7.2 Struktur grafischer Oberflächen

von Kerstin Dittert

Logische Schichten der Oberfläche

Das Prinzip der Strukturierung durch Schichtenbildung haben Sie bereits in Kapitel 2 und 3 kennengelernt. Für den Entwurf grafischer Oberflächen besitzt es besondere Bedeutung, um eine starke Kopplung von Oberfläche, Ablaufsteuerung und Datenhaltung zu vermeiden. Klare Abgrenzung von Verantwortlichkeiten zwischen den Schichten verbessert die Wartbarkeit des Programms und bildet die Grundlage für zukünftige Erweiterungen.

Häufig unterteilt man die Präsentationsschicht (siehe Kapitel 3) noch weiter *GUI*[1]*-Schicht* und *Präsentationsschicht*. Die GUI-Schicht enthält die rein grafischen Elemente der Oberfläche, während die Präsentationsschicht sich mit den ablaufrelevanten Aspekten beschäftigt und die Fachmasken beinhaltet. Darunter liegen die (meist serverseitigen) Schichten *Domäne* (Fachlogik) und *Persistenz* (Datenhaltung)[2].

Wird die Präsentationsschicht auf dem Client abgebildet, so spricht man von einem *Fat-Client*, da neben der Benutzerschnittstelle auch die komplette Ablaufsteuerung auf dem Client verwaltet wird. In diesem Fall wird oft auf eine explizite Trennung zwischen GUI- und Präsentationsschicht verzichtet.

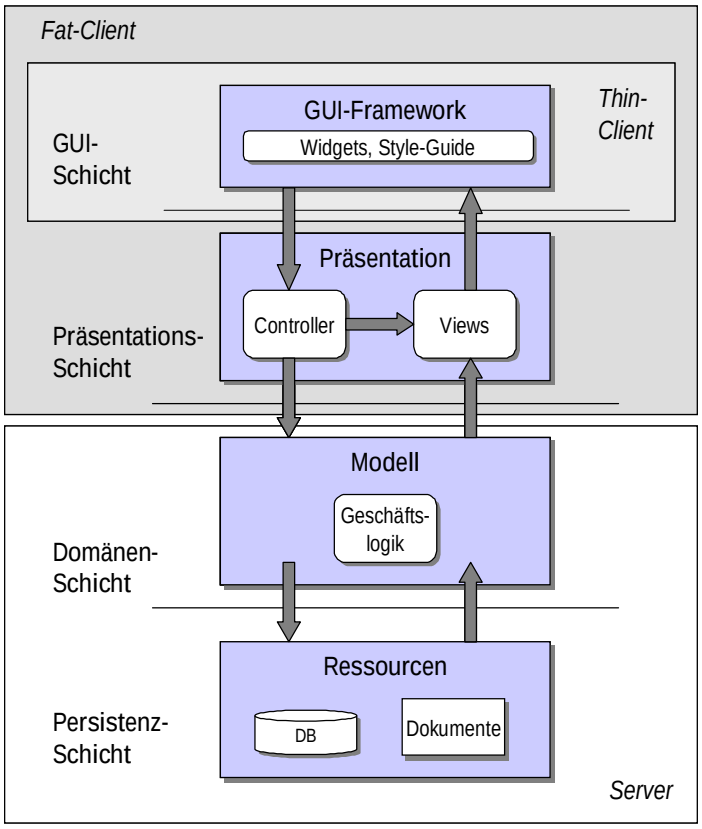

Abbildung 1: Schichtenbildung

[1] GUI: Graphical User Interface = grafische Benutzerschnittstelle.
Diese Benutzeroberflächen zeichnen sich durch den Einsatz von Fenstern und vielen verschiedenen Steuerungs- und Navigationselementen aus, die im Allgemeinen mit der Maus bedient werden. Auf PCs mit modernen Betriebssystemen werden für kommerzielle Anwendungen fast ausschließlich GUI's als Benutzerschnittstelle verwendet.

[2] Sind die persistenten Ressourcen der Anwendung stark verteilt, so wird häufig zusätzlich eine *Integrations*-Schicht zwischen Modell und Persistenz eingeführt. Diese ist für die Verarbeitung unterschiedliche Kommunikationsprotokolle, Transaktionsmonitore oder Datenbanken verantwortlich.

Beinhaltet der Client hingegen ausschließlich die GUI-Schicht, so handelt es sich um einen *Thin*-Client, da die Navigation der Anwendung zentral vom Server gesteuert wird.

Die GUI- und die Präsentationsschicht beinhalten zusammen die wesentlichen Komponenten zur grafischen Aufbereitung der Fachlogik und zur Steuerung des Arbeitsablaufes.

Im Projektverlauf betreffen die meisten Änderungswünsche die Benutzerschnittstelle und die Navigation, so dass dem Entwurf der GUI- und Präsentations-Schicht eine zentrale Bedeutung zukommt. Daher sollten die Schichten sollten möglichst einfach und flexibel gestaltet werden, so dass auf neue Anforderungen reagiert werden kann, ohne die gesamte System-Architektur in Frage zu stellen.

GUI-Schicht

Diese Schicht umfasst alle technischen Komponenten, die zur Komposition der Fachmasken verwendet werden. Dieses können bei einer Internet-Anwendung zum Beispiel HTML-Seiten mit einfachen Navigations- und Steuerungselementen sein. Bei einer komplexeren Anwendung mit objektorientierter Oberfläche kann diese Schicht aus vielen verschiedenen GUI-Widgets[3] und speziellen Anpassungen dieser Widgets bestehen.

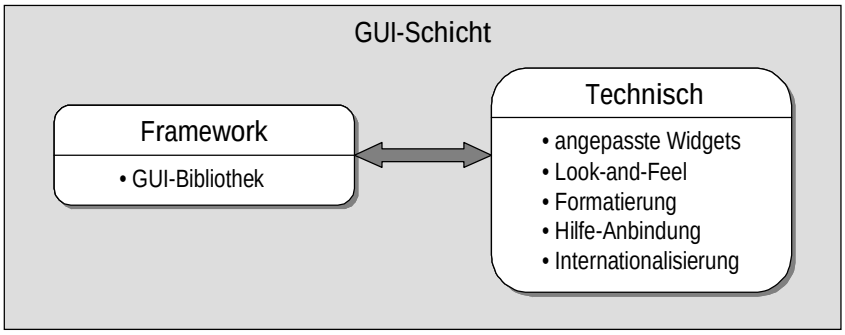

Abbildung 2 – Elemente der GUI-Schicht

Wenn viele verschiedene Widgets verwendet werden, so ist es häufig sinnvoll die Basis-Widgets des Frameworks anwendungsspezifisch anzupassen und diese in einer technischen Subkomponente der GUI-Schicht zu verwalten. Hier kann ein spezielles Aussehen (*Look-and-Feel*) der Anwendung realisiert wer-

[3] Ein Widget ist eine grafische Basis-Komponente die zur Komposition von Fenstern und Masken verwendet wird (z.B. Button, Texteingabefeld oder Dialog-Box). Widgets werden in speziellen GUI-Bibliotheken zur Verfügung gestellt, welche in ihrem Aussehen und Verhalten auf das zugrunde liegende Betriebssystem abgestimmt sind („Look-and-feel“). Beispiele derartiger Bibliotheken sind die Swing-Klassen für Java oder die Microsoft Foundation Classes für C++ unter Windows-Betriebssystemen.

den. Zusätzlich können dort spezifische Erweiterungen wie Formatierung, Maskierung und einfache Validierung umgesetzt werden. Diese technischen Basiselemente sind sowohl Empfänger als auch Auslöser aller elementaren GUI-Ereignisse wie etwa „Button gedrückt“ oder „Text geändert“. Sind derartige Anpassungen nicht erforderlich, so besteht die GUI-Schicht ausschließlich aus den Klassen des eingesetzten Framework.

Ein guter Entwurf zeichnet sich durch die harmonische Integration der angepassten Widgets aus. Hierfür können die nachfolgenden Punkte als Richtschnur dienen:

- Nutzen Sie die GUI-Widgets, die Ihnen das Framework zur Verfügung stellt.
- Passen Sie diese gegebenenfalls an, aber erfinden Sie nicht jedes Rad neu (auch wenn es ein paar Speichen mehr hat ...). Der beste Code ist der, den Sie und Ihr Team nicht entwerfen, programmieren und testen müssen! So steigern Sie die Robustheit und Wartbarkeit ihres Systems.
- Arbeiten Sie nicht „gegen“ das Framework, selbst wenn einige Eigenschaften der Framework-Architektur dies wünschenswert erscheinen lassen. Viele Frameworks kennen keine Trennung von View und Controller (siehe Abschnitt 7.4.1). Statt in diesem Fall alles „von Hand“ zu implementieren, sollten Sie das MVC-Pattern so modifizieren, dass es zur Arbeitsweise des Frameworks passt (etwa durch Verschmelzung der View- und Controller-Klassen). Andernfalls riskieren Sie einen hohen Mehraufwand. Neben den zusätzlichen Design- und Realisierungsaktivitäten führt die ineffiziente Vorgehensweise zur Frustration Ihres Teams und damit zu sinkender Produktivität.
- Finden Sie eine stimmige Balance zwischen Wiederverwendung, Variantenbildung und Einfachheit des Entwurfs. So ist es beispielsweise sinnvoll, spezielle Klassen für einen Abbrechen-, Speichern- und Hilfe-Button zu definieren, die in den Fachmasken die jeweilige Funktionalität ausführen. Eine noch höhere Abstraktion, welche die unterschiedlichen Aufgaben durch Parametrisierung umsetzt, führt häufig zu einer mangelnden fachlichen Klarheit des Entwurfs.

Präsentations-Schicht

Diese Schicht bereitet alle fachlichen Aspekte der Domäne auf: Sie steuert den Anwendungsablauf, verwaltet den Zustand und beinhaltet die Fachmasken der Anwendung.

Beim Entwurf der Schicht ist eine Zerlegung in kleinere Komponenten unabdingbar. Diese Aufgabenverteilung erfolgt in der Regel nach dem Model-View-Controller-Entwurfsmuster (vgl. Abschnitt 7.4.1).

Model-View-Controller (MVC)
Kapitel 7.4.1

Die Views der Präsentationsschicht entsprechen den Fachmasken der Anwendung, welche die Elemente der GUI-Schicht benutzen. Die Masken-Hierarchie orientiert sich an der fachlichen Struktur des Systems, so dass eine enge Kopplung zwischen Präsentationsschicht und Domäne besteht. Vermeiden Sie den übertriebenen Einsatz generischer Ansätze, um dieser Kopplung zu entgehen! Der vermeintliche Vorteil der unbegrenzten Erweiterbarkeit verkehrt sich schnell in sein Gegenteil:

- Delegationen durch die vielen Schichten generischer Komponenten erweisen sich häufig als echte Performance-Killer.
- Generische Ansätze erfordern meist eine externe Konfiguration der Komponenten. Viele Fehler werden deshalb nicht mehr zur Übersetzungszeit, sondern erst zur Laufzeit entdeckt. Dies macht die Anwendung fehleranfälliger und aufwändiger im Test.

