RWTH Aachen University
Media Computing Group
Prof. Dr. Jan Borchers

Proseminar Human-Computer Interaction
SS 2007

Implementation Support

Tobias Quix

Esther Schichler

Inhaltsverzeichnis

1 Einleitung

Als Grundlage für unsere Ausarbeitung diente das achte Kapitel, *Implementation Support*, des Buches *Human-Computer Interaction*[1].
Implementation Support befasst sich grundsätzlich weniger mit einer abstrakten Sicht auf Design und Analyse interaktiver Systeme, sondern zeigt Werkzeuge auf, die zur Implementierung eines solchen Systems benötigt werden. Somit wird Schritt für Schritt die Antwort auf folgende Frage erarbeitet: „Wie programmiert man eine interaktive Anwendung und welche Hilfsmittel stehen dazu zur Verfügung?“
Die hier vorgestellten Werkzeuge und Techniken zur Implementierung eines interaktiven Systems erlauben es dem Programmierer, die gewünschten Spezifikationen in Programmcode umzuseten (zum Beispiel Lesen von Eingaben verschiedener Eingabegeräte und die (graphische) Darstellung auf dem Display) und bestimmte Interaktionsobjekte, die auch dem Benutzer später zur Verfügung stehen, direkt zu entwickeln.

2 Fenstertechniksysteme (Window Systems)

Ein Fenstersystem stellt eine Graphische Benutzeroberfläche (GUI) dar. Im Vordergrund steht die Verwaltung von mehreren Fenstern.
Aus der Sicht des Programmierers werden durch das Fenstersystem graphische Basisfunktionen implementiert. Somit können beispielsweise Schriftzeichen dargestellt oder auch das Zeichnen von Linien, Kurven oder Pixelgraphiken ermöglicht werden.
Das Fenstersystem erleichtert dem Benutzer die Handhabung, indem es gestattet mit mehreren Programmen gleichzeitig arbeiten zu können, wodurch dem Anwender eine enorme Flexibilität zugesprochen wird. Dies wird umgesetzt, indem jedes Programm durch einen oder oftmals auch mehrere separate Bereiche des Bildschirms, den Fenstern, aufgeführt wird. Diese sind meist rechteckiger Form und können mit einem Zeigegerät, der Maus beispielsweise, frei bewegt werden, wobei sie auch überlappen dürfen.

2.1 Das Industry-Standard X Window System

Es gibt einige Fenstersysteme, wie das X Window System zum Beispiel, die erweiterte Fähigkeiten wie zum Beispiel die Netzwerktransparenz haben. Durch diese wird es dem Benutzer gestattet, die graphischen Oberflächen einer Anwendung auf einem anderen Computer darzustellen.
Das Industry-Standard X Window System ist ein klassisches Beispiel eines Fenstersystems. Das X Protokoll definiert eine Client-Server Architektur zwischen der Anwendung und seiner Darstellung. Die Anwendung als Client besagt was dargestellt werden soll und der Server spezifiziert die Art der Darstellung. Das Industry-Standard X Window System wurde Mitte der 80er Jahre am Institut von Massachusetts für Technologie (MIT) entwickelt und ist bekannt unter dem Namen X11 oder einfach nur X[2].

[1] [1] Human-Computer Interaction, Dix, Finlay, Abowd, Beale, Pearson-Verlag, 3rd Edition, 2004
[2] Quelle: [1], Seite 295, Abb. 8.3

