Kapitel 1

Fragestellungen bei der Entwicklung von GUIs

Was ist eine GUI?

- ***G***raphical ***U***ser ***I***nterface
- →Grafische Schnittstelle zwischen Mensch und technischem System.
- Der Mensch kommuniziert über grafische Oberflächen mit technischen Systemen.

Wo kommen GUIs vor?
Überall!

- Bordcomputer im Auto,
- Bankautomat,
- Kassenautomat im Parkhaus,
- „Klassische“ GUIs am PC,
- Navigationsgeräte,
- Smartphone, Tablet, Wearables (sehr kleine Anzeigen ...)

Die Kommunikation Mensch ←→Maschine erfolgt dabei i.d.R. bidirektional. Das bedeutet:

- Der Mensch kommuniziert über Eingaben zum System hin.
- Das System kommuniziert über Text-, Sprach- oder sonstige Ausgaben zum Menschen zurück.

Was bedeutet der Begriff „GUI“ genau?
Graphical ***U***ser ***I***nterface

Auf deutsch: Grafische Benutzerschnittstelle. Hiermit sind grafische Benutzungsoberflächen gemeint, wie wir sie von PC, Smartphone oder TabletPC kennen.

Was bedeutet der Begriff GUI-Design?
Entwurf des statischen Verhaltens und des Dialogverhaltens der GUI.

- Statisches Verhalten: Layout und Gestaltung der Fenster einer GUI.
- Dialogverhalten: Wann wird welches Fenster aufgeklappt? Wann wird eine Nachricht oder Zwischenfrage eingeblendet? Das Dialogverhalten einer Oberfläche modelliert den gewünschten Arbeitsablauf des Benutzers.

Ist das Dialogverhalten bei allen GUIs gleichartig? ***Nein!!!*** Ein Textprogramm oder eine Entwicklungsumgebung „lebt“ beispielsweise von Texteingaben – dafür ist diese Art von GUIs ja auch vorgesehen. Eine Smartphone-App dagegen wird i.d.R. nach dem Paradigma „Eingaben sind böse!“ entwickelt. Hier muss ein Dialog in wenigen Taps oder Klicks erledigt sein.

Kapitel 2

Entstehungsgeschichte von GUIs

- Lochkarten:
  - Ausgaben des Rechners auf Endlospapier
  - Fehlersuche in Programmen →Austausch von Lochkarten
- Terminalprogramme (VT 100, 80 Zeichen 24 Zeilen am Bildschirm darstellbar):
  - ermöglichten Befehlseingaben (z.B. ls, rm, ...)
  - Es gab noch keine Menüstrukturen
- Blockfenster im Textmodus (Unter MS-DOS: Word, Wordstar; Yast bei Linux) →Jedes Zeichen am Bildschirm adressierbar.
  - Erste Menüstrukturen
  - Menüpunkte waren über Tastatur erreichbar / Shortcuts
- Pixelgrafik: Jedes Pixel am Bildschirm adressierbar →Erste marktreife GUI-Systeme (Graphical User Interface)
  - MS-Windows
  - X Window unter Unix
  - Sun View
  - OSF/Motif
  - Apple / MAC
- Anfang von Weboberflächen
  - JS / PHP
  - JSP / JSF

  - ASP.NET

- Sprachunabhängige Widget-Sets: .NET
- Plattformunabhängige Widget-Sets:
  - Java: AWT/Swing, SWT/JFace/RCP/RAP
  - C++: Qt
  - Python
  - Fortgeschrittene Frameworks für Weboberflächen: GWT, RAP, React, Angular ....
- Smartphones/Tablets
  - Blackberry (nicht mehr unterstützt)
  - iPhone / iPad
  - Android
  - Windows 10 / .NET (für Smartphones nicht mehr unterstützt)
  - Tizen (mittlerweile Samsung)

Kapitel 3

GUIs aus Anwendersicht

3.1 Beschreibung von GUIs

Der Nutzer interessiert sich nicht dafür, wie eine GUI programmiert ist. Für ihn/sie ist eher von Bedeutung:

- Erkenne ich alle Elemente auf der Oberfläche?
  - Kontraste stark genug?
  - Oberflächenbereiche deutlich genug von einander abgesetzt?
  - Beschriftungen lesbar?
- Sind die Abläufe für mich in meinem (Arbeits-)Umfeld logisch und folgerichtig?
  - Lernbarkeit (explizit und implizit)
  - Selbsterklärende Oberfläche
  - Erwartungskonformität
- Wie gut werden meine Eingabefehler abgefangen/verhindert?
- Wie gefährlich ist es, wenn ich einen Eingabefehler mache?
- Ist die Oberfläche konsistent in Aussehen und Verhalten?
- Wie schnell kann ich mit der GUI arbeiten?

Aus diesen und ähnlichen Fragestellungen heraus haben sich einige Begriffe entwickelt, die GUIs aus Nutzersicht beschreiben:

Konsistenz

- Verhaltenskonsistenz: Gleiche Aktionen sollen mit gleichen Vorgehensweisen und gleichen Rückmeldungen versehen sein. (z. B. Verhalten beim Löschen)
- Beschriftungen, Rückmeldungen, usw. in der gleichen Sprache
- Farbkonsistenz, Konsistenz bei Beschriftungen

Unterschiedliche Navigationsarten

- Haptisch
  - Tastatur, Maus, Penpointer, Touchscreen,...
  - Tastatur mit Shortcuts/Mnemonics: für den geübten Nutzer (schnell)
  - Direkte Manipulation mit Maus/Penpointer/Touchscreen: für den weniger geübten Nutzer (einfacher).
  - Auf Smartphones/Tablets sind die Grenzen zwischen Tastatureingabe und direkter Manipulation fließend.
- Sprachlich
  - Dort, wo haptische Eingabe nicht möglich oder weniger effizient ist
  - (z. B. Navigationsgeräte)

SDI und MDI-Oberflächen

- ***S***ingle ***D***ocument ***I***terface: Ein Hauptfenster enthält nur ein Dokument zu einer Zeit. Beispiel: Microsoft Office. Hier wird beim Öffnen eines weiteren Dokumentes immer ein neues Hauptfenster mit voller Dekoration aufgeklappt.
- ***M***ultiple ***D***ocument ***I***terface: Hier kann ein Hauptfenster mehrere Dokumente zu einer Zeit enthalten. Die einzelnen Dokumente werden entweder in unterschiedlichen Tabs (Reitern) oder in ChildWindows (Sekundärfenstern) gehalten.