Bei vielen generischen Ansätzen gilt deshalb die Devise: „Nach allen Seiten offen ist nicht ganz dicht![4]“. Akzeptieren Sie die enge Kopplung von Domäne und Präsentationsschicht und stimmen Sie den Architektur-Entwurf darauf ab.

- Entwerfen Sie die Präsentationsschicht und die Domäne parallel.
- Halten Sie den Entwurf so einfach wie möglich.
- Dokumentieren Sie die Abhängigkeiten der beiden Schichten.
- Verbessern Sie die Wartungs- und Erweiterungsmöglichkeiten durch den Einsatz automatischer Testwerkzeuge.

Die Präsentationsschicht verwendet für die Aufbereitung der fachlichen Daten technische Modelle, welche auf die Bedürfnisse der Masken optimiert sind. Beim Entwurf dieser Modelle wird im ersten Schritt häufig das Analyse-Modell zu Grunde gelegt. Dieser Ansatz erscheint naheliegend, da die Benutzerschnittstelle ja diese Informationen darstellen soll. Häufig kann das Analyse-Modell jedoch nicht Eins-zu-Eins von der Präsentationsschicht verwendet werden. Enthaltene Abstraktionen sind für den Anwender irrelevant und oft unverständlich.

- „Denormalisieren“ [5] Sie das Analysemodell.
- Orientieren Sie sich an der fachlichen Hierarchie.
- Berücksichtigen Sie Performanz-Anforderungen, die sich aus dem Datenfluss zwischen den Masken, der GUI-Schicht und der Domäne ergeben.

Ablaufsteuerung grafischer Oberflächen Kapitel 7.4

Die Präsentationsschicht arbeitet zwar mit Modellen, sie ist aber nicht für die Fachlogik der Anwendung verantwortlich. Sie bereitet Inhalte auf und steuert den Fluss, die Geschäftsregeln sind jedoch innerhalb der Domäne repräsentiert. Betrachten wir die Geschäftsprozesse, die einer Anwendung zu Grunde liegen, so gehört die *Ablaufsteuerung der Masken* zum Aufgabengebiet der

[4] nach Petra Bremer.

[5] vergleichbar dem Prozess der Denormalisierung von logischen Datenmodellen, bevor diese in ein physisches Datenbankschema umgesetzt werden.

Präsentationsschicht. Einzelne *Geschäftsregeln* fallen hingegen in den Verantwortungsbereich der Domänen-Schicht.

Es ist oft schwierig zu entscheiden, welche Schicht die Verantwortung für die Implementierung einer bestimmten Funktionalität hat. Ausschlaggebend ist hierfür stets die Überlegung, ob es sich um eine Steuerungs-Aufgabe oder die Umsetzung von Geschäftsregeln handelt. Folgende Frage hilft bei der Entscheidung:

> *Benötige ich die Funktionalität noch, wenn ich die grafische Benutzeroberfläche durch eine Batch-Schnittstelle ersetze?*

- Können Sie diese Frage bejahen, so ist diese Fachlogik ein Bestandteil der Domäne.
- Im anderen Fall wird sie innerhalb der Präsentations-Schicht umgesetzt.

Typische Beispiele für *Logik innerhalb der Präsentationsschicht* sind folgende Aufgaben:

- Status-Informationen über Objekte in der Bearbeitung.
- Angaben über erlaubte Aktionen im aktuellen Zustand.
- Status- und Fortschrittsinformationen.
- Fehlerinformationen.

Unterschiedliche GUI-Typen
Kapitel 7.2

Die Menge der ausgetauschten Informationen zwischen Domänen- und Präsentationsschicht hängt von verwendeten Benutzeroberflächen-Typ und dem zu Grunde liegenden Transaktionsmodell ab. Objektorientierte Oberflächen benötigen mehr Informationen als formularbasierte, so dass hier eine stärkere Kopplung unvermeidlich ist.

Größere Anwendungen fordern vielfach, dass die Präsentationsschicht verschiedene GUI-Schichten bedienen kann (etwa einen HTML-Client, einen WAP-Client oder ähnliche)[6]. In einem solchen Fall muss auf Grund der fachlichen Kopplung der drei Schichten GUI, Präsentation und Domäne darauf geachtet werden, dass die Domänen-Schicht niemals auf die Präsentationsschicht zugreift. Die Verarbeitung innerhalb der Domäne darf nicht davon abhängen, wie der Fluss gesteuert wird, oder die Inhalte dargestellt werden. Die Präsentationsschicht richtet dann also Anfragen an die Domäne, aber niemals umgekehrt[7]!

[6] diese Anforderung wird auch mit dem Begriff „Multichannel“-Anwendung bezeichnet.

[7] Diese Forderung haben wir bereits bei der Einführung des Schichten-Musters in Kapitel 2 formuliert!

7.3 Ergonomie grafischer Oberflächen

von Kerstin Dittert

Grafische Benutzeroberflächen (GUI) bieten im Vergleich zu alphanumerischen Benutzerschnittstellen viele Gestaltungsmöglichkeiten. Dies betrifft sowohl die Arbeitsabläufe als auch die Art der Darstellung.

Aus Sicht von Software-Architekten stellt die Benutzeroberfläche eine der wichtigsten Schnittstellen des Systems dar.

Wir stellen Ihnen in diesem Abschnitt die wichtigsten Aspekte der Gestaltung von Benutzeroberflächen vor:

- Arbeitsmetaphern veranschaulichen die Struktur von Arbeitsabläufen.
- Metaphern korrespondieren mit objekt- und formularorientierten Oberflächen.
- Die Berücksichtigung ergonomischer Aspekte steigert die Effizienz ihres Systems.

7.3.1 Arbeitsmetaphern

Ablaufsteuerung grafischer Oberflächen Kapitel 7.4

Metaphern bieten uns Bilder aus der wirklichen Welt an, um die Struktur von Arbeitsabläufen und Anwender-Interaktionen zu veranschaulichen. Dabei deckt eine Metapher nicht immer alle Anwendungsfälle eines Systems ab. Oft fallen einzelne Anwendungsfälle oder Bearbeitungsschritte aus dem groben Raster heraus. Für die weitaus meisten Software-Systeme ist eine der beiden nachfolgenden Metaphern anwendbar.

Fabrik

In der Fabrik erfolgt ein Arbeitsschritt nach dem anderen. Das Ausgangsprodukt eines Bearbeitungsschrittes wird zum Eingangsprodukt des nächsten Arbeitsschrittes. Die Produktionsbänder einer Fabrik können sich verzweigen und wieder vereinigen. Jedes Produkt folgt jedoch nur einem Weg vom Ausgangsprodukt zum Endprodukt. Auf diesem Weg ist die Reihenfolge der Arbeitsschritte fest vorgegeben.

Häufig gibt es zwischen den einzelnen Schritten eine Qualitätskontrolle, die bisherige Ergebnisse überprüft und fehlerhafte Zwischenergebnisse ablehnt. Sind alle Arbeitsschritte durchlaufen, so liegt ein fehlerloses Produkt vor.

Der Arbeitsablauf folgt starren Regeln. Abkürzungen auf dem Weg zum gewünschten Endprodukt sind in der Regel nicht möglich. Die starre Vorgabe begünstigt eine leichte Erlernbarkeit der Arbeitsschritte und führt zu einem hohen Durchsatz. Die Arbeit des einzelnen Anwenders ist auf effiziente Massenverarbeitung ausgerichtet und in der Regel wenig kreativ.

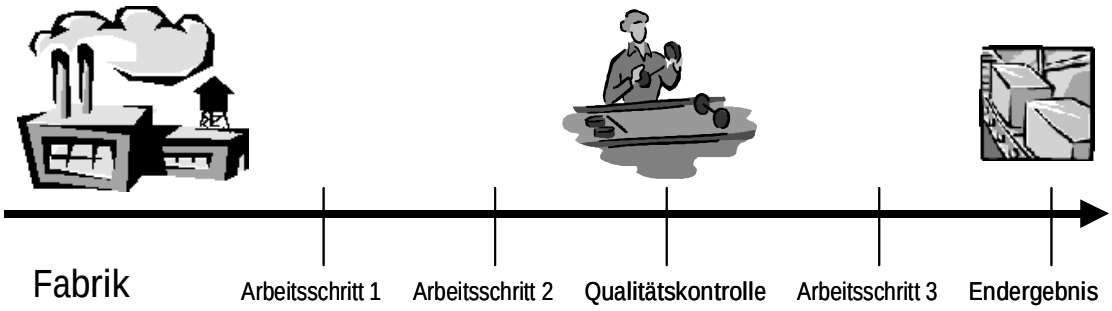

Abbildung 3 – Fabrik-Metapher

Diese Metapher ist passend für Systeme, die

- selten verwendet werden und daher an den Arbeitsablauf „erinnern“ müssen,
- von Anwendern benutzt werden, die nur wenig Erfahrung im Umgang mit EDV-Systemen haben (etwa Geldautomat),
- einen stark sequentiellen Arbeitslauf haben (etwa Internet-Shop).

Werkzeug-Material

Das Leitbild dieser Metapher ist ein Arbeitsplatz für qualifizierte Tätigkeiten, bei denen der Mensch im Mittelpunkt steht. Das Umfeld setzt die entsprechende Qualifikation zur Bewältigung der Aufgaben voraus und unterstützt die fachliche Weiterentwicklung des Anwenders. Die Arbeitsumgebung ist auf individuelle Arbeitsschwerpunkte und Erfahrung des Einzelnen hin anpassbar. Die fachlichen Konzepte sind klar erkennbar und unterstützen kreative Arbeit und komplexe Tätigkeiten.

Der Arbeitsablauf ist wenig strukturiert. Die vielzähligen Kombinationsmöglichkeiten von Werkzeug und Material, sowie beliebige Reihenfolgen der Arbeitsschritte definieren unterschiedliche Wege zum Ziel.

In der Domäne werden Werkzeuge, Material und der Verwendungszusammenhang unterschieden. Diese entsprechen den Arbeitsmitteln, den Arbeitsgegenständen und den Tätigkeiten. Eine Tätigkeit besteht in der Anwendung eines Werkzeuges auf das zu bearbeitenden Material. Die Prüfung der Arbeitsergebnisse kann implizit nach Anwendung eines Arbeitsschrittes erfolgen, oder explizit auf Anforderung des Bearbeiters.