Einer der wohl bedeutensten Unterschiede zu anderen Fenstertechniksystemen ist die Tatsache, dass der X-Server auf dem lokalen Arbeitsplatzrechner laufen kann, während der X-Client ebenfalls dort, aber auch auf einem entfernten Rechner ausgeführt werden kann. Durch diese Eigenschaft geprägt wird X als Standard angesehen und wird noch unabhängiger.
Unter einem X-Client versteht man dabei jede Art von graphischer Anwendung, die einen X-Server zur Darstellung benötigt. Die eigentlichen X-Clients arbeiten mit einem Window Manager zusammen. Der X-Server ist ein Programm, welches zur Steuerung des Displays dient. Ein Display beinhaltet dabei eine Zusammenstellung von Ein- und Ausgabegeräten (Tastatur, Maus, Bildschirm).
Der Fenstermanager (Window Manager) ist ein separater Client, welcher das Aussehen und Verhalten des Fenstersystems von X bestimmt. Der Fenstermanager entscheided, wie der Benutzer seine Sicht auf die Eingabe von einer Anwendung zur anderen ändern kann. So kann man ein Fenster als das Aktive wählen und somit alle nachfolgenden unter dessen Regie stellen. Weiterhin wird die Auswahl geboten, ob die sichtbaren Bildschirmfenster überlappen sollen oder nicht. Die Gestalt der Programme jedoch verantwortet der Client nicht. Diese Aufgabe übernimmt meistens das sogenannte Toolkit, das später genauer erläutert wird.

3 Programmierung einer interaktiven Anwendung

Eine interaktive Anwendung entspricht in etwa dem Client in der eingangs erwähnten Client-Server Architektur.
In der Regel ist eine solche Anwendung benutzergesteuert, da die gerade ausgeführte Aktion direkt durch die letzte Eingabe des benutzers gesteuert wird. Im Folgenden werden zwei Programmierparadigmen vorgestellt, die zur Organisation des Kontrollflusses in einer interaktiven Anwendung benutzt werden können.

3.1 Read-Evaluation Loop Paradigma

Das erste Paradigma ist die sogenannte Read-Evaluation Loop. Diese Schleife befindet sich direkt in dem Programm der Anwendung. In den Anfängen wurde ein Macintosh aussschließlich nach diesem Paradigmga programmiert, während heutzutage jedoch auch andere Paradigmen Anwendung finden. Der Server dient dabei als Brücke zwischen dem Gerät zur Benutzereingabe und der Client-Anwendung. In dieser Funktion bündelt der Server die gesamten Benutzereingaben, strukturiert diese und sendet sie als Befehle an den Client, der wiederum jeden ihm zugeteilten Befehl liest und verarbeitet und so das gesamte, für die Anwendung spezifische Verhalten bestimmt.

Read-Evaluation Loop in Pseudocode[3]

```
repeat
   read-event(myevent)
     case  myevent.type
           type_1 :
                  do type_1 processing
           type_2 :
                  do type_2 processing
           .
           .
           .
           type_n :
                  do type_n processing
     end case
 end repeat
```

Abbildung 3.1: Pseudocode - Beispiel

Der Nachteil hierbei ist, dass der Programmierer auch wirklich alle Befehle bei der Implementierung berücksichtigen muss; was mitunter sehr lästig sein kann. Toolkits (wie zum Beispiel MacApp von Macintosh), die wir im folgenden Teil noch vertiefen werden, unterstützen den Programmierer aber bei dieser Aufgabe.

3.2 Notification-Based Paradigma

Anders als die eingangs erwähnte Read-Evaluation Loop, arbeitet das zweite Programmierparadigma auf Basis von Mitteilungen. Ein der Anwendung externer Notifier verwaltet den Kontrollfluss und die Befehlsverarbeitung, filtert also die Benutzereingaben, und bewirkt so, dass nur diejenigen Befehle verarbeitet werden, die für die Anwendung von Interesse sind.
Das Programm der Anwendung legt fest, welche Befehle dies sind, stellt eine eingene Prozedur als sogenannten Callback für den jeweiligen Befehl auf und übergibt anschließend die Kontrolle an den Notifier. Wenn der Notifier einen Befehl des Window Systems erhält, prüft er, ob dieser Befehl von der Anwendung festgelegt wurde und, falls ja, übergibt ihn der dem Befehl zugeordneten Callback-Prozedur. Nach der Verarbeitung geht die Kontrolle wieder zum Notifier über, der entweder weitere Befehle verwalten kann oder terminiert. [4]