Konfigurierbarkeit/Anpassbarkeit

- ... durch den Nutzer: Der Nutzer/Kunde möchte in einem bestimmten Umfang das Aussehen und Verhalten der Software eigenen Bedürfnissen anpassen können
  - Shortcuts vergeben
  - Toolbar anpassen
  - Klingeltöne vergeben
- ... durch den Administrator: Einstellung von
  - Anfängermodus
  - Expertenmodus
- ... durch den Entwickler: Anpassung der SW an
  - verschiedene Kundenkreise
  - verschiedene Landessprachen.

Adaptive Oberflächen

Das Gegenstück zu Anpassbarkeit/Konfigurierbarkeit: Die Oberfläche stellt sich selbstständig auf das Verhalten des Nutzers ein durch z. B.

- Ausblenden selten genutzter Menüs
- Einblenden von Hilfeangeboten
- Anzeige der letzten Dateien im Zugriff

3.2 Qualitätsanforderungen an GUIs

Es gibt verschiedene Standards, mit denen Anforderungen an die Qualität von GUIs aus Nutzersicht gestellt werden:

- ISO-Standards, z.B. DIN EN ISO 9241. Er beschreibt einzuhaltende Grundprinzipien bei der Entwicklung von Oberflächen.
- StyleGuides: Hierbei handelt es sich um Standards, die von Firmen oder Firmenkonsortien für eine bestimmte Anwendergruppe oder ein bestimmtes Widget-Set aufgesetzt werden. Beispiele:
  - Windows User Experience Interaction Guidelines von Microsoft (siehe [7])
  - GNOMEs Richtlinien für Benutzeroberflächen (dt. Übersetzung, siehe [11])
  - Android Design (siehe [9]).
- StyleGuides haben häufig neben einer ausführlichen Beschreibung der Designrichtlinien eine Checkliste, die bei der Entwicklung von Oberflächen „abgehakt“ werden kann.

Kapitel 4

GUIs aus Entwicklersicht

4.1 Eventsteuerung in Benutzungsoberflächen

Eine Benutzungsoberfläche reagiert auf Benutzereingaben. Diese Eingaben können durch z. B. Tastatureingabe, Mausklick oder auch durch Spracheingabe erfolgen. Sie können auch von Systemkomponenten wie der Systemuhr kommen. So lange keine Eingabe erfolgt, tut die Oberfläche – ***nichts***. Dieses Verhalten wird meist Schleifenkonstruktionen implementiert, die Sie auch schon in der Vorlesung „Betriebssysteme“ kennengelernt haben. Sobald eine Eingabe irgendwelcher Art geschieht, wird ein ***Event-Objekt*** erzeugt, welches innerhalb der ***EventLoop*** (Ereignisschleife) ausgewertet wird. Wir unterscheiden folgende Varianten von Eventsteuerung:

- wartende Eventsteuerung,
- pollende Eventsteuerung,
- objektorientierte Eventsteuerung

Wartende Eventsteuerung

Im Schleifenrumpf wird immer wieder eine Abfragefunktion oder Abfragemethode aufgerufen, welche eine Rückmeldung über die Art des eingetretenen Ereignisses zurückliefert. Die Abfragefunktion (hier *warteAufEreignis()*) legt den Oberflächenprozeß so lange schlafen, bis ein Ereignis eingetreten ist. Erst bei Eintritt eines Ereignisses wacht die Funktion auf und liefert eine Information über dessen Art oder Typ (z. B. „Taste A gedrückt“) zurück. (Vgl. accept() in der Systemprogrammierung)

```
1 while (true) {
2   e = wartenAufEvent();
3   switch(e.type) {
4       case keyA: tuwas(); break;
5       case keyB: tuWasAnderes(); break;
6       default: break;
7   }
8 }
```

Listing 4.1: wartende Eventsteuerung

Pollende Eventsteuerung

Hier sieht die Schleifenkonstruktion fast gleich aus. Einziger Unterschied: Die Funktion *nachfragenNachEvent()* kehrt sofort nach ihrem Aufruf wieder zurück. Falls kein Ereignis anstand, wird dies anhand des Rückgabewertes an den Aufrufer gemeldet.

```
1 while (true) {
2   e = nachfragenNachEvent();
3   switch(e.type) {
4       case keyA: tuwas(); break;
5       case keyB: tuWasAnderes(); break;
6       case noEvent: break;
7       default: break;
8   }
9 }
```

Listing 4.2: pollende Eventsteuerung

Objektorientierte Eventsteuerung

Hier ist die EventLoop in der Regel in einer eigenen Methode gekapselt. Das Reaktionsverhalten der Oberfläche ist meist in eigene Klassen ausgelagert.

4.2 Widget-Sets: Elemente zum Aufbau von GUIs

Ein Baustein zum Aufbau von Benutzungsoberflächen wird auch als ***Widget*** bezeichnet. Ein Widget kann z. B. ein Push-Button mit der Aufschrift „Ok“ sein. Ein Dialogfenster zum Suchen und Ersetzen ist ebenfalls ein Widget – es ist nur etwas komplizierter aufgebaut und besteht seinerseits aus vielen kleinen Widgets:

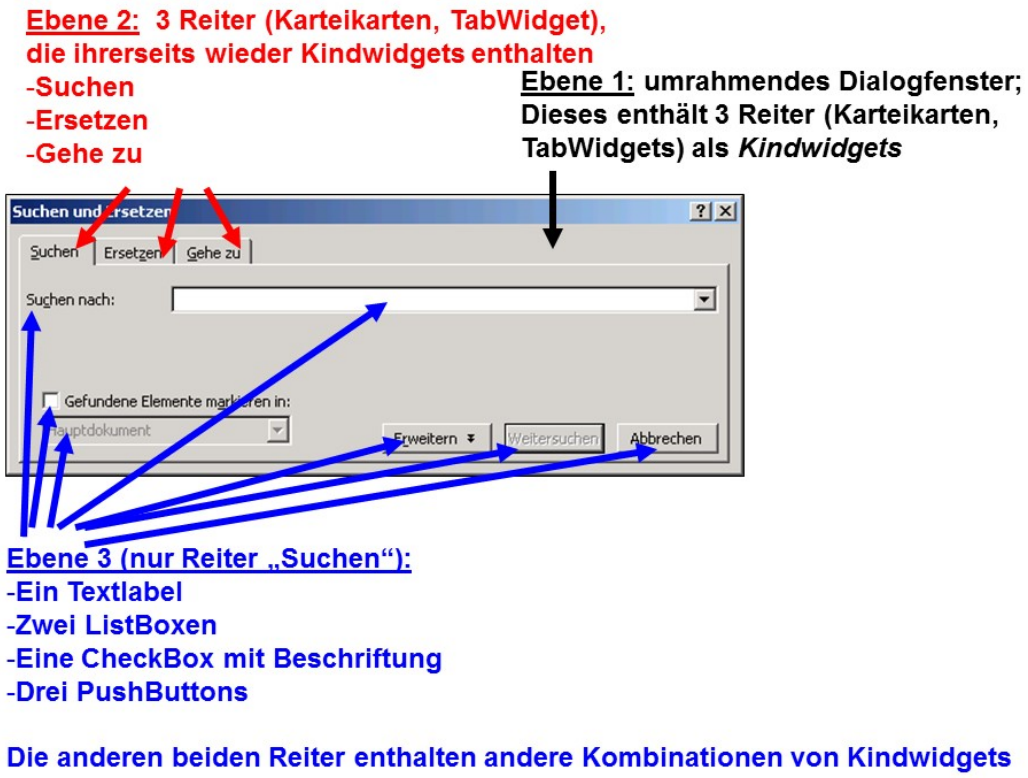

Abbildung 4.1: Widgets als Komposition aus kleineren Widgets

. Ein Widget wird in der OOP meist als Klasse implementiert. So gibt es unter SWT beispielsweise die Klasse *Button* für die auf Werkzeugleisten und Dialogfenstern häufig verwendeten PushButtons. Ein ***Widget-Set*** ist eine Software-Bibliothek, welche Widgets für den Aufbau von GUIs enthält.

Kompositionen aus Widgets, wie in Abbildung 4.1 werden als ***Widget-Hierarchie*** bezeichnet und oft in Form von Baumstrukturen dargestellt:

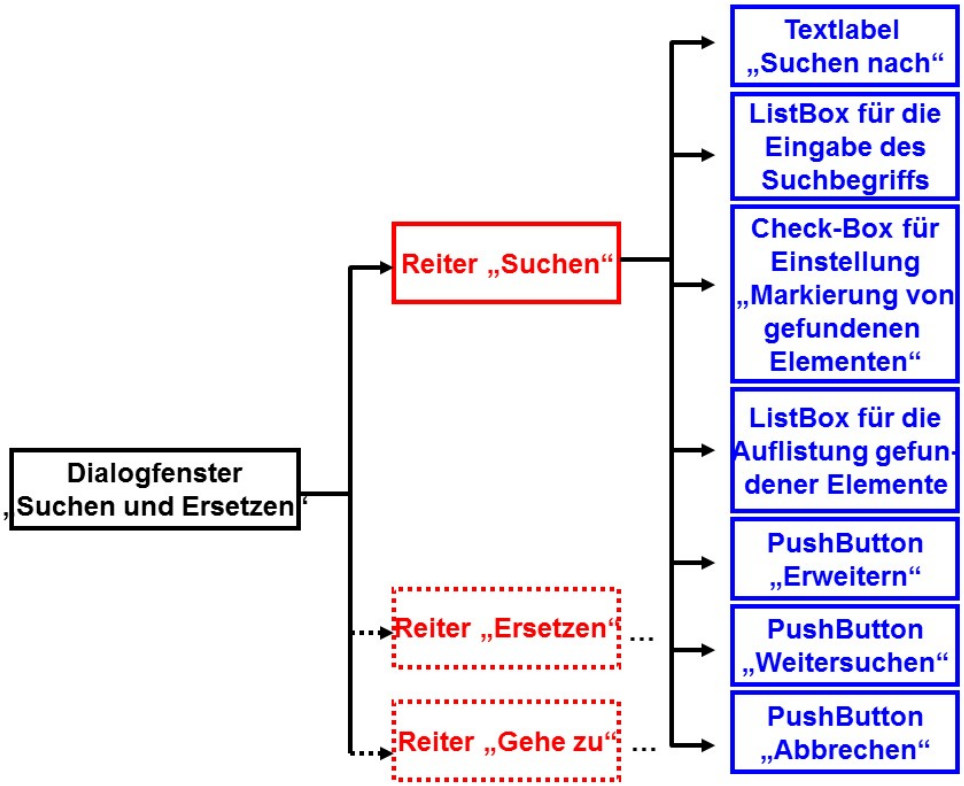

Abbildung 4.2: Widget-Hierarchie

4.3 Die wichtigsten Widgets

Unabhängig vom Hersteller lassen sich folgende Gruppen von Widgets unterscheiden:

Eingabe-Widgets

- TextEdit/LineEdit (Texteingabe)
- ComboBox (Klappliste)
- SpinBox (zum Eingeben, Inkrementieren und Dekrementieren von Zahlenwerten)

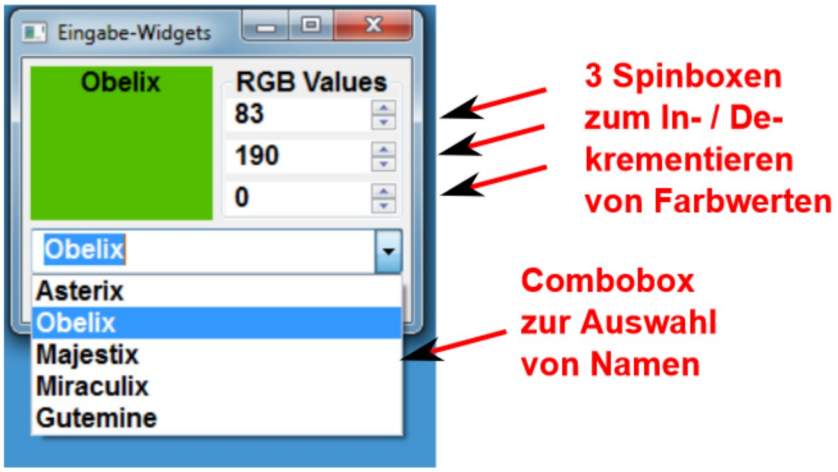

Abbildung 4.3: SpinBox und ComboBox

.