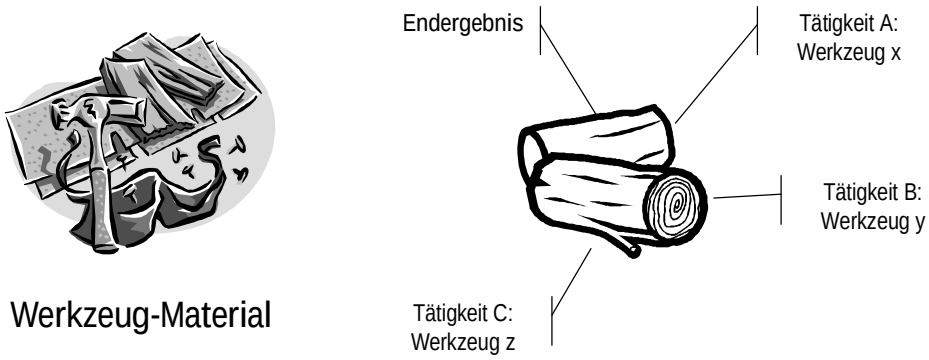

Abbildung 4 – Werkzeug-Material-Metapher

Die Metapher eignet sich für Anwendungen, die

- unterschiedliche Arbeitsabläufe für gleichartige Aufgaben abbilden,
- kreative Tätigkeiten durch eine Vielzahl von Wahlmöglichkeiten unterstützen (beispielsweise Bildbearbeitung),
- von Experten benutzt werden, die das System entsprechend ihrer individuellen Fähigkeiten optimal nutzen möchten.

Auswahl einer Arbeitsmetapher

- Bestimmen Sie die für Ihr System passende Arbeitsmetapher.
- Entwerfen Sie sämtliche Elemente der Benutzeroberfläche stets im Hinblick auf die Arbeitsmetapher. Der Oberflächenentwurf und die Auswahl einer geeigneten Ablaufsteuerung hängen mit der Metapher zusammen.
- Folgen unterschiedliche Anwendungsfälle ihres Systems anderen Metaphern, so stimmen Sie ihre System bewusst darauf ab. Versuchen Sie nicht, alle Anwendungsfälle künstlich auf die gleiche Metapher „zu trimmen“. Dies führt zu umständlicher und wenig intuitiver Navigation und vielen Anwenderfehlern. Statt dessen
  - wählen Sie für jeden Anwendungsfall die geeignete Metapher,
  - kategorisieren Sie die Anwendungsfälle im Hinblick auf die Metaphern,
  - zeichnen Sie die Leit-Metapher des Systems aus und entwerfen Sie die Oberfläche auf Basis dieser Leit-Metapher.

7.3.2 Unterschiedliche GUI-Typen

Moderne grafische Benutzerschnittstellen lassen sich im Wesentlichen zwei Grundtypen zuordnen: den *objektorientierten Oberflächen* für die Werkzeug-Material-Metapher und den *formularbasierten Oberflächen* für die Fabrik-Metapher.

Objektorientierte Oberfläche

In einer objektorientierten Oberfläche[8] wählt der Anwender zunächst ein Objekt aus und bearbeitet es anschließend. Die Auswahl der Bearbeitungsschritte erfolgt auch visuell oft nah am Objekt, z.B. mit Hilfe eines Kontext-Menüs. Beispiele für Anwendungen dieses Oberflächen-Typs sind Vektorgrafik-Programme, Textverarbeitungs-Systeme oder digitale Audio-Anwendungen. Die Arbeitsweise der Programme orientiert sich an der Werkzeug-Material-Metapher.

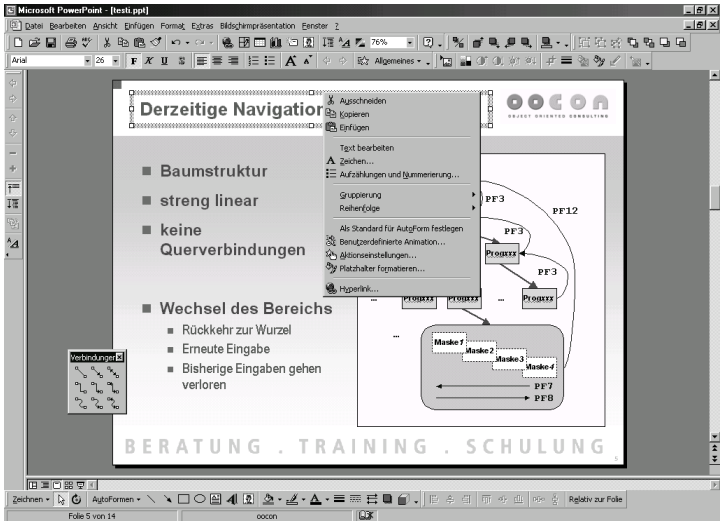

Abbildung 5 – Objektorientierte Oberfläche

Formularbasierte Oberfläche

Im Gegensatz zu den oben genannten Oberflächen orientieren sich formularbasierte Oberflächen[9] an einem fest vorgegebenen, sequentiellen Arbeitsablauf. Der Benutzer wird durch formularartige Eingabe-Masken geführt, welche einen Anwendungsfall abbilden.

Diese Arbeitsweise entspricht dem Verhalten von

- Host-Anwendungen:
  Maskeneingabe – Datenfreigabe – Maskenantwort ...
- Html-Internet-Anwendungen:
  Aufruf einer Seite – Aktion – Antwort mit neuer Seite ...
- „Wizards“ innerhalb einer Anwendung:
  Navigation nur mit „Vor“, „Zurück“ und „Abbrechen“ möglich.

Beispiele für derartige Oberflächen sind Internet-Buchläden oder Assistenten zum Erstellen von Software-Komponenten, die in vielen Software-Entwicklungsumgebungen vorhanden sind. Auch viele Sprachlernprogramme basieren auf dem Prinzip von Benutzer-Eingabe und anschließender Vorgabe des nächsten Arbeitsschrittes. Alle derartigen Anwendungen verwenden die Fabrik-Arbeitsmetapher.

[8] das Adjektiv „objektorientiert“ bezieht sich ausschliesslich auf die Arbeitsweise der zugehörigen Programme. Es hat nichts mit der Auswahl Programmiersprache zu tun, die für die Erstellung der Software verwendet werden. Zwar ist es in der Regel einfacher, objektorientierte Benutzeroberflächen mit einer objektorientierten Programmiersprache zu ertellen. Es ist jedoch ebenso mit strukturierten Programmiersprachen wie „C“ möglich.

[9] vlg. [ColKru]

Der Einsatz von formularbasierten Oberflächen ist immer dann vorteilhaft, wenn der Benutzer stark angeleitet werden soll und die Aufgaben formalisiert sind. Im Gegenzug führt eine derartige Benutzerschnittstelle zu einer geringen Flexibilität innerhalb der Anwendung. Bei größeren Aufgabenstellungen muss der Anwender die Reihenfolge und Inhalte der einzelnen Masken kennen, um seine Aufgabe bearbeiten zu können. Das System passt sich der Arbeitsweise der Anwender *nicht* an.

Abbildung 6 – Formularbasierte Oberfläche

Manchmal zwingen auch die eingeschränkten Ressourcen der Laufzeitumgebung zur Verwendung einer derartigen Oberfläche, obwohl sie für das Einsatzgebiet nicht passend ist. Dies kann z.B. bei Micro-Systemen wie Mobiltelefonen der Fall sein, welche nur über ein sehr kleines Display verfügen und keine externen Zeigegeräte wie eine Maus besitzen. In einem solchen Fall sollten Sie versuchen, die Flexibilität ihrer Benutzerschnittstelle durch Abkürzungen und Verzweigungsmöglichkeiten zu erhöhen, um zusätzliche Freiheitsgrade zu erreichen.

Wir haben Anwendungen mit *objektorientierten* Oberflächen gesehen, in denen Massenerfassungen von Belegen durchgeführt werden sollten. Dies ging völlig an den Anforderungen der Anwender vorbei, die sequentiell einen Belegstapel abarbeiteten und dabei ausschließlich die Tastatur benutzten, da sie „blind“ schrieben.

Erst eine völlig Umgestaltung dieses Anwendungsteiles auf einen formularbasierten Ansatz brachte letztlich die gewünschte Performance, Ergonomie und Benutzer-Akzeptanz.

7.3.3 Ergonomische Gestaltung

Bei der menschlichen Wahrnehmung und Informationsverarbeitung spielen Form und Struktur eine große Rolle. Visuelle Gestaltungsmittel haben gegenüber verbalen Beschreibungen den Vorteil, dass sie bei gleichem Platzbedarf

- eine weitaus höhere Informationsdichte besitzen,
- der assoziativen Verarbeitung entgegen kommen,
- Struktur und Semantik einfacher darstellen können.

Neues wird assoziativ verarbeitet und in das eigene Wissensnetz eingefügt. Überschaubare Einheiten können auf einen Blick erfasst werden. Bei der Gestaltung von Benutzerschnittstellen spielt deshalb die *7 plus/minus 2 - Regel* eine wichtige Rolle:

Stellen Sie 5 bis 9 Details innerhalb einer Gruppe dar.

Auswahl der Gestaltungsmittel

Bei der Auswahl eines Darstellungsmittels sollten Sie dessen implizite Bedeutungen bewusst einsetzen. Betrachten Sie die verschiedenen Darstellungen eines Teams in Abbildung 7:

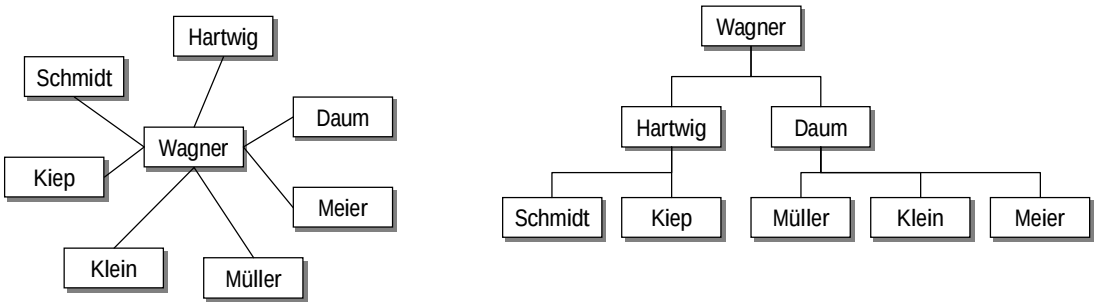

Abbildung 7 - Darstellung von Strukturen

Auf der linken Seite sind alle Mitarbeiter sternförmig um den Kollegen *Wagner* gruppiert. Dies wird mit gleichartigen Beziehungen assoziiert. Durch die Sonderrolle im Zentrum wird die Bedeutung von *Wagner* als Gruppenleiter hervorgehoben. Auf der rechten Seite wird die Gruppe hingegen in Form einer Hierarchie dargestellt. Auch hier tritt *Wagner* eindeutig als Kopf der Gruppe hervor. Darüber hinaus wird jedoch auch die Bedeutung von *Hartwig* und *Daum* als Teamleiter symbolisiert. Je höher eine Person in der Hierarchie angeordnet ist, desto größer ist ihre Bedeutung. Bei annähernd gleichen Platzbedarf transportiert die rechte Grafik zusätzliche Informationen.