4 Werkzeuge (Toolkits)

Der Begriff Toolkit umfasst allgemein eine Sammlung von Bibliotheken, Klassen und Schnittstellen. Aus der Sicht des Benutzers sind Ein- und Ausgabe getrennt. Um den

[3] Quelle: [1], Seite 296f.
[4] Abb. 3.2, siehe Seite 6; Quelle: [1], Seite 299

```
# include <xview/xview.h>
# include <xview/frame.h>
# include <xview/panel.h>

Frame frame;

main (argc, argv)
int argc;
char* argv[];
{
  Panel panel;
  void quit();

  xv.init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

  frame = (Frame) xv_create(NULL, FRAME,
    FRAME_LABEL,   argv[0],
    XV_WIDTH,      200,
    XV_HEIGHT,     100,
    NULL);

  panel = (Panel) xv_create(frame, PANEL, NULL);

  (void) xv_create(panel, PANEL_BUTTON,
           PANEL_LABEL_STRING,   "Quit",
           PANEL_NOTIFY_PROC,     quit,
           NULL);

  xv_main_loop(frame);
  exit(0);
}

void quit()
{
  xv_destroy_safe(frame);
}
```

Abbildung 3.2: Ein einfaches notification-based Programm, das ein Panel mit einem Button „Quit“ erzeugt. Klickt man auf "quit", verlässt man das Programm.

Benutzer dabei zu unterstützen, Ein- und Ausgabe mehr in Verbindung miteinander zu bringen, wird ein anderes Niveau der Abstraktion auf das Window System gelegt. Ein klassisches Beispiel dafür stellt die Maus als Zeigegerät dar, da der Eingang, welcher aus der Hardware kommt, von der Produktion des Maus-Cursors getrennt ist. Dennoch glaubt der Benutzer sich mit dem Cursor verbunden, da die visuelle Anzeige der Maus mit der Bewegung des Gerätes zeitlich übereinstimmt, was ein Benutzergefühl auslöst.

4.1 Eine objektorientierte Annährung an Toolkits

Ziel eines Toolkits ist eine vereinfachte Entwicklung von Anwendungen für den Programmierer. Dafür stellen sie eine Sammlung von vorgefertigten Standardfunktionen und Schnittstellen zur Verfügung. Somit können Entwickler verschiedener Anwendungen das gleiche Toolkit verwenden, wodurch eine Einheitlichkeit der Programme gewährleistet und die Benutzerfreundlichkeit erhöht wird.
Die zwei folgenden Eingenschaften von Interaktionsobjekten und Toolkits machen die Anwendungen für eine objektorientierte Annährung an die Programmierung zugänglich. Einmal kann es davon abhängen, dass eine Klasse von Interaktionsobjekten, welche in einer Anwendung mit nur geringfügigen Veränderungen zu jeder Instanz aufgerufen werden kann, definiert wird. Des weiteren ist das Bilden komplexer Interaktionsobjekte leichter gehalten, da ihre Definition auf einfachen Objekten aufbaut. Diese Ansicht von Instanziierung und Vererbung sind Ecksteine der objektorientierten Programmierung.

4.2 Einzel- und Mehrfachvererbung

Klassen dienen als Schablonen für Interaktionsobjekte und werden als Erklärung für die Bildung dieser verwendet. Werden Klassen von Interaktionsobjekten definiert, besteht die Möglichkeit, neue Klassen zu erstellen, welche die Eigenschaften der ein oder anderen Klasse erben können, wofür es im einfachsten Falle eine strenge Klassenhierarchie gibt. Diese besagt, dass Klassen nur von ihren Elternklassen erben können und wird Einzelvererbung genannt. Weiter gibt es noch eine kompliziertere Hierarchie, bei der definierenden neuen Klassen erlaubt ist, von mehreren Elternklassen zu erben, was man als Mehrfachvererbung bezeichnet.
Dem Programmierer steht es zur Verfügung, sich Verhalten und Erscheinungsbild eines Interaktionsobjektes nach seinen Wünschen zu schneidern, indem er Werte verschiedener Attribute setzt. Die Möglichkeit des Maßschneiderns ist häufig auf einen kleinen Satz von Attributen für jede gegebene Klasse beschränkt, um die Leistungsfähigkeit beizubehalten.

