Custom Package Framework

An der

Montessori Schule Huckepack e. V.

Vorgelegt von:

Tendsin Mende

Pohlandstraße 5

01309 Dresden

Abgabetermin:

27.02.2016

Betreuer:

Herr Dipl. Lehrer Gerd Bobe

Montessori Schule Huckepack e. V.

Inhaltsverzeichnis

1 Einführung

1.1 Die Situation

„Schön! Ich habe ein Linux, aber wie bearbeite ich ein Bild?“

An diesem Problem möchte ich mit meiner Arbeit ansetzen. Ich möchte ein- und Umsteigern das Suchen von Programmen erleichtern. Ich sehe drei haupt Probleme für neueinsteiger.

1.2 Grund 1

Auch wenn es die Programme von Windows und MacOSx nicht gibt, so ist die Auswahl an Programmen zu groß. Als Beispiel: Es gibt kein offizielles Adobe: Photoshop für Linux[1]. Es gibt aber Gimp, Krita, MyPaint, und viele mehr. Welches davon soll ich nehmen? Aus meiner erfahrung kann ich sagen: „Krita für malen und Texturbearbeitung und Gimp für Photos“. Aber diese Erfahrung hat der Normalverbraucher nicht. Meine Lösung ist einfach: Ich gebe dem Nutzer nur Gimp und Krita un eine Beschriebung mit meiner Erfahrung.

1.3 Grund 2

Das nächste Problem ist recht trivial. Wie istalliere ich Software?

Die meisten Umsteiger kommen von Windows. Sie erwarten, dass man ins internet geht, den richtigen Installer herunterläd und dann installiert. Auf Linux kann man das Installieren aber eher mit einem Appstore vergleichen. Man wählt sein Packet (Programm) und läd es sich von einem Spiegel-Server herunter. Das Packet weis selbst welche anderen Programme es benötigt und installiert diese nach. Der vorteil dieser Methode ist, dass sich viele Programme einzelne Unterprogramme teilen können. Ausserdem lässt sich das System so einfach mit dem Server abgleichen. Das Aktualiesieren der Programme ist einfacher.

2 Vorüberlegungen

2.1 Ziel

Meine ersten Überlegungen galten zwangsläufig meinem Ziel. Wie sollte mein Programm aussehen um einen möglichst einfachen einstieg in Linux zu gewährleisten?

[1] Vgl. https://helpx.adobe.com/photoshop/system-requirements.html, (14.08.2016, 13:57)

Ich kam zu dem naheliegenden Schluss, dass es das besste sei eine einfache Oberfläche zu konstruieren. Es enstanden die ersten Ideen.

Meine fertige Idee bestand aus einem Ramen, welcher in drei Teile gegliedert war. Der Programmauswahl, ein Dokumentationsbrowser und einem Informationssystem. Die Programmauswahl sollte Icons für verschiedene Programmarten bekommen (zB. „Video und Foto“, „Musik“, „Spiele“). Durch das anklicken der Icons sollte man in eine Feinauswahlen kommen, bis man nach maximal drei Verzeichnissen eine Programmauswahl präsentiert bekommt.

Der Dokumentationsbrowser soll die aufgabe übernehmen, Dokumentation zugänglich zu machen. Dabei denke ich primär an Schul- oder Firmeninterne Dokumentation.

Der Informationsbrowser wiederrum soll über neue Entwicklungen informieren. Es bietet sich an Blog-Webseiten oder Nachrichten an dieser Stelle zu laden.

Ein nicht zu unterschätzender Teil ist die Administartion eines solchen Programmes. Ich weis aus erfahrung meines ersten größeren Programmes[2], dass man ein gutes Konfigurations-System braucht. Bei CPF (kurz für: Custom Packaging Framework) würde ausserdem noch ein weiteres Problem hinzu kommen. Da es sich um die Installation von Programmen handelt, muss die Sicherheit des System gewährleistet sein. Das heist es muss genau geregelt werden wie man installiert und welche Programme welchen zugriff auf Dateien haben.

2.2 Konfiguration

2.2.1 Programm eigene Konfiguration

Die Programm eigene Konfiguration umschliest die Art, wie CPF reagieren soll. Dazu sollte gehören:

- Umgang mit der Datenbank
  - Speicherung von Daten
  - Abrufen von Daten
- Umgang mit dem Internet
  - Laden der Websiten in Documentation und Info-System
  - Herrunterladen von Programmen
  - Abgleichen der Datenbank mit online Repositorium
- Verhalten wärend der Installation

[2]siehe: https://github.com/SiebenCorgie/Beta-Launcher , 18.08.2016, 18:30

  - Automatisierungsgrad während der Installation
  - Passwortabfrage (welcher Benutzer fragt?)
- Massenaufträge
  - Grafische Rückmeldung

Die verschiedenen Optionen werden in verscheidenen Registerkarten eines Konfigurationsfensters geändert. Das ist erfahrungsgemäß die übersichtlichste Art.