Beim Entwurf der grafischen Benutzerschnittstelle ihres Systems sollten Sie folgende Heuristiken berücksichtigen:

- Stellen Sie die wichtigsten Informationen im oberen Teil ihrer Masken dar.
- Gruppieren Sie inhaltlich zusammengehörige Dinge nah beieinander.
- Richten Sie die einzelnen Widgets links- oder rechtsbündig an einem unsichtbaren Raster aus.
- Visualisieren Sie Strukturen durch Bäume, Netze etc.
- Ersetzen Sie verbale Beschreibungen durch Symbole, falls dies möglich ist (z.B. Stop-Zeichen für Fehler). Beachten Sie jedoch, dass Symbole oft nicht kulturübergreifend verständlich sind.

Kapitel Internationalisierung

- Verwenden Sie sinnvolle Eingabe-Vorbelegungen (Defaults). Unterlassen Sie dies jedoch, wenn es keinen „Favoriten“ gibt. Ein unpassender Defaultwert ist für den Anwender mit Mehrarbeit gegenüber einem leeren Eingabefeld verbunden[10].
- Lassen Sie sich bei der Gestaltung der Oberfläche von anderen Programmen, die für *ähnliche Aufgabenstellungen* gedacht sind, inspirieren[11].
- Folgen Sie den bewussten und unbewussten Erwartungen der Anwender. Dies macht die Benutzerschnittstelle durch Wiedererkennung intuitiv und leicht erlernbar.
- Überprüfen das GUI-Design so früh wie möglich an Hand von Anwender-Workshops.

Bei der detaillierten Gestaltung der Benutzerschnittstelle können ihnen spezielle GUI-Entwurfsmuster weiterhelfen[12].

Entwurfsmuster

Formular: Standardisierte Dateneingabe.

siehe auch *Schritt-für-Schritt*

Beispiele: Internet-Bestellung, Buchungssatz anlegen, Schadensanzeige

- Heben Sie Pflichteingaben von optionalen Feldern optisch hervor (z.B. durch die Hintergrundfarbe).
- Ausgabefelder müssen von Eingabefeldern klar zu unterscheiden sein.
- Machen Sie deutlich, welche Eingaben erwartet werden. Dies kann durch vorformatierte Felder oder Beispiele erfolgen.

[10] Erkennen des unpassenden Wertes, Löschen desselben und anschließende Eingabe der korrekten Daten

[11] viele Entwickler kopieren beim Oberflächendesign das, was ihnen bekannt und vertraut ist: Softwareentwicklungstools. Oftmals ist deren GUI-Design jedoch nicht für die Domäne Ihrer Anwendung geeignet.

[12] diese werden in der englischsprachigen Literatur oft auch als HCI-Design-Entwurfsmuster referenziert (Human-Computer Interface Design). Vlg. [Tidwell].

Verdichtete Information: Darstellung einer Vielzahl von *gleichartigen* und *gleich bedeutsamen* Daten. Der Anwender möchte sowohl den Überblick haben, als auch Detailinformationen sehen.

siehe auch
Tabelle, Hierarchie

Beispiele: Fahrplan, Landkarte, Organigramm

- Stellen Sie möglichst alle Informationen innerhalb *einer* Ansicht dar..
- Sparen Sie Platz durch Verwendung eines kleinen Fonts.
- Wählen Sie eine grafische Darstellung, die der Struktur der Informationen entspricht.
- Heben Sie wichtigere Informationen durch Farbe oder Fettdruck hervor.
- Setzen Sie Farbe und Schrifttypen zur Strukturierung ein. Der Anwender kann sich so schneller in der Informationsfülle orientieren.

Tabelle: Darstellung von verdichteten Daten in tabellarischer Form. Der Anwender möchte viele Informationen auf einmal sehen und Vergleiche anstellen. Ein einzelner Eintrag muss leicht aufzufinden sein.

Beispiele: Adressbuch, Stückliste, Suchergebnis

- Sortieren Sie die Daten nach logischen Kriterien.
- Erleichtern Sie das Auffinden einzelner Werte durch optische Anker (z.B. bei einer alphabetischen Sortierung durch Fettdruck für den ersten Eintrag eines neuen Buchstabens).
- Ordnen Sie die Spalten in absteigender Wichtigkeit von links nach rechts.

Hierarchie: Darstellung verdichteter, *hierarchisch geordneter* Daten. Zu den Detailinformationen ist das Wissen über die Informationsstruktur für den Anwender von großer Bedeutung. Es hilft ihm, Daten schnell wiederzufinden und Beziehungen zu erkennen.

Beispiele: Verzeichnisse (Dateien, Warengruppen etc.), Organigramm

- Stellen Sie die Daten in einer Baumstruktur dar.

Gruppe verwandter Dinge: Darstellung zusammenhängender Informationen (nach der 7+/-2-Regel). Kleine Informationseinheiten können vom Anwender schnell erkannt werden.

Beispiel: Adressblock auf einer Kunden-Maske

- Verwenden Sie visuelle Muster für die Anordnung der Gruppen (Symmetrie, Wiederholung etc.).
- Setzen die Gruppen durch breitere Zeilen- oder Spaltenabstände voneinander ab.
- Beschränken sie den Einsatz von Gruppierungsrahmen.

Expertensystem: Anwendung mit *langer Lebenszeit*, die von den Anwendern zur Bearbeitung *komplexer Aufgaben* verwendet wird. Der Benutzer will eine Vielzahl von Daten gleichzeitig sehen und bearbeiten. Längere Einarbeitungszeiten werden akzeptiert[13]. Die Anwendung muss den Anwender optimal bei seinen täglichen Aufgaben unterstützen.

Beispiel: Tabellenkalkulation, Bildbearbeitung, Audio-Schnittplatz

- Lassen Sie viel Platz für die Darstellung der Arbeitsgegenstände.
- Platzieren Sie Statusinformationen und Werkzeuge an den Rändern.
- Die Aufgaben müssen *performant* und mit möglichst wenigen Arbeitsschritten zu erledigen sein.
- Bieten Sie Möglichkeiten zur benutzerspezifischen Konfiguration.

Strukturelle Wiederholung: Darstellung und Navigation folgen stets dem gleichen visuellen und strukturellen Muster. Durch die permanente Wiederholung werden die Bedeutungen implizit erfasst und die Bedienung intuitiv.

Beispiele: Buch mit gleichartigem Layout aller Kapitel, Standard-Menüs und Standard-Dialoge in Windows-Programmen

- Verwenden Sie Symbole und Dialoge aus dem Betriebssystem-Standard, wann immer dies möglich ist.
- Bilden Sie gleichartige Aufgaben in gleichartigen Masken ab (einheitliches Maskenlayout, analoge Operationen und Navigation).
- Führen Sie Standards für Buttons und Kontext-Menüs ein (Kombination, Anordnung, Beschriftungen, Default-Button).
- Verwenden Sie gleiche Fonts und Farben zu Hervorhebung gleicher Bedeutung (Beschriftung, Erläuterung, Überschrift, Kontext der Bearbeitung etc.).

Details bei Bedarf: Darstellung oder Eingabe selten benötigter Detailinformationen.

Beispiele: Tooltips, Fußnote, Auswahl selbst definierter Farben, Optionen

- Stellen Sie selten benutzte Details hinter häufig verwendeten Informationen visuell zurück (z.B. Anordnung im unteren Maskenbereich, Einsatz kleinerer Fonts).
- Bei Platzmangel in der Oberfläche werden diese Informationen nur durch Benutzeraufforderung zugänglich gemacht (Kontext-Menü, Zusatz-Dialog).

[13] die Einarbeitungszeit ist minimal im Vergleich zur späteren produktiven Einsatzzeit der Anwendung.

Fortschrittsanzeige: Darstellung des *Fortschritts der aktuellen Operation*. Der Anwender möchte wissen, ob das System noch arbeitet. Er will die noch benötigte Zeit abschätzen. Sobald sich auf dem Bildschirm nichts mehr tut, ist der Benutzer beunruhigt, dass die Anwendung hängen könnte.

Beispiele: Datei-Download, Küchenwecker, Installations-Prozedur

- Sorgen Sie dafür, dass sich auf dem Bildschirm etwas verändert (Textausgabe, Fortschrittsbalken etc.).
- Geben Sie an, wie groß der Anteil der bereits erledigten Aufgaben ist und wie viel voraussichtlich noch zu tun ist.

Statusanzeige: Darstellung eines *Zustands*. Die Information ist für den Anwender nur gelegentlich interessant. Sie muss einfach zu finden sein, darf aber die Haupttätigkeit nicht stören.

Beispiele: Uhrzeit, Tachometer, Akku-Ladezustand, System-Ressourcen

- Verwenden Sie Symbole.
- Betonen Sie Wichtiges (z.B. durch Farbe).
- Stellen Sie die Information stets an der gleichen Stelle dar.
- Behalten Sie bei mehreren Statusinformationen immer die gleiche Anordnung bei.

Rückkehr an einen sicheren Ort: Rückkehr zu einen konsistenten *Ausgangspunkt*. Der Anwender kann sich in der Anwendung „verirrt“ haben.

Beispiele: *Home*-Button im Web-Browser, *Zurücksetzen*-Button in Eingabemasken, *Standard*-Button in Konfigurationsmasken

- Bieten Sie eine Rückkehrmöglickeit zum letzten konsistenten Zustand an (gesicherte Datei, aktuelle Konfiguration, Ausgangspunkt der Navigation etc.).

Schritt zurück: Rückkehr zum letzten Bearbeitungsschritt. Der Anwender hat den „Faden verloren“. Er möchte zu einem konsistenten Zustand zurückkehren, ohne die gesamte Bearbeitung abzubrechen.

Beispiele: In einem Buch eine Seite zurückblättern, *Undo*-Funktion, *Zurück*-Button einer Installationsroutine.

- Bieten Sie in sequentiellen Abläufen eine „Zurück“-Funktion an (formularbasierte Oberflächen).
- Die Möglichkeit der „Zurück“-Navigation erfordert in fast allen Fällen auch eine zugehörige „Vorwärts“-Navigation.

Schritt-für-Schritt: Anwender werden durch einen sequentiellen Arbeitslauf geführt.