5 User Interface Management Systems (UIMS)

Toolkits vereinfachen die Implementierung eines interaktiven Systems, da dem Programmierer ein Paket vorgefertigter Interaktionsobjekte mit bestimmtem Verhalten geboten wird. Jedoch sind auch der Programmierung mit Hilfe von Toolkits Grenzen gesetzt. Toolkits ...:

- ... sind teuer in der Herstellung.
- ... stellen nur eine limitierte Anzahl an Interaktionsobjekten zur Verfügung.
- ... sind für Nicht-Programmierer schwer zu nutzen.
- ... erlauben es selbst erfahrenen Programmierern nicht immer, definitiv nutzbare Schnittstellen zu entwickeln.

Sogenannte User Interface Management Systems (UIMS) stellen dem Entwickler dabei eine vielzahl von Techniken zur Verfügung, die über die Möglichkeiten von Toolkits hinausgehen. Dabei liegen die Schwerpunkte eines UIMS wie folgt:

- Entwicklung einer logischen Struktur eines interaktiven Systems, bezogen auf die Trennung von Anwendungssemantik und Präsentation
- Bereitstellen von Techniken zur Entwicklung der oben genannten logischen Struktur
- Unterstützung des Managements, der Ausführung und der Bewertung interaktiver Laufzeitsysteme

5.1 UIMS als logische Struktur

Einer der Schwerpunkte eines UIMS liegt in der Trennung der Semantik der Anwendung und der Benutzerschnittstelle. Gründe, die diese Trennung befürworten, sind:

- **Übertragbarkeit** Um ein- und dieselbe Anwendung auf verschiedenen Systemen zur Verfügung stellen zu können, ist es von Vorteil, die eigentliche Entwicklung von der Geräte-abhängigen Schnittstelle zu separieren.
- **Wiederverwendbarkeit** Trennung erhöht die Wahrscheinlichkeit, dass Komponenten wiederverwendet werden können, um Entwicklungskosten einzusparen.
- **mehrfache Schnittstellen** Damit eine Anwendung flexibler wird, können mehrere verschiedene Schnittstellen entwickelt werden, die über die gleiche Funktionalität verfügen.
- **Verfügbarkeit** Um die Effektivität einer interaktiven Anwendung zu optimieren, können der Benutzer und der Designer der Schnittstelle diese verändern, ohne jedoch die ihr zu Grunde liegende Anwendung anpassen zu müssen.

Durch die Trennung von Anwendung und Präsentation wird sofort auch die Frage nach der Kommunikation dieser beiden Komponenten aufgeworfen. Dazu existiert die sogenannte Dialogkontrolle, die zusammen mit Anwendung und Präsentation / Schnittstelle die drei Hauptkomponenten eines UIMS stellt.
Viele UIMS basieren auf dem notification-based Programmierparadigma. Hierbei übernimmt die Dialogkontrolle die Funktion des Notifiers und wird somit dazu benutzt, den Kontrollfluss zu steuern.