2.2.2 Datenbank

Die Datenbank soll aus einfachen Datensätzen mit verscheidenen Daten beruhen. zu diesen Daten soll gehören:

- Name des Programms
- Entwicklerbeschreibung
- Eigene Beschreibung
- Programm Type (Grafik, Video, Spiel, etc.)
- Programm Entwickler
- Programm Website
- Pfad zu Screenshot
- Pfad zu eventuellen Programmdateien

2.2.3 Datenbank management

Die Datenbank soll ausserdem erweiterbar sein. Dazu muss auch ein eignes Tool geschrieben werden. Die erweiterung wird aus einer Eingabemaske bestehen sowie der option die lokale Datenbank in das Repositorium hochzuladen.

2.3 Installation

2.3.1 Visualisierung

Das Installieren von Software soll mit minimalem visuellen eindrücken auskommen. Dabei möchte ich nur den Fortschritt, und den Namen der Aktion anzeigen. Die Werte sollen im unteren Teil des Programms angezeigt werden.

2.3.2 Installationsablauf

Die installation soll wie folgt ablaufen:

1. Programm Auswahl
2. Installations-Start
3. Passwortabfrage (benutzer wird aus Konfigurationsdateien gelesen)
4. Bestätigung
5. Instalation
6. Erfolgs/Misserfolgs-Nachricht

Bei einem Misserfolg soll nach möglichkeit der Fehler angezeigt werden. Insofern benötigt soll man auch den Lockbuch-Eintrag sehen können.

2.3.3 Massenaufträge

Das Programm soll Massenaufträge abarbeiten können. Dies soll vorallem Administratoren die Arbeit ersparen. Bei einem Massenauftrag werden viele Programme mit einem mal installiert.

2.3.4 Programmauswahl

Die Programme sollen wie in 2.1 beschrieben durch fein Filtern der Optionen ausgewählt werden. Ich werden versuchen die beschreibungen möglichst neutral zuhalten. Damit soll die entscheidung des Nutzers möglicht neutral bleiben, um zum gewünschten Programm zu gelangen.

2.4 Web-Integration

Die Webintegration soll über PyWebKit[3] erfolgen. Das ist eine von Apple entwickelte Umgebung, um relativ einfach Browserfähigkeiten in Gtk-Programme einzubauen. Ich werde es verwenden um einfache Websiten anzuzeigen. Die Lizensierung von WebKit[4] ermöglicht es mir das Programm kommerziell oder nichtkommerziell zu entwickeln und zu veröffentlichen.

[3]siehe: https://webkit.org/ , 14.08.2016, 20:18
[4]https://webkit.org/licensing-webkit/ , 14.08.2016 , 20:21

2.5 Programmiersprache

Als Programmiersprache nehme ich Python.Es gibt verschiedene Gründe dafür.

Der Erste ist, dass ich schon seit Sommer 2015 Programme mit Python schreibe. Ich kann es realtiv „flüssig“ schreiben. Diese Vorkenntnis ist nötig um in der gegebenen Zeit ein so komplexes Programm zuschreiben.

Der zweite Grund ist, dass Python eine interpretiere[5] Sprache ist. Der Vorteil ist, dass ein Python-Programm bei einem Programmfehler nicht abstürzt. Es führt in meinem Fall den Befehl nicht aus und kehrt in die Warte-Position zurück[6]. Dadurch ist das potenzielle Installieren fehlerhafter Software, oder frustrierende abstürze des Programms geringer. Ein Nachteil ist, dass man Programmfehler erst bei der Ausführung findet. Das Testen wird dadurch aufwendiger.

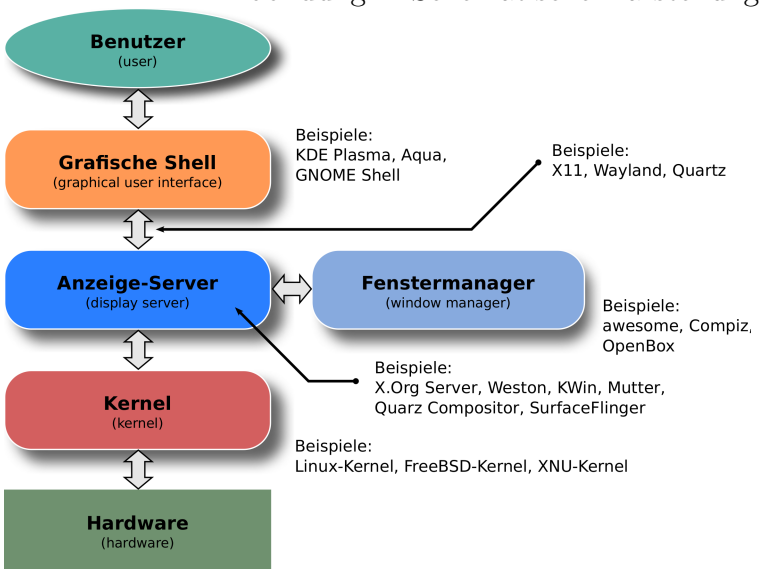
Der Dritte Grund ist Pythons anbindung an das Gtk+ Projekt. Wie ich im nächsten abschnitt beschreiben werde nutze ich das GTK+ um meine grafische Umsetzung zu realisieren. Bei einem so grundlegenden Element ist es wichtig, dass die Anbindung an die Programmiersprache gut ist. Dies ist bei Python glücklicherweise gegeben.