Beispiele: Kochrezept, Geldautomat, Installationsroutine

- Verwenden Sie eine formularbasierte Oberfläche.
- Geben Sie in jedem Schritt klare Instruktionen.
- Beschränken Sie die Schrittanzahl auf maximal neun Schritte.
- Verwenden Sie bei mehr als sechs Schritten eine Fortschrittsanzeige.
- Ermöglichen Sie Vorwärts- und Rückwärts-Navigation.

Mehrfach-Aktion: Auf mehrere gleichartige Objekte soll die gleiche Operation ausgeführt werden. Der Anwender möchte die einzelnen Bearbeitungsschritte nicht für jedes einzelne Element wiederholen[14].

Beispiele: Löschen mehrere Dateien, Verschieben mehrerer Objekte

- Wenn eine Operation dies zulässt, so geben Sie die Möglichkeit zur Mehrfachauswahl von Objekten.

7.3.4 Weiterführende Literatur

[UiDes] bietet ein Fülle an Ressourcen zum Thema Benutzerschnittstelle.

[HalShm] ist eine umfangreiche Sammlung gelungener und abschreckender Beispiele zu den Themen GUI-Design und Benutzer-Interaktion.

[Tidwell] beschreibt eine Entwurfsmuster-Sprache für Benutzerschnittstellen.

[UiDes] www.uidesign.net

[HalShm] http://www.iarchitect.com/shame.htm

[Bienh2000] Patterns for human oriented information presentation, Diethelm Bienhaus, 2000, Proc. of EuroPLoP2000

[Tidwell] Common ground: A pattern language for Human-Computer Interface Design, Jenifer Tidwell, http://www.mit.edu/~jtidwell/common_ground_onefile.html

[14] Maschinen eignen sich weitaus besser für die stupide Wiederholung gleicher Tätigkeiten. Menschen ermüden dabei und arbeiten schließlich fehlerhaft.

7.4 Ablaufsteuerung grafischer Oberflächen

von Kerstin Dittert

Verwendet man grafische Benutzeroberflächen, so stellen sich im Rahmen des Architektur-Entwurfs im wesentlichen folgende Fragen:

- Wie interagieren die Benutzer mit dem System?
- Wer steuert die Navigation der Anwendung?
- Wer kontrolliert den Zustandswechsel?
- Wer verarbeitet Ereignisse?
- Welche fachlichen Komponenten versorgen die Benutzeroberfläche mit Inhalt?

Die Struktur der Arbeitsabläufe hat Auswirkungen auf die Art der GUI und die zugehörige Ablaufsteuerung. Einige Software-Systeme bedürfen einer starken Anleitung des Anwenders. Dies billigt dem Benutzer nur wenige Freiheitsgrade zu, andererseits werden aber auch die Fehlermöglichkeiten reduziert. Bekannte Beispiele für diesen Anwendungstyp sind die "Assistenten", die bei der Installation von PC- Software unterstützen.

Andere Systeme bieten dem Benutzer zu jedem Zeitpunkt viele Entscheidungsmöglichkeiten und große Freiheit bei der Auswahl von Operationen. Diese werden unmittelbar auf den bearbeiteten Daten ausgeführt, längere Transaktionskontexte gibt es kaum. Textverarbeitungssysteme sind ein gutes Beispiel für diese Art von Anwendungen.

Die Art und Weise, in der ein Benutzer mit einem Programm interagiert, lässt sich verschiedenen Arbeitsmetaphern zuordnen (vgl. Abschnitt 7.3.1). Diese Metaphern korrespondieren ihrerseits mit den Oberflächentypen.

Die Bestimmung zusammengehöriger Arbeitsschritte und die Zuordnung von Arbeitsmetaphern sind daher wichtig für frühe Architekturentscheidungen. Im weiteren Projektverlauf werden Benutzeroberfläche und Ablaufsteuerung darauf abgestimmt. Beide Komponenten sind für die Bedienbarkeit einer Anwendung von entscheidender Bedeutung.

Diese Architekturbausteine sind zentrale Erfolgsfaktoren für die Akzeptanz des Systems. Sie besitzen entscheidenden Anteil am Projekterfolg.

7.4.1 Lösungskonzepte

Die folgenden Designvorschläge geben Ihnen Entwurfsmuster und Heuristiken für die effektive Umsetzung der Arbeitsmetaphern an die Hand.

Ereignisgesteuert: XX

Anschließend stellen wir das *Model-View-Controller* Entwurfsmuster und zugehörige Varianten vor. Diese Muster gelten als Standard für grafische Benutzeroberflächen ereignisgesteuerter Systeme.

Formularbasierte Oberfläche (Fabrik)

Diese Benutzerschnittstellen sind für einen fest vorgegebenen Arbeitsablauf vorgesehen. Ihre Masken entsprechen im wesentlichen Formularen, die ausgefüllt und vom Anwender bestätigt werden. Als Antwort liefert das System eine neue Maske zurück, die entweder den nächsten Arbeitsschritt formalisiert oder zur Fehlerkorrektur auffordert.

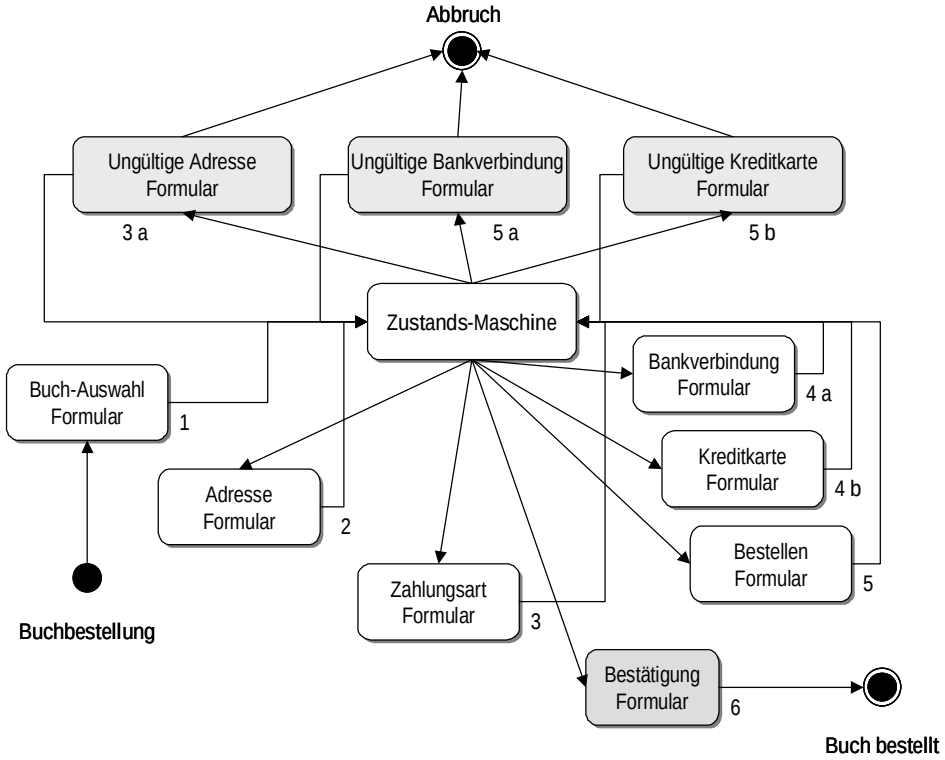

Abbildung 8 – Zustandsautomat für formularbasierte Oberfläche

Formularbasierte Anwendungen sind ein System aufeinanderfolgender Zustände, die durch Masken repräsentiert werden. Im Zentrum der Ablaufsteuerung steht deshalb ein Zustandsautomat. Abbildung 8 zeigt den Zustandsautomaten im Zusammenspiel mit verschiedenen Fachmasken, die eine Buchbestellung abbilden. Jede Maske sendet ihren Inhalt an den Zustandsautomaten. Dieser stellt fest, ob eine gültige Eingabe vorliegt oder nicht und ruft anschlie-

ßend die folgende Maske auf. Zustandsübergänge werden durch fachliche Ereignisse gesteuert.

Für den Entwurf des Zustandsautomaten bieten sich zwei Alternativen an:

- Übergänge werden durch externe Konfigurationstabellen gesteuert.
- Übergänge werden programmiert (im "Quellcode verdrahtet").

Die erste Möglichkeit bietet größtmögliche Flexibilität, bei großen Systemen wird die Konfiguration jedoch schnell unübersichtlich. Konfigurationsfehler treten nur als Laufzeitfehler auf und sind schwer zu testen.

Entscheiden Sie sich für die zweite Alternative, so wird das System robuster und durch bessere Verständlichkeit leichter wartbar. Fehlerhafte Zustandsübergänge werden schon bei der Übersetzung des Programms erkannt. Andererseits erfordert jede Änderung der Ablaufsteuerung oder Erweiterung der Anwendung eine Neukompilierung des Systems. Damit ist meist auch eine erneute Anwendungsverteilung nötig.

Da eine Neukompilierung bei Änderungen des Systems aber sowieso meist erforderlich ist (in der Regel werden Masken geändert oder neu hinzugefügt), erweisen sich die programmierte Zustände häufig als vorteilhafter.

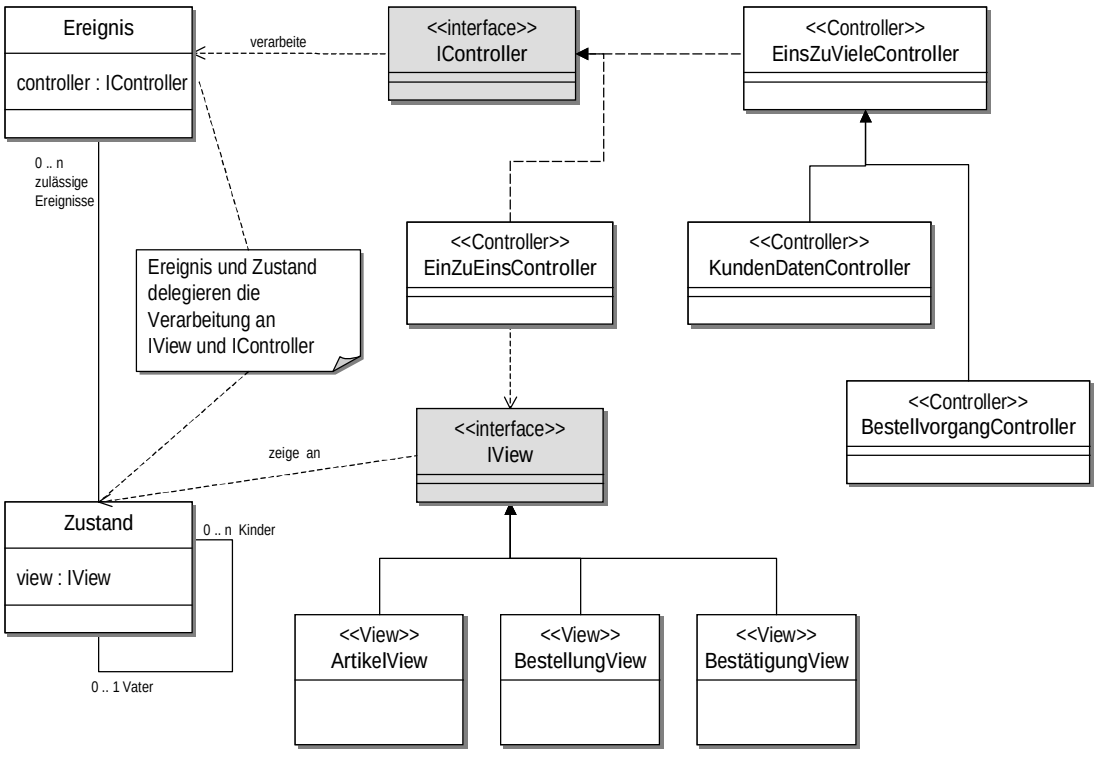

Abbildung 9 – Entwurf eines Zustandsautomaten

Egal auf welche Weise die Zustandsübergänge im System implementiert sind: Der Zustandsautomat sollte so übersichtlich strukturiert sein, dass Anpassungen einfach möglich sind. Da sich geänderte oder neue Anforderungen von

Anwendern häufig auf die Navigation beziehen, sind Änderungen hier sehr wahrscheinlich.