5.1.1 Seeheim Modell

Ein erstes Modell zur Darstellung der logischen Komponenten eines UIMS wurde in Seeheim, Deutschland, aufgestellt. Die einzelnen Komponenten sind hierbei wie folgt definiert:

- **Präsentation / Darstellung [presentation]** Komponente, die das Aussehen der Schnittstelle, inklusive Benutzereingabe und -ausgabe, bestimmt
- **Dialogkontrolle [dialog controll]** Komponente, die die Kommunikation zwischen Anwendung und Präsentation reguliert
- **Anwendungssemantik [application semantics]** Sicht auf die Anwendungssemantik der Schnittstelle

Das Schaubild[5] dient als graphische Interpretation des Seeheim Modells und verdeutlicht die Trennung der einzelnen Komponenten und deren Kommunikation. Zusätzlich sind hierbei der Benutzer und die Anwendung mit einbezogen, da somit die Betrachtung eines UIMS auf das ganze interaktive System ausgeweitet werden kann.
Mit den Jahren hat sich auch das Seeheim Modell verändert und wurde „modernisiert". Die sogenannte „lower box"[6] erlaubt, dass die Dialogkontrolle umgangen werden kann. Dadurch wird es möglich, dass Anwendung und Präsentation direkt und ohne Umwege miteinander kommunizieren, was in vielen Fällen die Effektivität des Modells und damit des ganzen Systems steigern kann.

5.1.2 Model-View-Controller (MVC) Modell[7]

Ein zweites Modell, dass eine logische Struktur für UIMS bietet, ist das Model-View-Controller Model (kurz: MVC). Es beantwortet die Frage, wie große und komplexe interaktive Systeme aus kleineren Komponenten zusammengesetzt werden können. Der Ansatz liegt dabei in der Feststellung, dass bei der Entwicklung einer logischen Struktur Vorteile aus einem objektorientierten Toolkit gezogen werden können. Die Ansiedlung des MVC Modells in einer objektorientierten Programmierumgebung ist also eine große Erweiterung zum Seeheim Modell.
Das MVC Modell beruht auf Smalltalk, einer der ersten erfolgreichen objektorientierten Programmiersprachen; wobei heutzutage auch Java (Swing) und andere Sprachen gebräuchlich sind. Ein erstes GUI, basierend auf dem MVC Modell, war zum Beispiel Smalltalk-80[8].
In Smalltalk kann die Verbindung zwischen Anwendungssemantik und Präsentation durch die so genannte MVC-Triade[9] realisiert werden.

[5] Quelle: [1], Seite 308, Abb. 8.10
[6] Quelle: [1], Seite 308, Abb. 8.10
[7] Quelle: [1] und [3] www-media.informatik.rwth-aachen.de/dis2_ss07_aachen.html, Video vom 18.04.2007
[8] Smalltalk-80 war eine der ersten graphischen Benutzeroberflächen, die dem Benutzer mehrere Fenster zur Interaktion bereitstellt.
[9] Quelle: [1], Seite 310, Abb. 8.11

- **Model** repräsentiert die Anwendungssemantik
- **View** managed die graphische und / oder textuelle Ausgabe
- **Controller** interpretiert die Eingabe

Ein Beispiel für die Funktionen und das Zusammenwirken von Model, View und Controller ist zum Beispiel ein System zur Wettervorhersage. Temperatur- oder Luftdruckwerte oder sonstiges werden als Zahlenwerte im Model gespeichert. Der Controller wäre die Komponente, die durch Temperaturfühler oder auch durch textuelle Eingabe bestimmte Werte liest und verarbeitet. Und View wäre in diesem Fall die (graphische) Darstellung auf dem Bildschirm in Form von Diagrammen, Tabellen etc.
Das Model hat dabei zwei grundlegende Eigenschaften. Zum einen erwartet es eine Benachrichtigung des Controllers, der bei Änderung eines Wertes die im Model gespeicherte Zahl „manipuliert". Zum anderen sorgt es für ein Update des View. Eine geänderte Eingabe resultiert in einer anderen Ausgabe.
Dadurch wird das MVC Modell auch so wichtig und für viele interaktive Anwendungen grundlegend. Die Anwedung selber, also das Model, ist unabhängig von View und Controller. Die generischen Klassen `View` und `Controller` bedürfen keiner großen Modifikation nach ihrer Instanziierung, da Smalltalk eine vorimplementierte Auswahl an Eingabe-/Ausgabeverhalten bietet. Diese beiden Komponenten treten außerdem stets als Paar auf, haben Instanzen der jeweils anderen Klasse und eine Instanz für das Model. Jedes dieser Paare ist nämlich grundlegend mit einem Model verlinkt und logischerweise kann ein Modell durch verschiedene View-Controller Paare realisiert werden. In dem Beispiel sind zum Beispiel Fühler und textuelle Eingabe zwei verschiedene Controller. Somit besitzt auch das Model keine generische Klasse, sondern ist auf die jeweilige Anwendung angepasster Code. Durch die Verknüpfung unterschiedlicher MVC-Triaden kann also ein komplexes interaktives System aus verschiedenen kleineren Komponenten beziehungsweise Teilanwendungen zusammengesetzt und entwickelt werden.