2.6 Interface-Toolkit

Das Interface-Toolkit[7] ist auf Linux zu Verständigung zwischen dem Display-Server (z.B. X11 oder Wayland) und dem Benutzer zuständig. Es stellt verschiedene Elemente bereit, die zusammen das fertige Programm-Interface ergeben (siehe Abb.1).

[5]Siehe: https://de.wikipedia.org/wiki/Interpreter
[6]Die Charackteristik eines Gtk+ Interfaces wird Später noch geklährt.
[7]kurze Erklährung: https://de.wikipedia.org/wiki/Toolkit

Abbildung 1: *Schematische Darstellung der Beziehungen*



Quelle: https://en.wikipedia.org/wiki/File:Schema_of_the_layers_of_the_graphical_user_interface.
, Shmuel Csaba Otto Traian, 29.10.2013

Ich habe mich für Gtk+[8] entscheiden. Der wichtigste Grund wurde schon in „2.5 Programmiersprache“ angeführt. Gtk+ hat eine exzellente Pythonanbindung (ab jetzt API , kurz für Application-Programming-Interface). Allerdings macht sich Gtk+ durch weitere Vorteile prädestiniert für den Einsatz in meinem Programm.

Ein weiterer großer Vorteil ist die Lizensierung. Gtk+ ist unter LGPL[9] Lizensiert[10]. Diese erlaubt es mir Gtk ohne Restriktionen einzusetzen. Die LGPL Lizens ist eine der am weitesten verbreiteten Lizensen in der OpenSource-Welt. Der erfolg durch die Lizens und die Qualität lässt sich auf Linux leicht erkennen. Es gibt sehr viele Programme die mit dem Gtk erstellt wurden.Desweiteren gibt es sogar eine ganze Desktop-Umgebung die auf Gtk basiert. Genannt „Gnome“[11], welche ich selbst benutze. Das Gtk Projekt ist eines der ältesten[12] und erfolgreichsten Linux-Projekte im grafischen Sekment. Mit dem erfolg kommt eine weite Verbreitung im Linux-Anwenderbereich. Dre kommt mir insofern zugute, als das es auf fast allen Distriubutionen die benötigten Packete für mein Programm gibt.

Der letzte Grund für das Gtk+ ist meine Erfahrung. Wie in 2.5 erwähnt habe ich schon einige Erfahrung auf dem Gebiet von Python. Dazu gehört es auch, dass ich ab Ostern 2016 ein Programm[13] zum Organisieren und Installieren der UnrealEngine 4 auf Linux geschrieben habe. Dieses war auch mit Gtk erstellt worden. Ich habe wärend des Programmierens viel Erfahrung gesammelt. Damals habe ich mehrer Toolkits miteinander

[8]Englisch: http://www.gtk.org/ ,18.08.2016, 21:25
[9]Englisch: http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html , 18.08.2016 , 21:32
[10]Englisch: http://www.gtk.org/ , 18.08.2016, 21:32
[11]Englische Referenz: https://www.gnome.org/
[12]https://www.gimp.org/about/ancient_history.html , 08.18.2016 , 21:47
[13]siehe: https://github.com/SiebenCorgie/Beta-Launcher

verglichen (z.B. Qt und wxWidgets). Am Ende habe ich mich für das Gtk entscheiden. Das Ergebniss ist ein gefestigtes Anfängerwissen über ein Toolkit welches ich mir bewusst gesucht habe.

2.7 Unterstützung

=> Unterstützung von .deb (debian) und .tar.xz (arch-based) weil RPM keine erfahrung etc.

Abbildungsverzeichnis

3 Anmerkung zum Literaturverzeichnis

Das Literaturverzeichnis kennzeichnet Quellen mit englischen Inhalt mit „EN“.

Literatur

[1] EN: WebKit Hauptseite: https://webkit.org/

[2] EN: WebKit Lizens: https: https://webkit.org/licensing-webkit/ , gesamtes Dokument

[3] EN: Gtk Hauptseite: http://www.gtk.org/

[4] EN. LGPL Lizens Webseite: http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[5] EN: Kurzfassung des Gtk: https://www.gimp.org/about/ancient_history.html

[6] EN: Beta-Launcher, Mein erstes Gtk-Projekt: https://github.com/SiebenCorgie/Beta-Launcher

[7] EN: Unreal-Engine-4 Hauptseite: https://www.unrealengine.com/what-is-unreal-engine-4