Objektorientierte Dialogspezifikation mit ODSN

Gerd Szwillus

Fachbereich Mathematik/Informatik, Universität - GH Paderborn

Zusammenfassung

Wir diskutieren die zentrale Rolle eines - in der Spezifikationssprache ODSN notierten - objektorientierten Kontrollmodells im Entwicklungsprozeß graphischer interaktiver Benutzungsschnittstellen. In diesem Papier geht es vorwiegend um die Strukturierung des Entwurfsprozesses in sinnvolle und effizient miteinander kooperierende Entwurfseinheiten, die eine separate Behandlung der relevanten Entwurfsbereiche erlaubt, ohne den Zusammenhang dazwischen außer acht zu lassen.

1 Einführung

Das Design graphischer Benutzungsschnittstellen impliziert die simultane Behandlung verschiedener Aspekte; zu den wichtigsten dabei zählen:

- **visuelles Design** (z.B. Metaphern, Aussehen, Gestaltung, Layout),
- **strukturelles Design** (z.B. Abfolge von Bildschirmen, Struktur von Menüs, Informationsverteilung auf verschiedene Fenster oder Dialogboxen),
- **Interaktionstechniken** (z.B. Einsatz von Tastatur, Maus, Touch-Screen, Spracheingabe, textuelle Ausgabe, Graphikausgabe mit den jeweils typischen Ein/Ausgabetechniken),
- **Softwarestrukturierung** (z.B. Objektstrukturen, aktive und passive Komponenten).

Heutzutage werden graphische Benutzungsschnittstellen typischerweise mit sogenannten *Interface Buildern* [5], einer unterliegenden Graphik-Basissoftware, wie *X-Windows* und einer prozeduralen oder objektorientierten Programmiersprache, wie etwa C oder C++, programmiert. Der Grundgedanke dieser Art von Entwicklung:

- Anordnen und Gestalten der *Widgets* mit dem Interface Builder,
- Anbinden von Verhalten an die Ereignisse, die „auf" den Widgets ausgelöst werden mit der Programmiersprache unter Verwendung von Routinen des Basissystems für graphische Operationen.

Die Konsequenz dieser Arbeitsweise ist es, daß die Betonung auf der visuellen Gestaltung der Standard-Widgets liegt, die einheitlich im Interface Builder durchgeführt wird. Alle anderen Aspekte aber werden teilweise im Interface Builder behandelt und teilweise, verteilt auf viele „kleine" *Callback*-Funktionen, in Code der Programmiersprache versteckt. So werden zwar einzelne, ggf. komplexe Dialogboxen im Interface Builder interaktiv gestaltet; die Abfolge von Dialogboxen, oder das Auftauchen von Fenstern, oder gar welche graphischen Elemente in den Fenstern erscheinen wird in Code definiert - untereinander und mit semantischen Aktionen vermischt. Somit leistet diese Arbeitsweise eine sehr gute Spezifikation der Widgethierarchie, aber macht eine separate und kohärente Behandlung und Darstellung der anderen Aspekte (bis auf die Softwarestrukturierung) nahezu unmöglich.

Man erkennt an verschiedenen modernen Ansätzen zur Spezifikation von Benutzungsschnittstellen, daß ein Bedarf für die konsequente Behandlung der einzelnen Aspekte vorhanden ist:

- Janssen's Dialognetze [6] oder auch der THESEUS-Ansatz [11] befassen sich explizit mit dem strukturellen Design von Benutzungsschnittstellen,

- OBJECTION [13] und andere Ansätze versuchen, den Entwurf und Einsatz anwendungsspezifischer Graphiken ähnlich einfach und effizient zu gestalten wie die Verwendung von Standardwidgets,
- Petri-Netz-Ansätze [1,2] modellieren Interaktionstechniken und stellen ihren Bezug zum strukturellen Design her.