Die verschiedenen Masken sollten Sie nach ihrem dynamischen Verhalten kategorisieren. Unterschiede gibt es zum Beispiel zwischen Fehlermasken, Eingabemasken und Suchmasken. Für diese Kategorien können Abstraktionen eingeführt werden, von denen konkreten Ausprägungen (etwa eine Adressen-Eingabe-Maske) abgeleitet werden. Arbeiten Sie mit einer objektorientierten Programmiersprache, so werden die Maskenkategorien als abstrakte Basisklassen umgesetzt. Konkrete Fachmasken spezialisieren diese Basisklassen.

Objektorientierte Oberfläche (Werkzeug-Material)

Für den Entwurf einer objektorientierten Oberfläche müssen zuerst folgende Fragen beantwortet werden:

- Welches Material (=Arbeitsgegenstand) wird bearbeitet?
- Welche Werkzeuge (=Arbeitsmittel) werden dafür eingesetzt?
- Welche Verwendungszusammenhänge (=Tätigkeiten) bestehen zwischen Werkzeug und Material?
- Welche Umgebung enthält die Werkzeuge und Materialien?

Hierbei unterscheidet man zuerst die Werkzeug- und Materialeigenschaften der Domäne.

Das *Material* ist Gegenstand der Tätigkeit, es wird bearbeitet und kann mit anderen Materialien kombiniert werden.

Mit Hilfe von *Werkzeugen* können die Materialien bearbeitet werden. Sie haben großen Einfluss auf die Qualität der Arbeitsergebnisse. Werden Ergonomie und Effizienz der Werkzeuge vernachlässigt, so leidet die Akzeptanz der Anwendung beträchtlich. Bei der Benutzung des Werkzeugs sollte stets das Material im Vordergrund stehen und nicht das Werkzeug selbst! Jedes Werkzeug beinhaltet ein funktionale und eine interaktive Subkomponente. Der funktionale Anteil beinhaltet alle Operationen, die sich auf das Material auswirken. Der interaktive Teil des Werkzeugs bestimmt hingegen die Handhabung durch den Anwender.

Haben Sie Werkzeuge und Material bestimmt, so können Sie deren mögliche Kombinationen als *Verwendungszusammenhänge* (auch *Aspekte* genannt) beschreiben.

Die *Umgebung* (zum Beispiel ein Schreibtisch) dient der Strukturierung des Arbeitsplatz. Sie beinhaltet Werkzeuge und Materialien und sorgt für Konsistenz zwischen den verschiedenen Materialien.

Beispiel: Wollen Sie in einem Vektorgrafik-Programm ein Objekt auf eine bestimmte Position verschieben, so können Sie dies mit der Maus tun, und sich die neuen Koordinaten anzeigen lassen. Es werden einige Versuche nötig sein, bis Sie den richtigen Punkt „treffen“. Sie müssen sich also stark auf die *Interaktionskomponente*, d.h. die Bedienung des Werkzeuges konzentrieren. Ein geeigneteres Werkzeug bietet Ihnen Eingabe-Felder für die neuen Koordinaten an. Hier können Sie sich ganz auf das Material (verschiebbares Objekt) und seine Bearbeitung (neue Koordinaten) konzentrieren und gelangen schnell zum gewünschten Ergebnis.

Beim Design einer objektorientierten Oberfläche sollten Sie folgende Entwurfsregeln berücksichtigen[15]:

- Entwerfen Sie die Werkzeuge durch Dekomposition: Jeder Aspekt entspricht einem eigenen Werkzeug.
- Jedes Werkzeug setzt sich aus einer Funktions- und einer Interaktionskomponente zusammen
- Werkzeuge können gekoppelt werden.
- Das Umgebungsobjekt übernimmt die Aufgaben eines Materialverwalters und Werkzeugkoordinators.

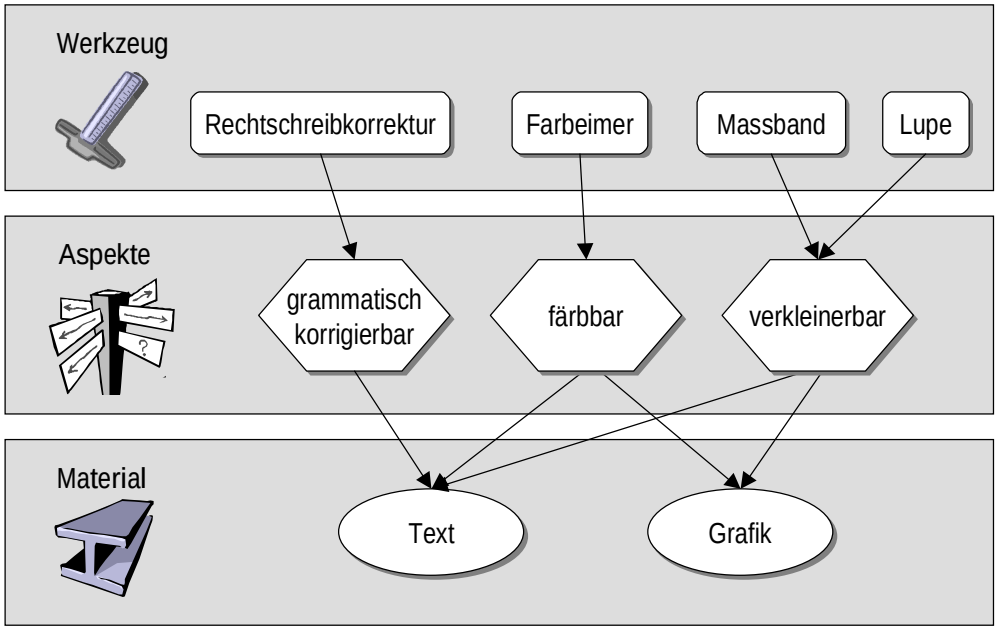

Abbildung 10 – Elemente der objektorientierten Oberfläche

[15] Vorschläge für das Feindesign der Komponenten finden Sie in [Riehle]

Model-View-Controller (MVC)

Dieses Entwurfsmuster entstand während der Verbreitung ereignisgesteuerter grafischer Benutzeroberflächen. Viele objektorientierte Entwicklungsumgebungen wurden mit grafischen GUI-Editoren ausgestattet, in denen die Verarbeitungslogik direkt an die Steuerungselemente angebunden werden konnte. Dieser Ansatz erschien naheliegend, da im Entwicklungsprozess meist zuerst mit dem Maskenentwurf begonnen wurde. Als Folge waren jedoch die Ablaufsteuerung und die Datenhaltung komplett in den Masken vereint.

Dieser Ansatz ist geeignet für kleine Systeme, die schnell entwickelt werden sollen. Die starke Kopplung von Logik und Darstellung bringt jedoch etliche Nachteile mit sich:

- Sollen mehrere Masken den gleichen Datenbestand in unterschiedlichen Sichten darstellen, so müssen die Daten redundant gehalten werden. Dies führt zu Performanz- und Konsistenzproblemen.
- Die Vermengung von Präsentation, Datenhaltung und Ablaufsteuerung macht den Programmcode schwer verständlich. Bei späteren Wartungsarbeiten oder Erweiterungen sind Seiteneffekte wahrscheinlich[16].
- Neue Masken müssen stets von Grund auf neu programmiert werden. Eine Wiederverwertung bestehender Komponenten ist kaum möglich.
- Eine Portierung der Anwendung (etwa auf eine andere Datenbank) kann sehr aufwändig werden. Im Extremfall muss fast jede Klasse angepasst werden.

Als Antwort auf diese Probleme wurde das MVC-Muster entworfen, welches die GUI- und Präsentationsschicht strukturiert. Es entkoppelt die Darstellung (*View*) von der Datenhaltung (*Model*) und der Ablaufsteuerung (*Controller*) im System. Viele GUI-Frameworks basieren auf diesem Muster, beispielsweise die Java-Swing-Bibliothek.

Die Anwendung dieses Musters führt durch die Trennung der Verantwortlichkeiten zu einer größeren Flexibilität und besseren Erweiterbarkeit des Systems. Andererseits ist mit einem erhöhten Entwurfs- und Realisierungsaufwand zu rechnen. Dieser Mehraufwand zahlt sich bei Anwendungen aus, die:

- Auf mehrere Rechner (z.B. Client-Server) verteilt sind.
- Eine größere Anzahl von Masken besitzen (> 20 Masken).
- Eine lange Lebenszeit haben.
- Portierbar sein sollen.
- Mehrere gleichzeitige Sichten auf einen Datenausschnitt haben.

Die *Views* der Anwendung stellen einen Zustand der Anwendung dar. Im allgemeinen entsprechen sie den Fachmasken. Sie kommunizieren mit den zuge-

[16] Denken Sie an die Regel von Murphy: Wenn etwas schief gehen kann, wird es auch schief gehen.

hörigen Komponenten der GUI-Schicht (den Widgets), aus denen sie zusammengesetzt sind. Ein View nimmt Benutzereingaben (etwa per Tastatur oder Maus) an und leitet diese als fachliches Ereignis an den Controller weiter. Jeder View kann sein zugehöriges Modell über den aktuellen Status befragen, um die Darstellung zu aktualisieren. Ein View kann zusätzlich auch eine funktionale Komponente besitzen oder sogar ausschließlich aus ihr bestehen. Ein Beispiel hierfür sind Internet-Anwendungen, die ihre grafische Oberfläche mit Hilfe von Servlets aufbereiten: Die Views entsprechen den Servlets, welche in der GUI-Schicht mit Hilfe von Html-Elementen dargestellt werden.