Buttons

- PushButton (einfache Schaltfläche)
- RadioButton (1-aus-N-Auswahl)
- CheckBox (Mehrfachauswahl, Häkchen)

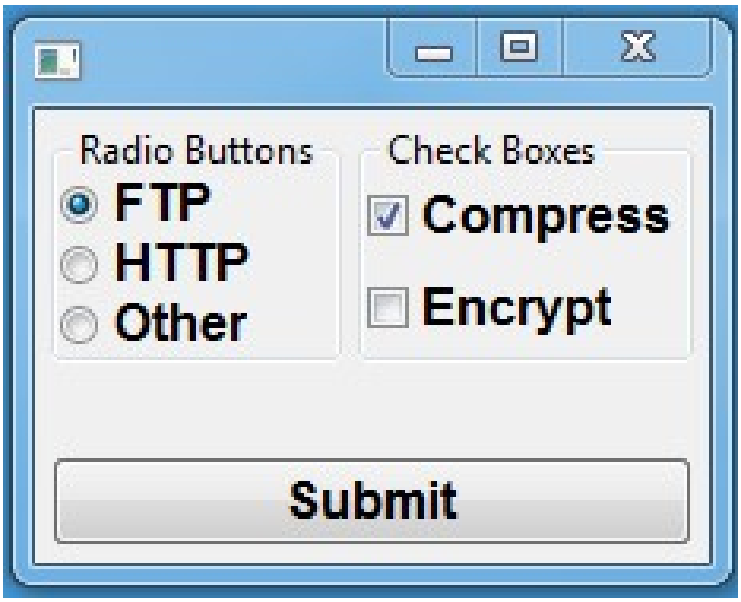

Abbildung 4.4: Beispiel mit verschiedenen Buttons

.

Anzeige-Widgets

- Label (Textanzeige)
- Icon (Bildanzeige)
- ProgressBar (Fortschrittsanzeige)

Fenster

- MainWindow (Hauptfenster mit Menüleiste, ggf. Toolbar)
- DialogWindow (Dialogfenster, z.B. Datei →Öffnen)
- Tabs (Reiter, werden gern in MDI-Oberflächen verwendet)
- ChildWindow (Sekundärfenster; wurden früher gern in MDI-Oberflächen verwendet)

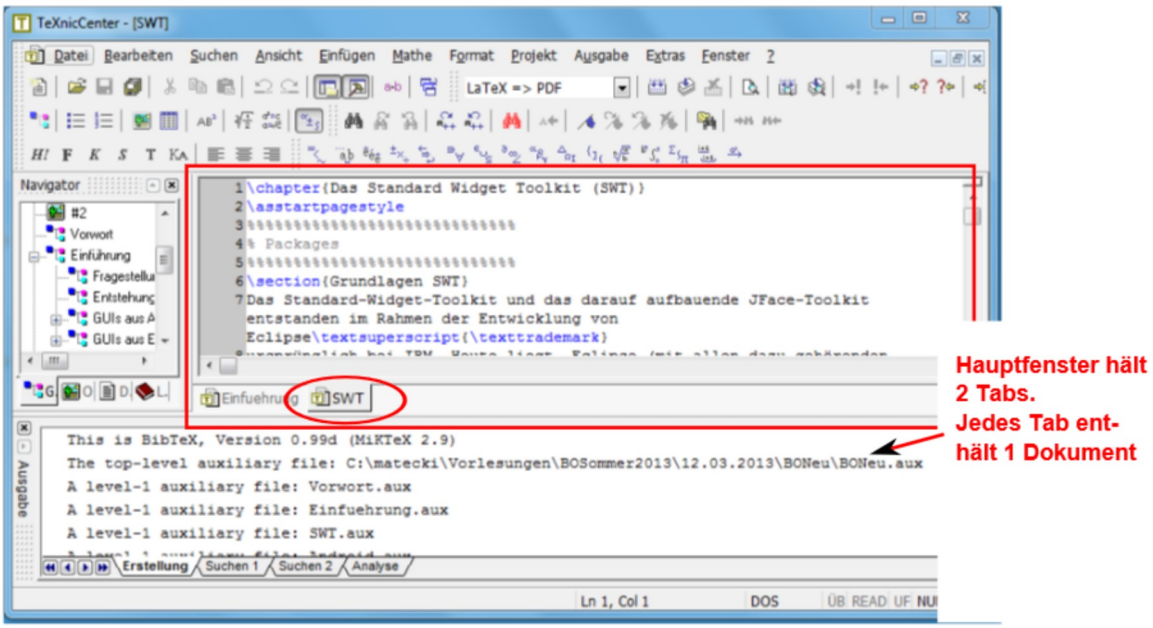

Abbildung 4.5: Hauptfenster mit Tabs

.

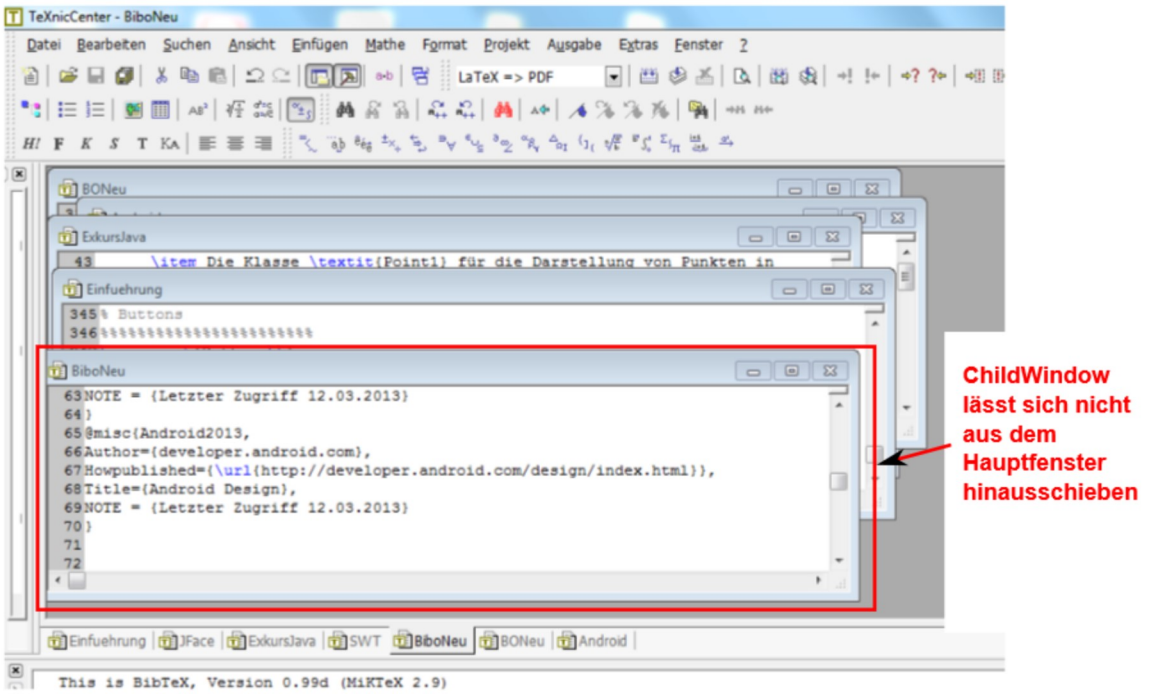

Abbildung 4.6: Hauptfenster mit Sekundärfenster

.