In diesem Papier soll eine Modellierungstechnik unter dem Begriff *objektorientiertes Kontrollmodell* präsentiert werden, die - wurzelnd in den Ideen der klassischen UIMS-Ansätze [4,9] - eine Strukturierung von Benutzungsschnittstellen vornimmt, die es erlaubt, in separaten, sich gegenseitig ergänzenden Spezifikationsschritten, die angesprochenen Aspekte explizit zu behandeln und miteinander in Bezug zu setzen. Es entsteht dabei ein formales Modell, welches das strukturelle Design als zentralen Gestaltungsspielraum aufspannt. Dieses kann dann schrittweise angereichert werden um das visuelle Design der beteiligten Objekte - entweder als Standardelemente oder als applikationsspezifische Graphiken -, und um die Anbindung spezieller Interaktionstechniken. Die im Kontrollmodell entstehende Strukturierung in Objekte mit Attributen und Methoden kann unmittelbare Vorgabe für die Softwarestruktur der gesamten Applikation sein.

Dadurch daß das Kernmodell formal spezifiziert wird, profitiert man von den bekannten Vorteilen eines expliziten Modells: das Modell kann **analysiert** werden, es kann gegen eine formale Spezifikation **verifiziert** werden, man kann daraus **Prototypen** ableiten und es **separat** von der Programmentwicklung und anderen Designschritten entwickeln. Das objektorientierte Kontrollmodell-System ist z. Zt. in Implementation als Sprache *ODSN (Objektorientierte Dialogspezifikations-Notation)*. Der Vorläufer von ODSN, die Sprache DSN/2 [7], ist nicht objektorientiert, verfügt aber über das lauffähige Werkzeug KAP [8] zur Definition und Ausführung von Prototypen von DSN/2-Kontrollmodellen. Die Arbeit mit KAP zeigt, daß Modularisierung und Strukturierung in Objekte unbedingt erforderlich ist, um die Komplexität realistisch großer Modelle zu beherrschen. Die (positiven) Erfahrungen der Modellierung von Geräte-Schnittstellen mit KAP motivierten uns, DSN/2 zu ODSN zu erweitern und eine entsprechende - zur Zeit noch in Arbeit befindliche - Werkzeugumgebung zu entwickeln.

2 Zustände: Die zentrale Bedeutung des Kontrollmodells

Die Beschreibung der möglichen Zustände, die eine Benutzungsschnittstelle annehmen kann, und die zwischen diesen Zuständen gegebenen Übergänge, beinhalten zwangsläufig wesentliche Designentscheidungen. Der „aktuelle" Zustand bestimmt, welche Aktionen der Benutzer (oder auch das System) durchführen kann und welche nicht - eine ganz entscheidende Information für den Benutzer: „Wo bin ich ? Wie kam ich hierher ? Warum kann ich etwas nicht tun ?". In modernen Benutzungsschnittstellen findet sich dieses Phänomen beispielsweise

- beim „Ausgrauen" von Menüpunkten,
- in kontextabhängig sich verändernden Menüs,
- in objektabhängigen *Drop-Down*-Menüs,
- oder beim impliziten Moduswechsel durch Bewegen der Maus.

Für gute *Benutzbarkeit („Usability")* ist das Design eines gut verständlichen oder auch möglichst unauffälligen Kontrollverhaltens daher unumgänglich. ODSN erlaubt es, dieses Verhalten in einer objektorientiert strukturierten Sprache explizit zu spezifizieren. Das resultierende *objektorientierte Kontrollmodell* wird damit zur zentralen Instanz der Benutzungsschnittstelle,

aber auch des Entwicklungsprozesses. Durch Anreicherung, Verfeinerung und Modifikation entsteht schrittweise ein Modell, das

- es erlaubt, früh im Entwicklungsprozeß **Prototypen** zu bauen,
- damit das Kontrollmodell und seine Auswirkung auf die **Usability** zu testen,
- es **formal** zu **analysieren** auf mögliche Designfehler,
- und es als Ausgangspunkt für eine **Generierung** oder auch (nur) eine **formale Spezifikation** für die Programmiertätigkeit zu verwenden.

