Diplomarbeit

Thema: Moderne Ansätze zur Oberflächengestaltung für hardwarenahe Programmierung

Inhaltsverzeichnis

Seitenüberschlag:

1	1
2.1	2
2.2	3
3	3
4.1	2
4.2	3
4.3	3
4.4	1
5.1	1
5.2	3
5.3	1
5.4	1
6.1	3
6.2	2
6.3	5
6.4	25
6.5	1
7	1
8	1

Summe: 62

1. Begriffsklärungen

Klärung Begriff „modern“

[Erklärung]
→ sehr agil entwickeln; ständige Anforderungsänderungen; Entscheidungen möglichst lange hinauszögern

1 Klärung Begriff „hardwarenah“

[Erklärung]

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

2 Klärung Begriff „Prototyp“

[Erklärung]

- Unterschied Prototyp + Software

2. Vorbetrachtungen zu Möglichkeiten der Programmierung von Benutzeroberflächen

2.1. Historische Entwicklung von Benutzeroberflächen

[Kurze Historische Einordnung der Entwicklung von Benutzeroberflächen; nicht weiter auf einzelne DP eingehen, da diese im kommenden Kapitel erklärt werden] → Diesen Teil mit zuletzt schreiben, da man ihn gut strecken und stauchen kann, evtl. komplett weglassen

[1] [2] [3]

2.2. Konzepte für die Programmierung von Benutzeroberflächen

[Erklärung von verschiedenen Konzepten]

2.2.1. MVC

2.2.2. MVP

2.2.3. MVVM

[1]

3. Vorbetrachtungen zu Einschränkungen durch hardwarenahe Programmierung

Rechenleistung

→ Betrachtungen zum Prozessor

→ häufig 8- oder 16-bit, dieser Fall wird aber nicht betrachtet

→ Abgrenzung zu Microcontrollern an dieser Stelle

Memory

→ Eingeschränkt

Entwicklungskosten vs. Produktionskosten

→ Trade-Off zwischen Entwicklungskosten und Produktionskosten erklären

→ Erklären, dass über diese Ansätze lediglich Entwicklungskosten gesenkt werden sollen und dadurch minimal-geringe Mehrkosten in der Produktion in Kauf genommen werden → hängt von Anzahl an ausgerollten Geräten ab

Lebensdauer

→ ...

Ausfallsicherheit

→ ...

[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]

[4, S. 5]

4. Bewertung von Technologien zur Eignung für hardwarenahe Benutzeroberflächenerstellung

4.1. Bewertungskriterien

[Allgemeine Anforderungen:]

→ 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

→ Möglichst viele Programmierparadigmen unterstützen ↔ keine Framework-Strukturen vorgeben, um dynamische Programmierung zu ermöglichen

- Langlebigkeit des Quellcodes?

- Vendor Locking [5]

[Anforderungen für hardwarenahe Programmierung:]

- Overhead durch dynamische Allokalisierung von Memory (besonders in Bezug auf Objektorientierung) → Objektstruktur möglichst flach halten

- Keine VM (daher kein java)

- Konflikt zwischen Objektorientierung und Graphikkartenbibliothek, welche als Statemachine arbeitet

- Kein Garbage-Collector

4.2. Ausgewählte Technologien

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

4.3. Vergleich

[Punkte vergeben, jede Punktevergabe kurz begründen]

4.4. Fazit

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

5. Auszeichnungssprachen

5.1. Funktion von Auszeichnungssprachen

[Allgemeine Funktion von Auszeichnungssprachen erklären]

[Einsatzmöglichkeiten von Auszeichnungssprachen + Konsequenzen, Bezug nehmen auf Programmierung von graphischen Benutzeroberflächen]

5.2. Ausgewählte Auszeichnungssprachen

5.2.1. XML

5.2.2. XAML

5.2.3. JSON

5.2.4. YAML

5.3. Problematik der Typisierung

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

5.4. Begründete Auswahl

6. Prototypische Entwicklung

6.1. Anforderungen an die Software

6.1.1. Funktionale Anforderungen

- Implementation eines Taschenrechners?

- Login-Page?

- Kontaktliste?

6.1.2. Nicht-Funktionale Anforderungen

6.1.3. Rahmenbedingungen

Portabilität soll gewährleistet bleiben

6.2. Grobentwurf

Grober Softwareentwurf (Schritte zum kompilieren eines Projektes)

6.3. Feinentwurf

Vorstellung der Konzepte:

- Erklärung der Markup Language Syntax

- Resource-Provider Konzept

- Objekterstellungs-Injektion

- Referenzierung von Objekten in der Markup Language von außerhalb

- Typüberladung

6.4. Implementation

6.4.1. Objektorientierung in C

6.4.2. Ausgewählte Basisklassen

6.4.3. Implementation der jeweiligen Konzepte mit ausgewählten Codebeispielen (Kapitel aufsplitten je Konzept)

6.5. Validierung

[Überprüfen, ob Rahmenbedingungen erfüllt worden sind]

7. Praxisnahe Anwendungsfälle

[Ausflug geben in bestimmte Bereiche von embedded devices]

- Smart Home Anwendungen

- Automaten jeglicher Art (Fahrkartenautomaten, Getränkeautomaten, …)

- Haushaltsgeräte

- Bürogeräte (Drucker, ...)

8. Abschließende Bemerkungen und mögliche Zukunftsausblicke des Projektes

9. Literaturverzeichnis

[1] „GUI Architectures“, *martinfowler.com*. [Online]. Verfügbar unter: https://martinfowler.com/eaaDev/uiArchs.html. [Zugegriffen: 14-Sep-2018].

[2] „Organizing Presentation Logic“, *martinfowler.com*. [Online]. Verfügbar unter: https://martinfowler.com/eaaDev/OrganizingPresentations.html. [Zugegriffen: 14-Sep-2018].

[3] „Understanding Basics of UI Design Pattern MVC, MVP and MVVM - CodeProject“. [Online]. Verfügbar unter: https://www.codeproject.com/Articles/228214/Understanding-Basics-of-UI-Design-Pattern-MVC-MVP. [Zugegriffen: 14-Sep-2018].

[4] M. Barr, *Programming Embedded Systems in C and C++*. O’Reilly Media, Inc., 1999.

[5] „Trends in der modernen Software-Entwicklung“. [Online]. Verfügbar unter: https://www.microsoft.com/germany/techwiese/know-how/trends-in-der-modernen-software-entwicklung.aspx. [Zugegriffen: 14-Sep-2018].