensweise, die sehr viele Vorteile bringt:

- eine sehr hohe Wartbarkeit des geschriebenen Codes
- Fehlerquellen können bei Änderungen am Code schnell ausfindig gemacht werden
- Code wird vor dem Schreiben besser durchdacht

Es ist meiner Meinung nach ein sehr zu empfehlender Ansatz, welchen ich auch zukünftig weiter zu verfolgen gedenke.

Das Problem, welches ich mit diesem Ansatz sehe ist, dass es in der Regel natürlich mehr Zeit benötigt, um zu einem ersten funktionierenden Code zu gelangen.

Ich habe selbst während der Entwicklung der Anbindung teilweise den „Test-Driven“ Entwicklungsansatz wieder verworfen, um schneller voran zu kommen. Vor allem, wenn man Code umarbeitet, fällt man gerne in alte Muster zurück, denn wenn sich Code als falsch herausstellt, müssen natürlich auch sämtliche Unit Tests geändert werden, worauf man gerne verzichten würde.

6.4. Ausblick

Der von mir im Zuge meiner Bachelorarbeit erstellte Code speichert alle von *Mythic22* eingehenden Daten in einem Objekt zwischen. Es werden jedoch bei weitem nicht alle Daten verwendet, da sie im Moment nicht benötigt werden. Hier ist auf jeden Fall noch Potenzial zu finden. Unter anderem könnte man die Daten von Diagrammen verwenden, um diese in *Elexis* darzustellen.

Ein anderer ausbaufähiger Punkt wäre es, das Gerät *Mythic22* von *Elexis* aus anzusprechen und es so zu steuern.

Ein weiterer verbesserungswürdiger Abschnitt ist auch die in Kapitel „6.2 Anmerkungen“ erwähnte Mapping Tabelle. Hierfür wäre es denkbar Code zu schreiben, welcher den Vorgang des Erstellens einer solchen Mapping Tabelle automatisch durchführt.

Schlussendlich wäre es auch noch denkbar, das Grundgerüst der Implementierung zu verwenden, um andere Geräte anzubinden.

Literatur

McAffer, Jeff; Lemieux, Jean-Michel; Aniszczyk, Chris (2010) „Rich Client Platform – Second Edition“, 1. Auflage, Pearson Education Inc., Boston

Tahchiev, Petar; Leme, Felipe; Massol, Vincent; Gregory, Gary (2009) „JUnit in Action“, 2. Auflage, Manning Publications, Greenwich, Connecticut

Dr. Med. Zürcher, Heini (2009). Elexis: Der Weg zur elektronischen Arztpraxis. Defacto - Fakten und Meinungen der argomed Ärzte AG 2/09, S.2-3.

Zur Rose Ärzte AG (2010). Die elektronische Praxis – mit Elexis .Zur Rose Folium 1/10, S.2-3.

The Eclipse Foundation (2011) „OSGi“ Online im Internet <http://wiki.eclipse.org/OSGi>
(Zugriff am 24.07.2011)

Abbildungsverzeichnis

Abbildung 1 - Ausschnitt gesendeter Analysedaten von Mythic22	9
Abbildung 2- Mythic22Result Objekt	19
Abbildung 3 - Klasse NetListener und Klasse InputHandler	20
Abbildung 4 - Laborwerte in Elexis.....	22
Abbildung 5 - Eingabefeld für ein Laboritem	23
Abbildung 6 - Beispiel einer Mapping Tabelle.....	23
Abbildung 7 - Klasse PersistencyHandler	24
Abbildung 8 - Einstellungsseite für Mythic22	25
Abbildung 9 - Mythic22-Listener Start/Stopp Button	26

Listings

Listing 1 - Callback Interface.....	32
Listing 2 - Beispiel Konstruktor mit Callback Funktion als Parameter	33
Listing 3 - Aufruf der Callback Funktion im Thread	33

7. Anhang

7.1. Callback in Java

Wie schon zuvor erwähnt habe ich durch einen Irrtum Code produziert, welcher wieder verworfen wurde. In diesem Code habe ich Callback-Funktionen benutzt. Diese werden in vielen Programmiersprachen durch Funktionszeiger realisiert. In Java gibt es jedoch keine Funktionszeiger und ich konnte auch keine passende in Java implementierte Alternative dazu finden. Eine Callback-Funktion lässt sich jedoch sehr einfach selbst realisieren.

Ziel der Callback-Funktion war es, die Ergebnisse eines endlos laufenden Threads asynchron zu verarbeiten. Um dies zu realisieren, muss zuerst ein Interface definiert werden mit der Callback Methode. Diese Methode bekommt die Ergebnisse des Thread als Parameter übergeben (siehe Listing 1 - Callback Interface).

Der Konstruktor der Klasse, aus der der Thread hervorgeht, soll nun ein Objekt übergeben bekommen, welches zuvor definiertes Interface implementiert (siehe Listing 2 - Beispiel Konstruktor mit Callback Funktion als Parameter). Die Implementierung dieses Objekts gibt schlussendlich an, was mit dem Ergebnis, welches der Thread asynchron liefert, geschieht.

Zuletzt muss in dem Thread nur noch die Callback-Methode des im Konstruktor übergebenen Objekts aufgerufen werden mit den zu sendenden Daten als Parameter (siehe Listing 3 - Aufruf der Callback Funktion im Thread).

```
public interface Callbackable {  
  
    public void CallbackEventOccured(String mythicOutput);  
  
}
```

Listing 1 - Callback Interface

```
public NetListener(String serverIP, int serverPort, int readInterval,  
Callbackable callbackMethod) {  
    super();  
}
```



```
m_serverIP = serverIP;  
m_serverPort = serverPort;  
m_readInterval = readInterval;  
m_callbackMethod = callbackMethod;  
}
```

Listing 2 - Beispiel Konstruktor mit Callback Funktion als Parameter

```
public void run() {  
  
    while (m_running) {  
  
        String temp = readFromServer();  
        if (temp != null) {  
            m_callbackMethod.CallbackEventOccured(temp);  
        }  
    }  
}
```

Listing 3 - Aufruf der Callback Funktion im Thread