Die *Controller* sind für die Ablaufsteuerung der Anwendung zuständig, d.h. sie sind Zustandsmaschinen. Sie setzen Ereignisse in Domänen-Operationen um und bestimmen den Folge-Zustand des Systems. Bei einer Zustandsänderung benachrichtigen sie die Modelle.

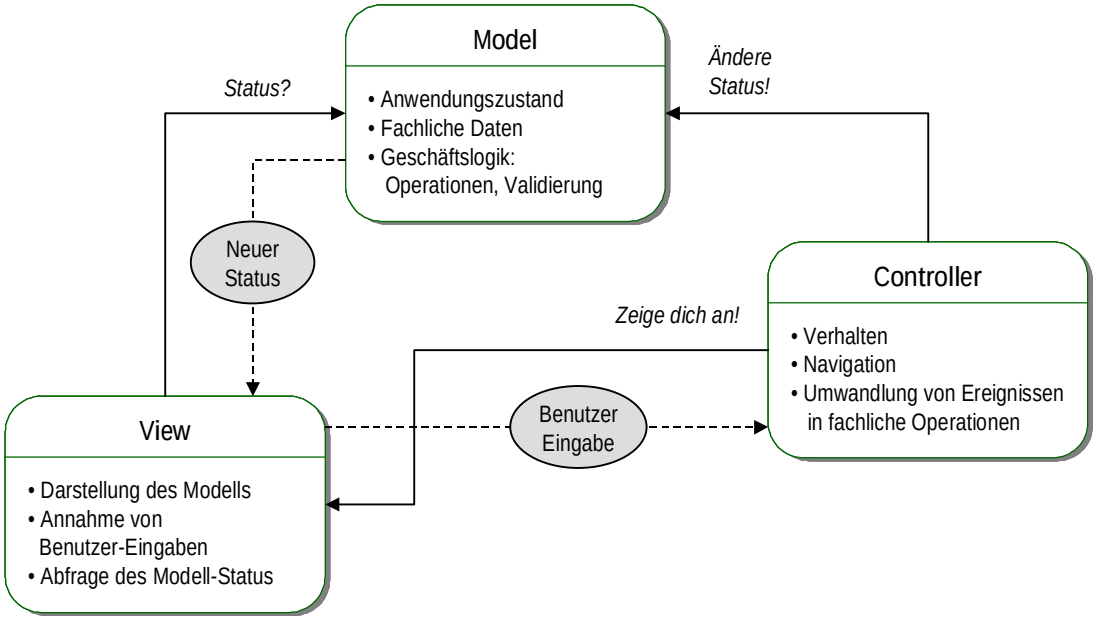

Abbildung 11 – Model-View-Controller (MVC)

Die *Modelle* beinhalten die fachlichen Informationen und strukturieren sie in einer Art und Weise, die für die Views passend ist. Sie stellen also keinen direkten Ausschnitt des Domänen-Modells dar, sondern nur die für die zugehörigen Masken relevanten Teile. Ein gutes Beispiel hierfür sind Modelle, die Suchergebnis-Masken mit Inhalt versorgen. In der Regel werden nicht alle Attribute der gefundenen Objekte angezeigt. Aus Performanz-Gründen wird der Modell-Ausschnitt deshalb häufig auf die tatsächlich dargestellten Attribute beschränkt.

Views und Controller gehören zur Präsentationsschicht, die Modelle hingegen zur Fachdomäne.

Meistens wird das Muster so angewendet, dass zu jeder Maske ein Tripel *Model-View-Controller* existiert. Varianten können jedoch sinnvoll sein: Falls verschiedene Masken (Views) Ausschnitte des gleichen Datenbestandes zeigen, so müssen sie über ein gemeinsames Modell verfügen. Dies gewährleistet die Konsistenz der dargestellten Daten.

Serverseitiger MVC (Thin-Client)

Diese Variante des MVC-Musters kann gut am Beispiel einer Internet-Anwendung mit einem Web-Browser als Client erläutert werden. Der Browser kommuniziert mit dem Server über das http-Protokoll. Auf dem Client existiert kein ausführbarer Programmcode, wie etwa Java oder Javascript. Die GUI-Schicht besteht ausschliesslich aus den Html-Komponenten des Browsers.

Formularbasierte Oberfläche Kapitel 7.3.2

Der Browser ist ein klassisches Beispiel einer formularbasierten Benutzeroberfläche: Gibt der Benutzer in einer Maske etwas ein, so wird eine Anfrage an den Server gesendet. Dieser antwortet mit einer neuen Maske.

Der Client kann einen einfachen Status halten (z.B. durch Cookies oder verborgene Html-Felder), er kann seinen Status aber auch unerwartet verändern. Dies kann z.B. durch Vor- oder Zurück-Navigation im Browser erfolgen.

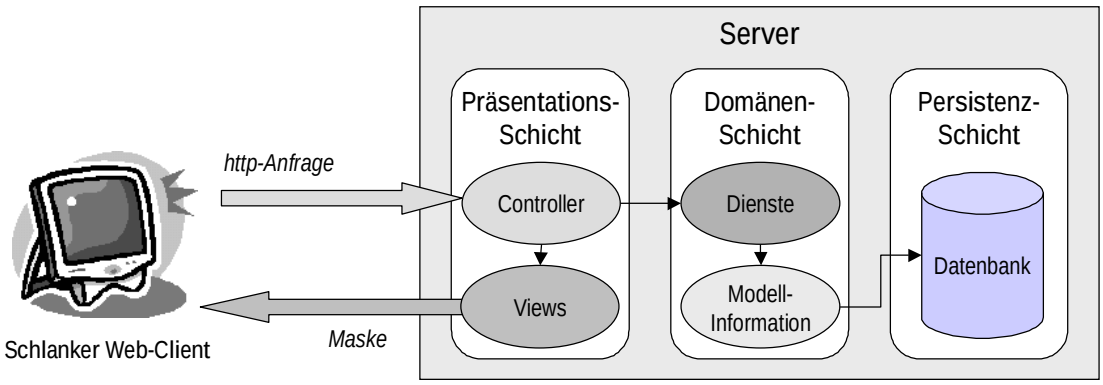

Abbildung 12 – Schlanker MVC

Beim Entwurf der Präsentationsschicht müssen zunächst atomare Arbeitsschritte und logische Transaktionen identifiziert werden. Jeder atomare Arbeitsschritt entspricht einem Zustand, d.h. einer View. Eine Transaktion besteht in der Regel aus mehren atomaren Arbeitsschritten, welche als http-Request übermittelt werden. Da die Anwendung von mehreren Benutzern gleichzeitig verwendet werden kann, müssen innerhalb der Transaktionen Konsistenz und Isolation gewährleistet sein. Dies erfordert eine deterministische Präsentationsschicht.

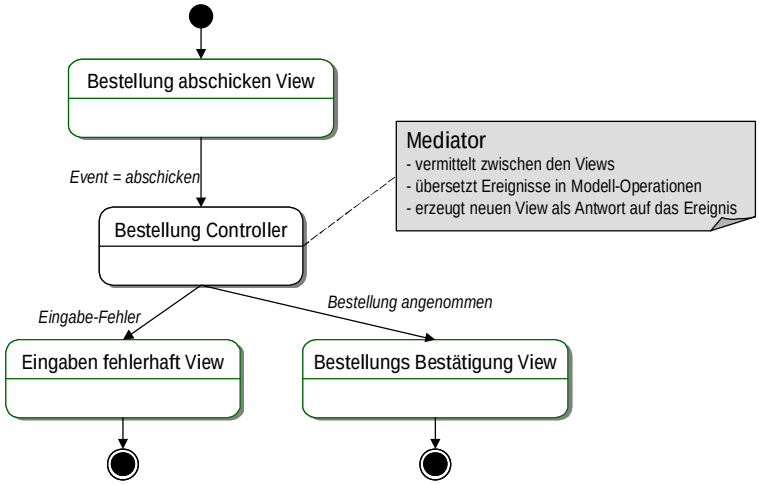

Abbildung 13 – Controller eines formularbasierten Systems

Für den Entwurf der Präsentationsschicht können die Transaktionen mit Hilfe von UML-Zustandsdiagrammen modelliert werden.

- Kern der Präsentationsschicht sind mehrere Zustandsmaschinen.
- Sie folgen dem Entwurf aus Abschnitt 7.3.2 (Formularbasierte Oberfläche).
- Sie arbeiten nach dem Mediator-Entwurfsmuster als Vermittler zwischen den Views und den Modellen.
- Jede logische Transaktion wird durch einen Controller gesteuert.
- Das Zusammenwirken dieser elementaren Controller kann durch einen übergeordneten Controller gesteuert werden.

Im Unterschied zum „klassischen“ MVC-Muster, werden die Views nicht vom Controller benachrichtigt, da jede Maske auf Anfrage neu erzeugt wird. Dies minimiert den Kommunikationsaufwand zwischen Client und Server und steigert so die Performanz des Systems beträchtlich.

MVC-Hierarchie

Ist eine Oberfläche sehr komplex, wie z.B. bei objektorientierten Benutzerschnittstellen, so muss das MVC-Muster verfeinert werden. Die einzelnen Views setzen sich meist aus kleineren fachlichen Subkomponenten zusammen, welche eigene Verantwortlichkeiten im Bezug auf Darstellung, Ablauf und Fachlogik haben. Beispiele hierfür sind ein Hauptfenster der Anwendung, mit Statuszeile und Menüs. Auch Fachmasken können aus wiederverwendbaren Komponenten für die Bearbeitung fachlich unteilbarer Einheiten (z.B. Adresse oder Bankverbindung) zusammengesetzt sein.

Diese Hierarchie der GUI-Komponenten kann in eine korrespondierende Struktur von MVC-Tripeln umgesetzt werden.

Für jede GUI-Subkomponente (etwa Statusbar oder Menü) gibt es ein MVC-Tripel, welches die Darstellung und den Zustandswechsel steuert. Gleichzeitig wird die Vater-Kind-Beziehung der GUI-Widgets auf die zugehörigen MVC-Controller übertragen. So ist etwa der Controller des Hauptfensters der Vater des Status-Zeilen-Controllers und des Menü-Controllers. Die Controller bilden damit eine Kette gemäß dem Zuständigkeitsketten-Muster, welches eine Aufgaben-Delegation erlaubt. Dies kann genutzt werden, um die Verantwortung an Subkomponenten oder übergeordnete Komponenten weiterzureichen. Der Controller des Hauptfensters bildet die Wurzel der Controller-Hierarchie. Er bearbeitet alle Anfragen, die keines seiner Kinder bearbeitet hat.