Die schon erwähnten klassischen Ansätze zur Spezifikation von Dialogen verwendeten ebenfalls den Zustandsbegriff; dabei wurden die Zustände einer Benutzungsschnittstelle explizit dargestellt. In modernen Oberflächen würde dabei eine unbehandelbar hohe Zahl von Zuständen entstehen - daher verloren diese Ansätze an Bedeutung. Mit PPS [10] hat Dan Olsen die Idee eingebracht, statt dessen Objekte mit „lokalen" Zuständen zu definieren, deren Gesamtheit dann den „großen" Zustandsraum bestimmt, ohne diesen jemals explizit komplett aufzuzählen (oder zu zeichnen). Diesen Grundgedanken, der in die Dialogbeschreibungssprache DSN [3] einfloß, wird jetzt in ODSN ausgebaut zu einem komplexen, objektorientierten Kontrollmodell. Wir zeigen, daß dies eine relevante Abstraktionsebene ist, die separat spezifierbar, definierbar, ausführbar und analysierbar ist. Außerdem definiert sie klare Schnittstellen zur Präsentation (Ausgabe und Eingabe) und zur Applikation.

3 Das objektorientierte Kontrollmodell

3.1 Beschreibung einzelner Objekte

3.1.1 Zustandsraum

Eine ODSN-Spezifikation beschreibt Objekte der Benutzungsschnittstelle, die zu jedem Zeitpunkt jeweils einen wohldefinierten Zustand besitzen und diesen Zustand durch das Eintreten von Ereignissen wechseln können. Dabei wird für jedes Objekt der Zustandsraum durch *Felder* spezifiziert; dies sind Listen von sogenannten *Flags*. Zu jedem Zeitpunkt ist jeweils **genau eines** der Flags eines Feldes „gültig" - es können niemals zwei Flags eines Feldes gleichzeitig angenommen werden. So ist etwa durch die Felder HighlightState, BorderState und Colour, notiert als

```
HighlightState   (#NotHigh #High)
BorderState      (#Normal #Thick)
Colour           (#Red #Blue #Yellow #Green)
```

festgelegt, daß ein Objekt mit diesen Feldern als Zustandsraum-Definition unter anderem im Zustand (#High #Thick #Green) sein und insgesamt 16(=2*2*4) verschiedene Zustände annehmen kann.

3.1.2 Ereignisse

Der Zustand eines Objektes ändert sich ausschließlich durch das Eintreten von Ereignissen. Dies kommen typischerweise von **außerhalb** des Objektes und werden entweder vom Benutzer oder von kooperierenden Objekten ausgelöst bzw. erzeugt. Diese Ereignisse werden im Objekt - als INPUT EVENTS - definiert. So legt

```
INPUT EVENTS
   Highlighting (iHighlight   iDeHighlight)
   Border       (iNormalize   iThicken)
   Picking      (iHit)
```

fest, daß das Objekt auf die angegebenen Eingabeereignisse reagieren kann.

3.1.3 Zustandsänderungen

Die Zustandsänderungen, die ein Ereignis hervorruft, werden durch die *Übergangsregeln* eines Objekts definiert. Regeln enthalten auf der „linken" Seite eine Aufzählung der Flags, die gesetzt sein müssen, sowie ein Ereignis, das die Regel schalten läßt; auf der „rechten" Seite steht die Liste der Flags, die durch das Schalten der Regeln gesetzt werden. Außerdem kann eine Regel rechts auch Ereignisse (eigene oder solche anderer Objekte) enthalten, die dann durch das Schalten der Regel ausgelöst werden. So besagt die Regel

#Green #NotHigh iHighlight → #High #Blue,

daß bei gesetzter Farbe #Green und falls das Objekt nicht-„highlighted" (#NotHigh) ist, das Ereignis iHighlight einerseits die Farbe auf #Blue ändert und außerdem der Zustand „highlighted" (#High) gesetzt wird. Für jeden aktuellen Zustand kann man dem Kontrollmodell durch Analyse entnehmen, welche Ereignisse in diesem Zustand überhaupt zum Schalten einer Regel führen können. Wäre etwa die obige Regel die einzige Regel mit dem Ereignis iHighlight, so würde dieses Ereignis zum Beispiel bei Farbe #Yellow oder im Highlight-Zustand #High nicht zum Schalten irgendeiner Regel führen und wäre somit ein nicht erlaubter Übergang. Insgesamt entsteht durch die Definition von Zustandsraum, verarbeiteten Ereignisse und Regeln eine Beschreibung von Objektverhalten, die festlegt,