5.1.3 Presentation-Abstraction-Controll (PAC) Modell

Das PAC Modell ist eine so genannte multi-agent Architektur, die von Coutaz[10] entwickelt wurde. Wie das MVC Modell, basiert auch das PAC Modell[11] auf einer Verknüpfung von Triaden, bei denen allerdings noch mehr Komponenten zusammengefasst sind. Die Anwendungssemantik wird durch die Abstraction-Komponente repräsentiert, Ein- und Ausgabe werden in einer Presentation-Komponente zusammengefasst und eine explizite Controll-Komponente managed den Dialog zwischen Anwendung und Präsentation.
Der wichtigste Unterschied zwischen beiden Modellen ist die Tatsache, dass Ein- und Ausgabe beim PAC Modell in einer Komponente zusammengefasst sind. Gerade dadurch ist es leicht nachvollziehbar, dass das PAC Modell auch wirklich mehr Modellcharakter hat und eher eine logische Struktur eines UIMS ist. Das Zusammenfassen von Ein- und Ausgabe in

[10] Joelle Coutaz ist Professorin für Informatik an der Joseph Fourier Universität in Grenoble, Frankreich. Dort leitet sie den Lehrstuhl für HCI in dem Labor CLIPS (Communication Langagière and Interaction Personne Système). Coutaz ist die Autorin des in Teil 5 vorgestellten PAC Modells.
[11] Quelle: [1], Seite 311, Abb. 8.12

einer Komponente zeigt dabei schon die Abwendung von einer bestimmten Programmierumgebung, womit das PAC Modell letztlich weniger von der Implementierung abhängig ist.

5.2 UIMS zur Entwicklung realitätsnaher Benutzerschnittstellen[12]

Durch die zunehmende Verbreitung von Wireless-Technologien und Geräten (wie zum Beispiel Handys, Kameras, Sensoren etc.) ist es wichtig geworden den Benutzern immer und überall den Service bieten zu können über die Hardwaregeräte auf die verschiedenen Anwendungen zugreifen zu können. Deshalb ist so eine neue Art der Benutzersschnittstelle entstanden, deren Entwicklung auf der Beschreibung durch UIMS beruht.
Die neue Art der Schnittstellen wird realitätsnahe Schnittstelle (engl.: Realtiy-Based Interface) genannt, beruhend auf den vielfältigen Möglichkeiten für den Benutzer, mit Objekten in der Realität zu interagieren. Da jedoch solch eine Schnittstelle schwieriger zu entwickeln ist als eine „normale“ Schnittstelle, werden wir UIMS hier als ein Werkzeug darstellen, das die Entwicklung realitätsnaher Schnittstellen erleichtert.

5.2.1 Motivation zur Entwicklung realitätsnaher Schnittstellen

Heutzutage werden massenweise Informationen und Dienste über die verschiedensten Hardwaregeräte ausgetauscht. Da bietet ein Wireless-Netz die besten Möglichkeiten, eine ganze Sammlung an Informationen zu beherbergen. Geräte wie Kameras, Sensoren und ähnliches, die in das Netz integriert werden, motivieren dabei die Entwicklung realitätsnaher Schnittstellen und die Bereitstellung von Anwendungen, die es dem Benutzer erlauben, mit der realen Welt zu interagieren, und nicht nur mit traditionellen Ein- und Ausgabegeräten. Dazu stehen verschiedenste Interaktionstechniken zu Verfügung, unter anderem konkrete und physikalische Benutzerschnittstellen und die Interaktion mit Kleinstgeräten.