Menüs und ähnliche Auswahlmechanismen

Sie werden meist im Hauptfenster verwendet, um dem Nutzer unterschiedliche Aktionen anbieten zu können.

- MenuBar mit Pulldown-Menüs (Menüleiste oben am Hauptfenster)
- Pulldown-Menü: Wirkt sich global auf die gesamte Anwendung aus.
- MenuItem (einzelne Menüoption).

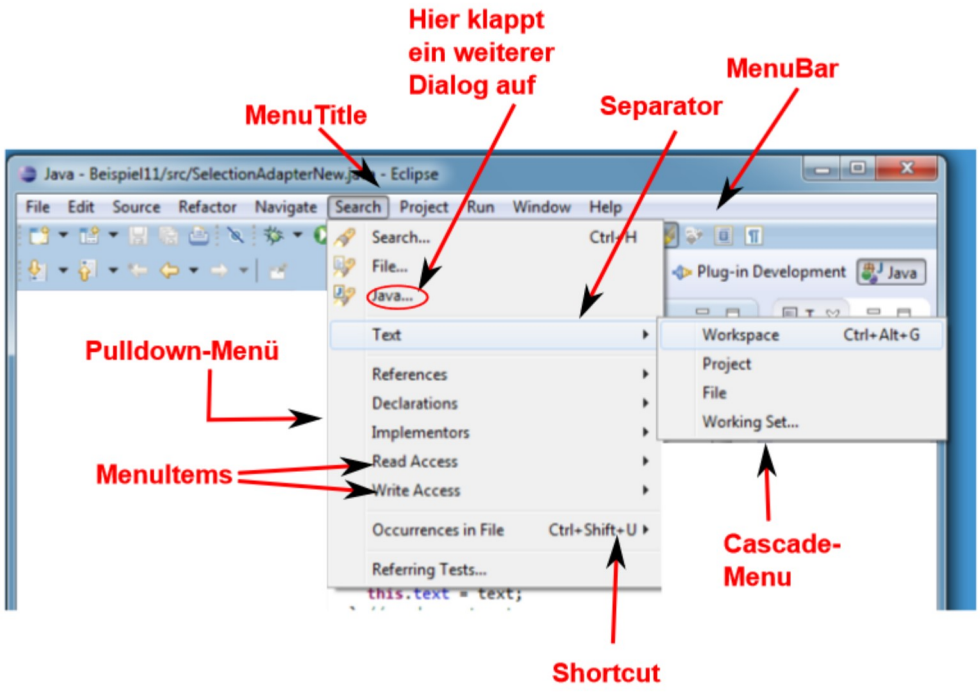

Abbildung 4.7: Menübalken

- Popup-Menü (Kontextmenü)
  - wird in der Arbeitsfläche des Hauptfensters mit der rechten Maustaste aktiviert,
  - wirkt sich lokal auf den Kontext, in dem es aktiviert wurde, aus.
  - hat keinen Menütitel
- ToolBar (Leiste mit Buttons für häufig genutzte Aktionen; meist unter der Menüleiste)
- ToolItem / ToolBarItem (einzelne Aktion auf dem ToolBar)

- CoolBar (ToolBar mit beweglichen Items)
- RibbonBar (Menüleisten-Ersatz bei vielen Microsoft Anwendungen, z. B. Office); Einzelne „Menüs“ erscheinen als Tab mit ToolItems

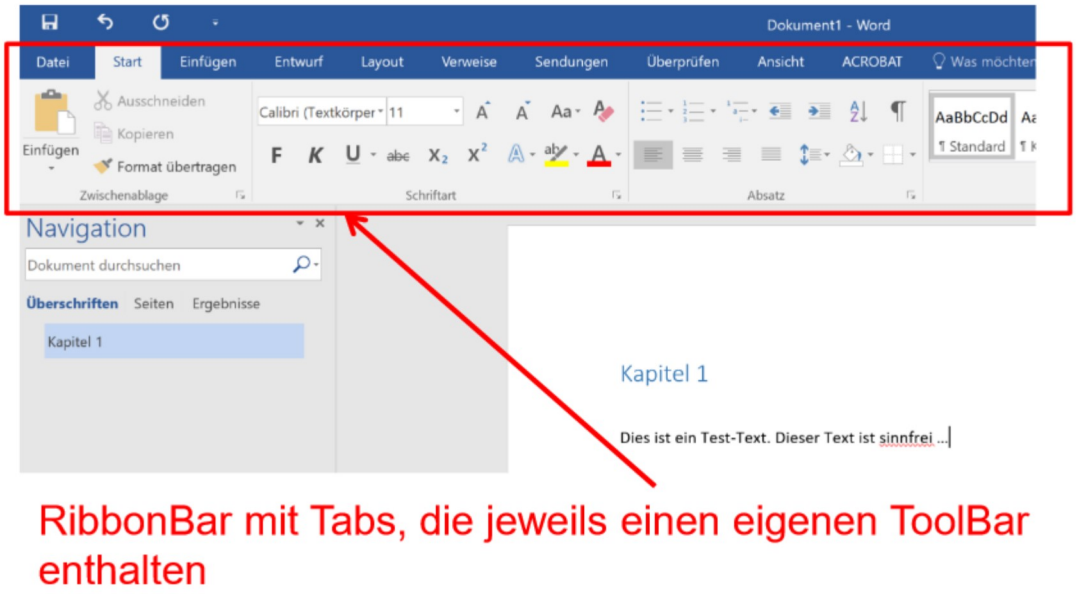

Abbildung 4.8: RibbonBar bei Micosoft Word

Menüoptionen können mit Tastaturkürzeln versehen werden. Hierbei gibt es zwei Arten von Kürzeln:

- ***Shortcuts:*** Beschleunigung der Auswahl durch direktes Ansprechen von Menüoptionen (kein Aufklappen notwendig)
  - Mindestens eine Taste ist eine Funktionstaste (z. B. Strg oder F1-F12)
  - Nur Menüoptionen, auf die direkt eine Aktion folgt, kann ein Shortcut zugeordnet werden (keine Menütitel und keine Kaskadenmenüs)
  - Shortcuts müssen über alle Menüoptionen des aktiven Fensters hinweg eindeutig sein!
  - *Strg+O* aktiviert direkt die Aktion der Menüoption *Datei* →*Öffnen*:
- ***Mnemonics:*** In Menütitel / Name der Menüoption kann eines der Zeichen des Namens als Tastenkürzel bestimmt werden. Dieses Zeichen ist dann unterstrichen.
  - Mnemonics für Menütitel werden meist mit *Alt+Zeichen* aktiviert. Danach ist das Menü aufgeklappt.
  - Mnemonics für die Optionen des aufgeklappten Menüs werden dann nur noch mit *Zeichen* aktiviert.
  - Beispiel: *Datei* →*Öffnen*: *Alt + D* klappt Dateimenü auf. Zeichen *f* wählt dann die Menüoption *Öffnen*.

