Diplomarbeit

Thema: Moderne Ansätze zur Oberflächengestaltung für hardwarenahe Programmierung

Inhaltsverzeichnis

1. Begriffsklärungen

1.1. Klärung Begriff „modern“

[Erklärung]

1.2. Klärung Begriff „hardwarenah“

[Erklärung]

→ Nicht auf Anforderungen für Bewertungskriterien eingehen. Lediglich allgemein erklären, worum es sich handelt (limitierte Rechen- und Energieresourcen, sensitives Zeitverhalten usw.)

1.3. Klärung Begriff „Prototyp“

- Unterschied Prototyp + Software

2. Vorbetrachtungen

2.1. Historische Einordnung der Entwicklung von Benutzeroberflächen

[Kurze Historische Einordnung der Entwicklung von Benutzeroberflächen] → Diesen Teil mit zuletzt schreiben, da man ihn gut strecken und stauchen kann, evtl. komplett weglassen

2.2. Möglichkeiten zur Gestaltung von Benutzeroberflächen

[Hinweis: Möglichkeiten zur Gestaltung von Benutzeroberflächen zu allgemein formuliert]

[Erklärung von verschiedenen Konzepten]

2.2.1. MVC

2.2.2. MVP

2.2.3. MVVM

2.3. Anforderungen für hardwarenahe Programmierung aufstellen

[Garbage Collection sorgt für unvorhergesehenes Zeitverhalten → nicht benutzen]

[Möglichst selten Speicher-Allokalisierung durchführen]

[Evtl. Ausflug in Fragmentierung von heaps geben → ganz zum schluss schreiben, evtl. weglassen]

3. Vergleich von populären Möglichkeiten zur Benutzeroberflächengestaltung

3.1. Bewertungskriterien aufstellen

[Allgemeine Anforderungen festlegen:]

→ Erweiterbarkeit mit neuen Elementtypen

→ Wiederverwendbarkeit von erstellten Views in anderen Views

→ Widerspruch zwischen Performance zur Laufzeit und Entwicklerkomfort → Lösen über interaktiven Modus mgl.?

→ Keinen Quellcode für View-Elemente schreiben (die keine weitere Funktionalität beinhalten)

→ Einbinden von bestehenden Objekten und Variablen möglich

[Anforderungen für hardwarenahe Programmierung festlegen:]

- Overhead durch dynamische Allokalisierung von Memory (besonders in Bezug auf Objektorientierung)

- Keine VM (daher kein java)

- Impedance Mismatch: Graphikkartenbibliothek arbeitet als Statemachine → daher Objektstruktur möblichst flach halten [Hinweis: Begriff so nicht in der Literatur gefunden, aber ich halte ihn für treffend für diesen Punkt]

- Kein Garbage-Collector

- Langlebigkeit des Quellcodes

3.2. Technologien ermitteln

[Zu jeder Möglichkeit min. einen prominenten Vertreter auswählen, Technologien kurz vorstellen]

3.3. Vergleichen

[Punkte vergeben, jede Punktevergabe kurz begründen]

3.4. Fazit ziehen

[Stärken von bestimmten Ansätzen betonen, Schwachpunkte für weiteres Vorgehen erwähnen]

4. Auszeichnungssprache/ Markup language

[Hinweis: Begriff Datendefinitionssprache ist nicht korrekt: „*Datendefinitionssprache, Data Description Language (DDL), Data Definition Language (DDL);* eine Sprache, die zur Beschreibung der Struktur einer Datenbank aus der Sicht des konzeptionellen Datenmodells,

externen Datenmodells oder internen Datenmodells dient. Zu einem Datenbankmanagementsystem (DBMS) gehört stets eine Datenbeschreibungssprache.„]

4.1. Funktion von Markup Languages

[Allgemeine Funktion von Markup Languages erklären]

4.2. Vorstellung von ausgewählten Markup Languages

4.2.1. XML

4.2.2. XAML

4.2.3. JSON

4.2.4. YAML

4.3. Problematik der Typisierung

[Kurz auf die Problematik der Typisierung eingehen → Implizite Typvergabe bei embedded Programming häufig nicht gewollt]

4.4. Begründete Auswahl

5. Prototypische Implementierung

5.1. Rahmenbedingungen

5.1.1. Interaktiver Modus

5.1.2. Portabilität

5.2. Vorstellung des Konzepts

[Erklärung der Markup Language]

[Erklärung der Kompilierung des YAML Codes]

[Erklärung interaktiver Modus]

5.3. Gewährleistung der Rahmenbedingungen

[Erklärung, dass Rahmenbedingungen erfüllt worden sind]

6. Praxisnahe Anwendungsfälle

[Ausflug geben in bestimmte Bereiche von embedded devices]

- Smart Home Anwendungen

- Automaten jeglicher Art (Fahrkartenautomaten, Getränkeautomaten, …)

- Haushaltsgeräte

- Bürogeräte (Drucker, ...)

7. Abschließende Bemerkungen und mögliche Zukunftsausblicke des Projektes