5.2.2 Schwierigkeiten auf dem Weg zur Realisierung

Auf dem Weg zu einer realitätsnahen Schnittstelle gibt es einige Hindernisse, die man überwinden muss; obwohl zum Beispiel durch die immer geschulteren Benutzer die Interaktion mit solchen Schnittstellen als einfach empfunden wird. Einige Schwierigkeiten entstehen durch die parallele physikalische und digitale Ausgabe, verschiedene Arten der Interaktion sowie durch viele Benutzer, die gleichzeitig auf die Hardwaregeräte zugreifen wollen. Außerdem spielt die Übertragbarkeit hier eine große Rolle. Ein- und dieselbe Anwendungssemantik muss dynamisch auf anderen Systemen übernommen werden können.
Hier bieten Toolkits und Modelle zu wenige Möglichkeiten zur Entwicklung realitätsnaher Benutzerschnittstellen. Mit dem Einsatz von UIMS auf Grundlage von User Interface Description Languages (UIDL) können wir die Entwicklung vereinfachen und werden dazu von den UIMS bei der Beschreibung und Programmierung realitätsnaher Schnittstellen unterstützt.

[12] [2] A Framework for Building Reality-Based Interfaces for Wireless-Grid Applications, Orit Shaer, CHI 2005, Doctoral Consortium, 02.-07. April, Portland, Oregon, USA

5.3 Überlegungen zur Implementierung eines UIMS

5.3 Überlegungen zur Implementierung eines UIMS

Um ein UIMS zu entwickeln reicht es nicht aus, die Betrachtung verschiedener Modelle auf rein logische Ebene vorzunehmen. Natürlich ist es wichtig, zu wissen, wie die einzelne Komponenten einer solchen logische Struktur realisiert werden können. So zum Beispiel die unterschiedlichen Komponenten Präsentation, Dialogkontrolle und Anwendung des Seeheim Modells. In vielen Fällen (zum Beispiel beim Standard-X-Toolkit) ist die notification-based Programmierung ein Weg, die Anwendungsschnittstelle als Notifier zu programmieren.
Die sieben wichtigsten Techniken zur Implementierung hat Myers[13] zusammengestellt, dessen GARNET System in diesem Zusammenhang kurz vorgestellt werden soll.

Menü-Netzwerke
- Kommunikation zwischen Anwendung und Präsentation als Netzerk von Menüs und Untermenüs
- Programmierung der Verbindungen zwischen den einzelnen Menüs und den Aktionen, die sie ausführen

Grammatische Notationen
- formale kontextfreie (Typ 2) Grammatik (zum Beispiel die Backus-Naur-Form) zur Beschreibung des Dialogs
- Verwendung bei Kommando-basierten Oberflächen

Zustandsübergangsdiagramme
- graphische Darstellung des Dialogs durch STN's (engl.: State-Transition-Diagrams)
- Übersicht über die verschiedenen Zustände, in denen sich der Dialog befinden kann

Ereignissprachen
- Vorteil, das lokale Ein- und Ausgabeverhalten in Produktionsregeln zu beschreiben

Deklarative Sprachen
- Beschreibung, wie Anwendung und Präsentation in Beziehung zueinander stehen
- Modellierung einer Datenbank, auf die beide Komponenten zugreifen können

Constraints
- spezieller Unterpunkt der deklarativen Sprachen
- vor allem in dem von Hill vorgeschlagenen Abstraction-Link-View Modell (ALV, 'Al-Vee') benutzt

Graphische Spezifikation
- graphische Umsetzung des Dialogs (programming by demonstration)
- Programmierung der Interaktionsobjekte, die später auch dem Benutzer zur Verfügung stehen