4.4 Software-Entwurfstechniken für die Entwicklung von GUIs

Beim Entwurf von Oberflächen stellen sich zwei Fragen:

- Wie sollen die Arbeitsabläufe des Nutzers durch das Dialogverhalten der Oberfläche modelliert werden? →Modellierung der Requirements des Nutzers.
- Wie ist die Oberflächen-Software zu strukturieren? Hier gibt es unterschiedliche Architekturansätze, die alle ein gemeinsames Ziel verfolgen: Eine wartbare und leicht erweiterbare Software zu erstellen.

4.4.1 Modellierung der Arbeitsabläufe (Workflows)

Arbeitsabläufe werden häufig über Ablaufdiagramme oder UML-Aktivitätsdiagramme modelliert. Ein Beispiel hierzu:

Wir sind damit beauftragt, einen Editor zu erstellen, der im Augenblick nur 3 Funktionalitäten besitzen soll:

Anforderungen	
Nr.	**Beschreibung**
Afo1	Bereits als Textdatei gespeicherte Dokumente sollen in den Editor geladen werden können. Hierbei sind geeignete Rückfragen zu stellen und Rückmeldungen zu geben.
Afo2	Bearbeitete Dokumente im Editor sollen als Textdatei abgespeichert werden können. Hierbei sind geeignete Sicherheitsabfragen zu stellen.
Afo3	Bereits geladene Dokumente sollen im Editor editiert werden können.

Tabelle 4.1: Anforderungen

Eine Modellierung des Arbeitsablaufes für z. B. Afo1 könnte folgendermaßen aussehen:
Die folgenden Bilder zeigen die wichtigsten Sprachelemente für UML2.5-Aktivitätsdiagramme:

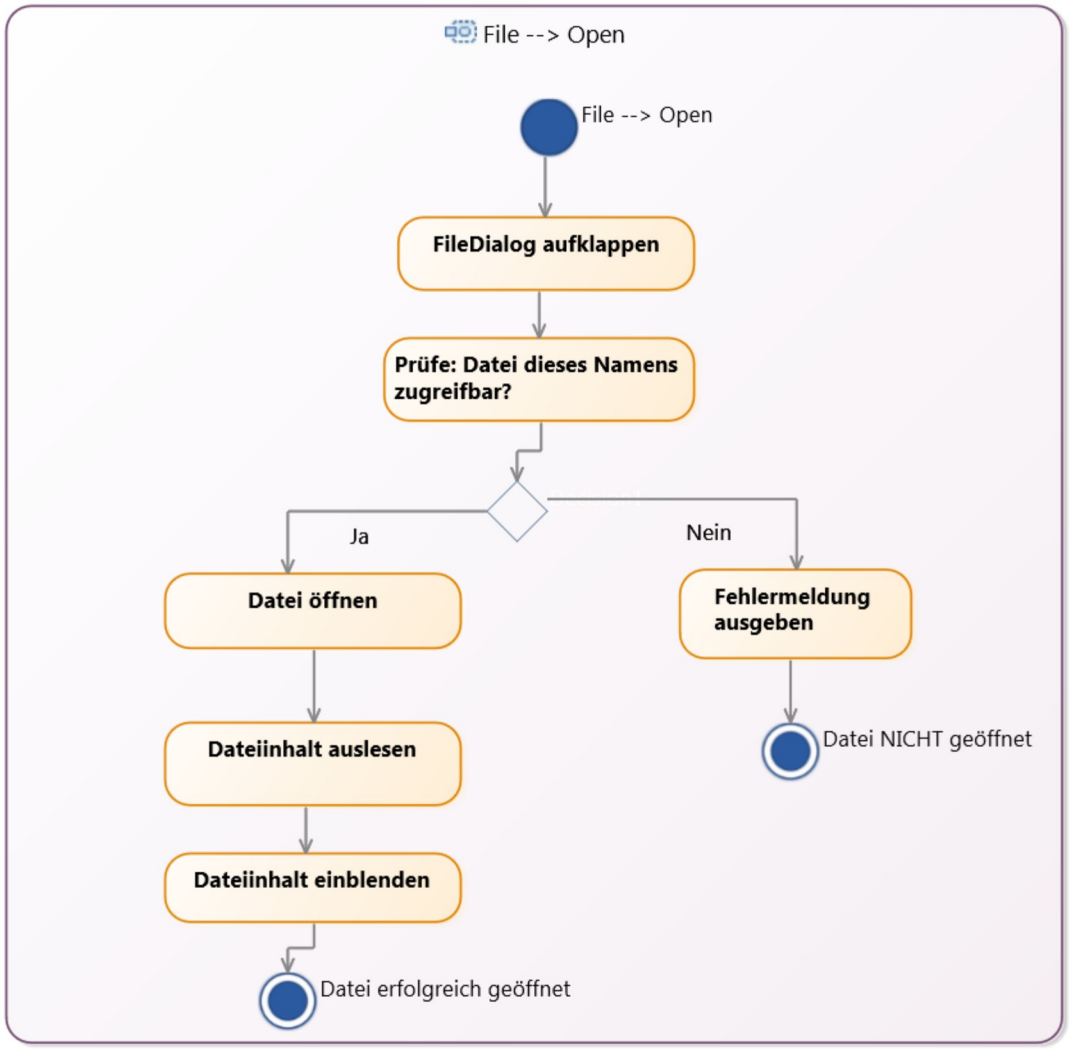

Abbildung 4.9: Modellierung von Afo1 mit Aktivitätsdiagramm

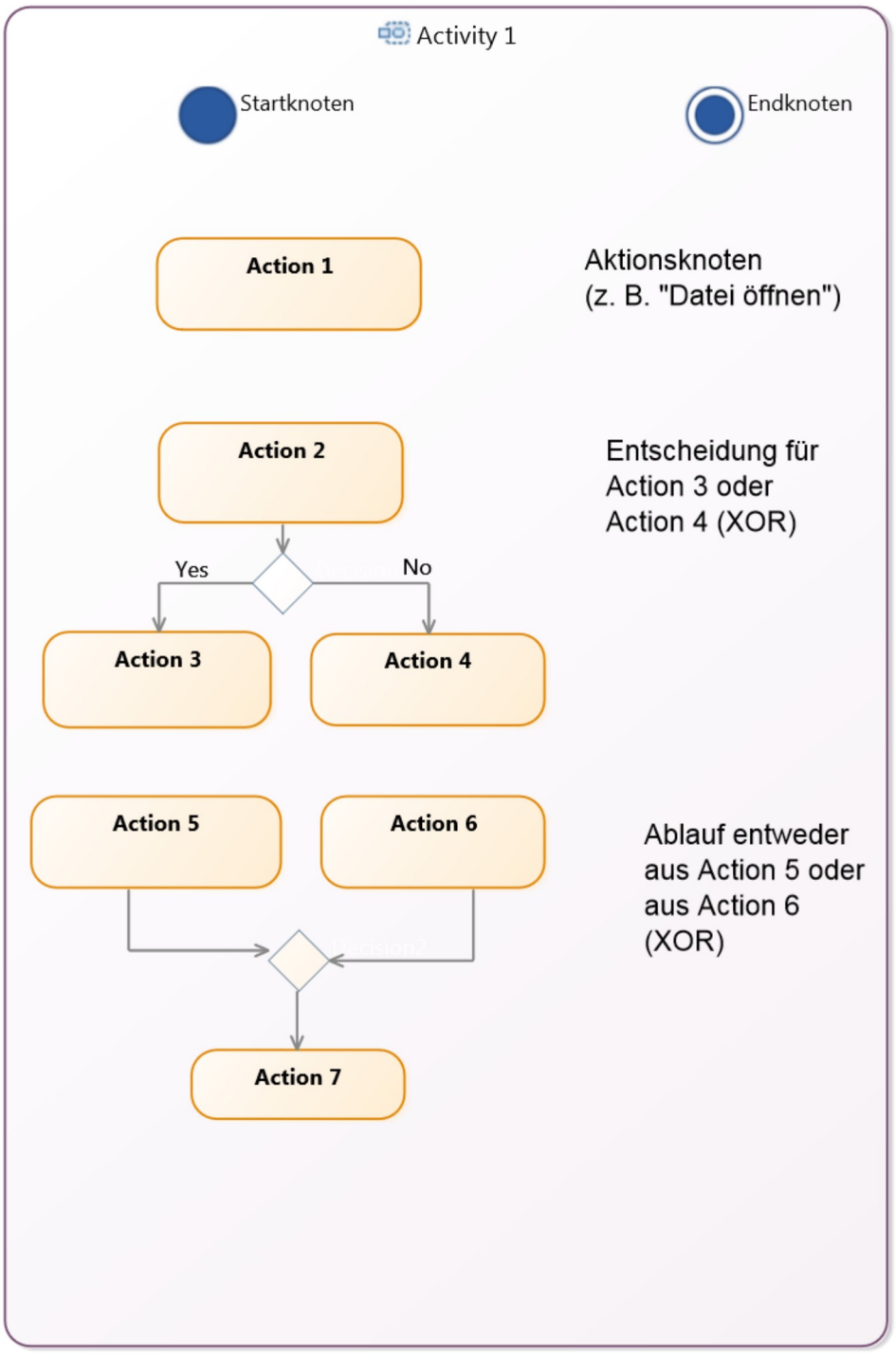

Abbildung 4.10: Sprachelemente UML für Aktivitätsdiagramme – Teil 1

Eine ***Activity*** ist hierbei ein vollständig modellierter Ablauf. Eine ***Action*** ist ein Aktionsknoten innerhalb des Diagramms.

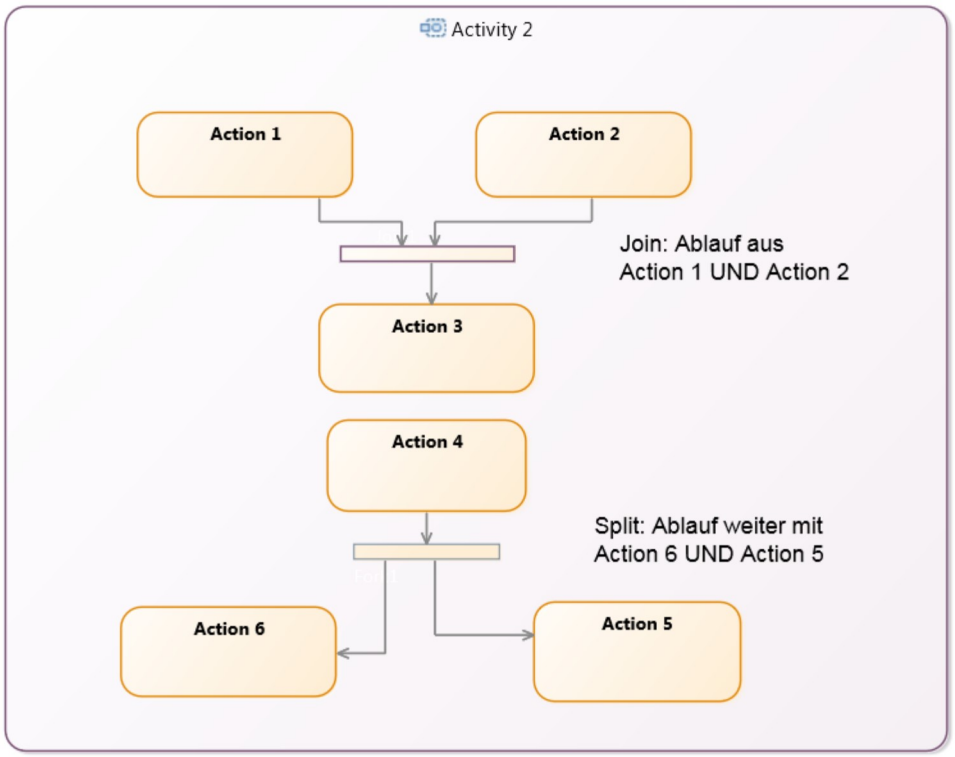

Abbildung 4.11: Sprachelemente UML für Aktivitätsdiagramme – Teil 2

4.4.2 Architekturparadigmen für GUIs

Benutzungsoberflächen sind eine Schnittstelle zwischen Mensch und Rechnersystem. In der Regel ermöglicht eine Benutzungsoberfläche es dem Menschen, auf Daten zuzugreifen, Berechnungen anzustoßen, usw... . Mit anderen Worten: Die Benutzungsoberfläche stößt in der Regel Software-Komponenten an, welche selbst nichts mit der Oberfläche zu tun haben. Beispiele für derartige Software-Komponenten:

- Datenbank-Abfragen
- Dateizugriffe auf Dateien unterschiedlicher Formate
- Netzwerkzugriffe
- Algorithmen

Mit anderen Worten: GUI-Anwendungen enthalten

- oberflächenbezogene und
- nicht oberflächenbezogene

Software-Komponenten.

Merke

Eines der wichtigsten Architekturparadigmen ist es, Oberflächencode von Nicht-Oberflächencode streng zu trennen. Ansonsten erhalten wir nicht-änderungsfreundlichen, monolithischen Code.

Weiterhin wird beim Oberflächencode unterschieden zwischen:

- Präsentationsverhalten (Layout, Fonts, etc. – statisch) und
- Dialogverhalten (Reaktion auf Nutzeraktionen, Ein-/Ausblenden – dynamisch).

Das Präsentationsverhalten kann direkt ausprogrammiert werden. Es kann aber auch mit entsprechenden Designprogrammen gezeichnet werden. Diese erzeugen dann i.d.R. XML- Beschreibungen des Oberflächen-Layouts, welche anschließend in die Zielsprache übersetzt werden. Wir werden beide Wege kennenlernen.
Auf Basis dieser Begriffsbildungen funktioniert das ***3-Schichtenmodell***:

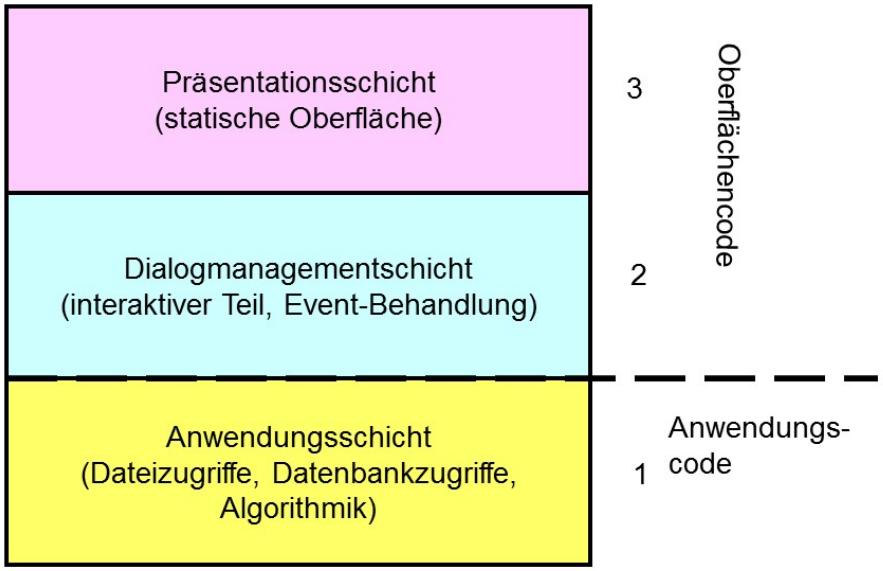

Abbildung 4.12: 3-Schichten-Modell für GUIs

.

Für umfangreichere GUI-Systeme, die auch anspruchsvolle Datenhaltungskonzepte beinhalten, kommt die ***M***odel ***V***iew ***C***ontroller-Architektur zum Einsatz: . Sie wird oft irrtüm-

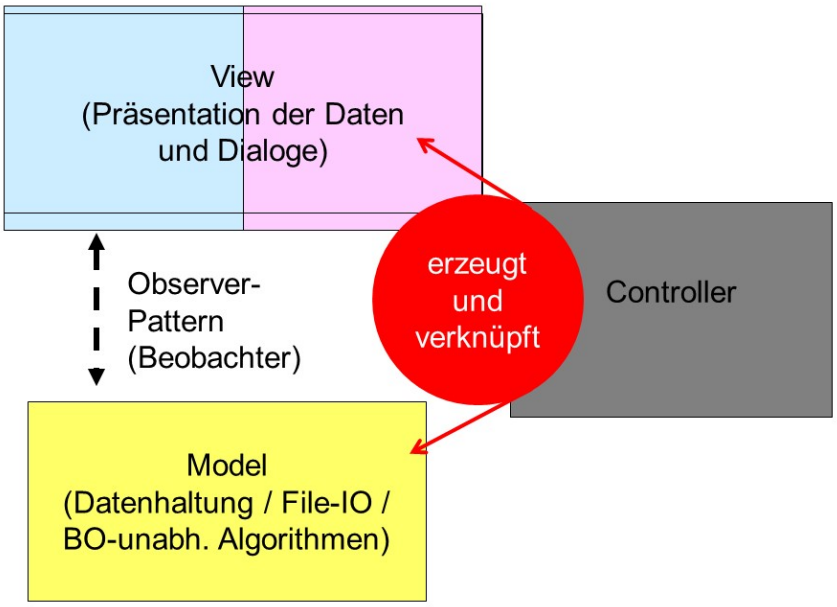

Abbildung 4.13: MVC-Architektur für GUIs

licherweise mit der 3-Schichten-Architektur gleich gesetzt. Bei einer MVC-Architektur existieren zwar ebenfalls 3 Teilbereiche. Sie sind jedoch etwas anders aufgeteilt als in der 3-Schicht-Architektur:

- Model: entspricht weitgehend der Anwendungsschicht im 3-Schichtenmodell. Jedoch ist im Model eine reaktionsfähige „Verschalung“ der Datenhaltung enthalten, die auf inhaltliche Änderungen in der Oberfläche – z. B. einer Tabelle – reagiert und den Datenbestand aktualisiert.
- View: Der Teil der Oberfläche, der für die Datendarstellung zuständig ist – z. B. ein *TreeViewer* oder *TableViewer*.
- Controller: Der Teil der Oberfläche, welcher die View steuert, sowie View und Model miteinander verknüpft.