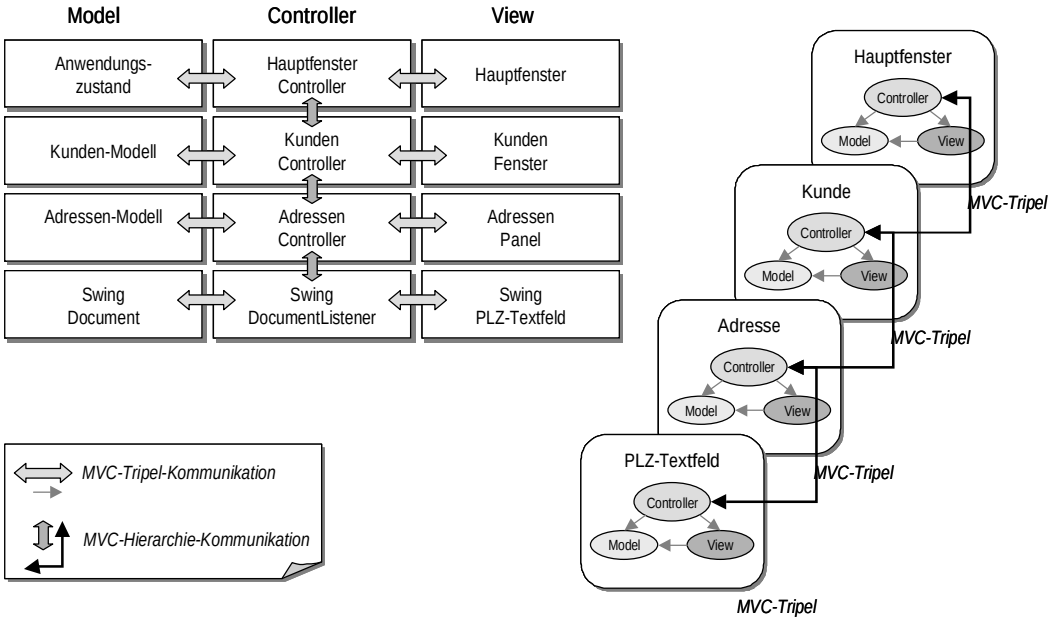

Abbildung 14 – MVC-Hierarchie

Die Kommunikation erfolgt innerhalb der MVC-Tripel gemäß dem MVC-Paradigma. Zusätzlich erfolgt eine gleichartige Kommunikation innerhalb der MVC-Hierarchie. Dies lockert die Komponentenkopplung und erleichtert Änderungen im System. Subkomponenten werden auf Grund standardisierter Schnittstellen leicht austauschbar.

Nachteil eines solchen Entwurfs ist der Aufwand für Design und Implementierung. Durch die starke Entkopplung kann sogar die Übersicht über den logischen Fluss der Anwendung verloren gehen. Darüber hinaus können Performanz-Einbussen durch eine Vielzahl von Indirektionen auftreten. Auch hier gilt es also Augenmass zu bewahren, und die Anzahl der MVC-Tripel auf ein nützliches Mass zu beschränken.

- Nicht jeder einzelne Button benötigt sein eigenes Modell und einen dazugehörigen Controller!

- Als Faustregel können Sie davon ausgehen, dass jede Maske einen eigenen Controller benötigt.
- Zusätzliche Controller könnten erforderlich werden, wenn
  - eine Maske wiederverwendbare Subkomponenten besitzt,
  - mehre Maske gleichzeitig dieselben Modellinformationen darstellen,
  - einzelne Subkomponenten eine höhere Aktualisierungsrate als der Rest der Maske besitzen (z.B. Statusbar, Statuszeile).

Zeigen mehrere Views Ausschnitte aus dem gleichen Fachkomplex, so können diese Masken über ein Modell mit zugehörigen Submodellen synchronisiert werden. Ein Controller steuert die Kommunikation zwischen den Teilmodellen und den Views. Zur Benachrichtigung der abhängigen Modelle werden spezielle Daten-Ereignisse definiert, welche über den Vater-Controller an dessen Kinder-Controller und die zugehörigen Modelle verteilt werden.

Abbildung 15 zeigt einen derartigen Entwurf. Dabei wird deutlich, dass auch das Tripel-Paradigma nicht immer beibehalten werden muss. Im Beispiel existiert zum allgemeinen Adressmodell kein View. Statt dessen werden Adressen entweder als einzelne Adresse innerhalb des Kunden-Views bearbeitet oder aber in einer Adressliste dargestellt. Beide View-spezifischen Modelle benötigen nur einen Ausschnitt des gesamten Adressmodells. Diese Teilmodelle kommunizieren mit dem gesamten Adressmodell über den Adressen-Controller, welcher Daten-Ereignisse empfängt und an seiner Kinder-Controller weiterleitet.

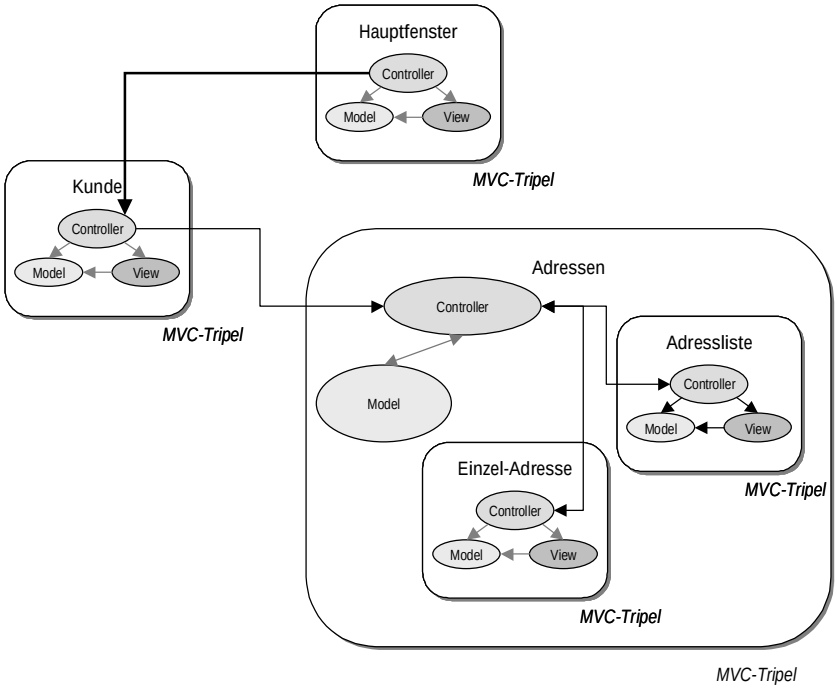

Abbildung 15 – Strukturierung einer MVC-Hierarchie

7.4.2 Konsequenzen

Model-View-Controller

Das Entwurfsmuster bietet durch die Entkopplung der Verantwortlichkeiten viele Vorteile. Durch den erhöhten Kommunikationsaufwand zwischen den Komponenten ergeben sich jedoch auch neue Randbedingungen für die Systemarchitektur:

- In verteilten Systemen müssen die Benachrichtigungen des Modells und die Anfragen an das Modell gut ausbalanciert werden, um unnötige Netzbelastungen zu vermeiden. So sollten Sie die Attribute eines Modells durch Proxy-Objekte oder Value-Objekte *en-bloc* abfragen, statt einzelne getXY()-Methodenaufrufe zu nutzen.
- Controller und Modell sind eng gekoppelt. Falls das Modell ständig im Fluss ist, wird es schwierig den Controller zu entwerfen. Abhilfe kann hier (bei einer asynchronen Verarbeitung ohne Rückgabewerte[17]) das Befehlsmuster (*Command*) schaffen, welches einen Befehl von der konkreten Ausführung entkoppelt. In allen übrigen Fällen können Controller- und Modellentwürfe nur parallel verfeinert werden. Siehe hierzu [Gamma95].
- Viele Entwicklungsumgebungen generieren die Controller für technische Basis-Ereignisse direkt in die View-Klassen. Das widerspricht zwar der „reinen MVC-Lehre“, sollte aber dennoch im Entwurf berücksichtigt werden. Der Versuch, einem derartigen Framework das MVC-Pattern quasi überzustülpen führt zu einer Unzahl von Klassen oder Prozeduren, welche wiederum den Code nahezu unlesbar machen. Darüber hinaus resultiert dieser Ansatz häufig in einer schlechten Performance, da zahllose Indirektionen durchlaufen werden müssen. In einem solchen Fall empfiehlt es sich, die grundlegenden Design-Pattern des Framework aufzugreifen und gegebenenfalls durch Adapter zu erweitern.

7.4.3 Interaktionen

Anwendungen mit grafischen Benutzeroberflächen haben spezielle Anforderungen an die *Ereignisbehandlung*, *Fehlerbehandlung* und *Verteilung*:

Die *Ereignisbehandlung* muss

- neben den technischen GUI Ereignissen auch die fachlichen Ereignisse verwalten können,
- über Client und Server verteilt funktionieren.

[17] für Anfragen, welche Ergebnisse zurückliefern, ist das Muster meist nicht geeignet. Die Kommunikation muss dann vielfach wieder künstlich synchronisiert werden.

Die *Fehleraufbereitung* für den Anwender sollte sich harmonisch in den gewählten Oberflächentyp einfügen:

- strukturierend und anleitend im Fall von formularbasierten Oberflächen,
- unterstützend bei objektorientierten Oberflächen, gegebenenfalls auch vom Anwender abschaltbar oder konfigurierbar.

Bei der Konzeption der *Verteilung* sollten schlanke Modell-Strukturen entworfen werden, welche die Informationen aus der Fachdomäne performant an die Präsentationsschicht verteilen.

7.4.4 Weiterführende Literatur

[Riehle] beschreibt den Entwurf objektorientierter Oberflächen.

[ColKru] stellen Entwurfsmuster für Softwaresysteme mit formularbasierten Oberflächen vor.

[Hmvc] beschreiben den Komponentenentwurf eines Fat-Client mit einer hierarchischen MVC-Struktur.

[Buschmann96] führt das MVC Muster aus.

[Riehle] Entwurfsmuster für Softwarewerkzeuge, Dirk Riehle, Addison Wesley 1997

[ColKru] Form-Based User Interfaces – The Architectural Patterns, Jens Coldewey, Ingolf Krüger, Proceedings of the 2nd European Conference on Pattern; Bad Irsee 1997, http://www.coldewey.com

[Col001] User Interface Software, Jens Coldewey, Conference on the Pattern Languages of Programming, 1998, Allerton Park, Illinois http://www.coldewey.com

[SrvMvc] Server-side MVC Architecture, uidesign.net http://www.uidesign.net/1999/papers/webmvc_part1.html http://www.uidesign.net/1999/papers/webmvc_part2.html

[Hmvc] HVMC: The layered pattern for developing strong client tiers, Jason Cai, Ranjit Kapila, Gauray Pal, http://www.javaworld.com