Diese Techniken sind nicht grundsätzlich getrennt voneinander zu betrachten, sondern steigern sogar in Kombination miteinander die Effektivität eines interaktiven Systems. Ein Beispiel hhierfür ist das GARNET System von Myers. Diese sogenannte User In-

[13] Brad A. Myers ist Professor für Informatik (Fachbereich HCI) an der Carnegie Mellon Universität. Dort war eines seiner Projekte das Garnet System, eine User Interface Development Environment für graphische, interaktive Benutzeroberflächen.

terface Development Environment vereint eine deklarative Constraint-Sprache mit einer graphischen Spezifikation. GARNET stellt verschiedene Werkzeuge zur Verfügung, die es ermöglichen, eine direkt manipulierbare, interaktive graphische Benutzerschnittstelle zu entwickeln. Auf Ebene der Constraints bietet das System ein graphisches Toolkit, mit dem sich Graphikobjekte sehr einfach darstellen lassen und automatisch vom System erkannt werden. Außerdem wird dem Entwickler Lapidar, ein Interface Builder, zur Verfügung gestellt, mit dem sich zum Beispiel Toolbars oder Scrolleisten entwickeln und auch nutzen lassen können. Insgesamt also ist das GARNET System eine große Sammlung an Tools, die es erlauben, eine Benutzerschnittstelle graphisch zu implementieren und so auch zum Beispiel Nicht-Programmierer einen einfacheren Zugang zur Implementierung bekommen.

6 Zusammenfassung und Ausblick

In dieser Ausarbeitung haben wir uns Gedanken darüber gemacht, auf welche Weise wir die Entwicklung eines interaktiven Systems, im Zuge der Programmierung, unterstüzten können und welche Hilfsmittel uns hierzu zur Verfügung stehen.
Dabei ist die allgemeinste Ebene die der Window Systems, welche die Grundlage aller modernen WIMP-Interfaces sind und dem Programmierer erlauben, die Kontrolle über mehrere Geräte-unabhängige Anwendungen zu haben. Des Weiteren haben wir zwei Programmierparadgimen vorgestellt, die den Kontrollfluss des Dialogs regulieren; entweder innerhalb der Anwendung (read-evaluation) oder als externer Notifier (notification-based). Toolkits bieten uns eine andere Ebene der Abstraktion, indem sie Ein- und Ausgabeverhalten verknüpfen und uns Interaktionsobjekte bereitstellen, mit denen wir die Komponenten eines interaktiven Systems zusammensetzten können. Als letztes Level der Abstraktion haben wir User Interface Management Systems (UIMS) kennengelernt. Sie stellen eine logische Struktur der Komponenten eines interaktiven Systems dar und finden Anwendung in vielen Bereichen der Human-Computer Interaction; so zum Beispiel in der Entwicklung realitätsnaher Benutzerschnittstellen. Zum Schluss haben wir einige Techniken zur Implementierung eines UIMS kennengelernt und in diesem Zusammenhang herausgestellt, dass eine Tendenz in Richtung graphischer Spezifikationen existiert. Diese Entwicklung hat jedoch die Diskussion zur Folge, ob es wert ist, die Entwicklung interkativer Systeme auf Kosten der Ausdrucksstärke auch für Nicht-Programmierer zu öffnen.

Literatur

1 Human-Computer Interaction, Dix, Finlay, Abowd, Beale, Pearson-Verlag, 3rd Edition, 2004

2 A Framework for Building Reality-Based Interfaces for Wireless-Grid Applications, Orit Shaer, CHI 2005, Doctoral Consortium, 02.-07. April, Portland, Oregon, USA

3 www-media.informatik.rwth-aachen.de/dis2_ss07_aachen.html, Video vom 18.04.2007

4 www.cs.cmu.edu/afs/cs.cmu.edu/project/garnet/doc/papers/garnetIEEE.abstract