- welche Zustände ein Objekt überhaupt annehmen kann (Zustandsraum),
- welche Ereignisse prinzipiell von diesem Objekt entgegengenommen werden (Aufzählung der INPUT EVENTS),
- welche Ereignisse genau in jedem Zustand als Eingaben „erlaubt" sind (Analyse der linken Seiten der Regeln),
- wie der dann resultierende neue Zustand aussieht (Analyse der rechten Seiten der Regeln).

3.2 Objektstrukturen

Derartige Beschreibungen können als Typbeschreibungen (Klassen) oder als individuelle Objektdefinitionen in ODSN spezifiziert werden. Klassenbeschreibungen können dann beliebig oft instanziiert werden. Die Notation für *statische Instanziierung* sieht wie folgt aus:

CLASS Rectangle ... ENDCLASS
OBJECT Canvas; OBJECT r1, r2: Rectangle; ... ENDOBJECT

Dem Objekt Canvas stehen unter den Namen r1 und r2 zwei Objekte vom Typ Rectangle zur Verfügung, die in der darüber angedeuteten Klassenbeschreibung definiert sind. Um auch dynamische Effekte - wie Objekterzeugung und -zerstörung - modellieren zu können, gibt es auch die Möglichkeit der dynamischen Instanziierung über Regeln: Nach Definition eines Objekttyps kann ein Objekt durch Schaltenlassen einer Regel erzeugt, zerstört, aktiviert und deaktiviert werden. Die folgende Regel etwa erzeugt ein neues Objekt vom Typ Rectangle, wenn sie schaltet: #CreatingR iPos → NEW Rectangle.iCreate.

NEW ist ein Schlüsselwort; iCreate ist ein vordefiniertes Ereignis, das implizit für jede Klasse definiert ist. Analog kann man einem derart erzeugten Objekt u.a. die Ereignisse iDestroy, iActivate und iDeActivate schicken. Dieses Konzept beinhaltet auch das Ansprechen dynamisch erzeugter Elemente, soll hier aber nicht genauer behandelt werden (s. jedoch [12] und Abschnitt 3.4 Datenfluß).

3.3 Objektkooperation

Damit die Objekte einer solchen Spezifikation kooperieren können, müssen sie Informationen austauschen. Wir hatten bereits gesagt, daß jedes Objekt ausschließlich durch Auslösen „seiner" Eingabeereignisse agieren kann. In ODSN können, aufgrund der Objektstruktur, Ereignisse sowohl **vom Benutzer** als auch **von anderen Objekten** kommen; zur Definition einer Benutzungsschnittstelle ist es unumgänglich, eindeutig festzulegen, welche Ereignisse der Benutzer auslösen können soll, und welche intern erzeugt werden. Zu diesem Zweck markieren wir die Eingabeereignisse als „extern" (vom Benutzer kommend) und „intern" (von anderen Objekten kommend). Extern ist die Voreinstellung, daher fügt man i.w. das Schlüsselwort intern zu den internen Eingabeereignissen zu. So beschreibt die folgende Modifikation der Eingabeereignisse der Objekte vom Typ Rectangle, daß vom Benutzer lediglich das Ereignis iHit empfangen wird, während die anderen Ereignisse ausschließlich von anderen Objekten kommen können:

```
class Rectangle
    INPUT EVENTS
        intern Highlighting  (iHighlight, iDeHighlight)
        intern Border        (iNormalize, iThicken)
        Picking              (iHit)
    ...
ENDCLASS
```

Das Objekt Canvas kann nun durch Verwenden von r1.iThicken auf der rechten Seite einer Regel dem Rectangle-Objekt r1 den „Befehl" geben, seinen Rand fett darzustellen.

Andererseits bedingt Objektkooperation auch, daß ein Objekt (etwa r1) einem anderen Objekt (etwa Canvas) Informationen über seinen Zustand geben können muß. Dies geschieht durch **Bekanntmachen** einiger (oder aller) Flags eines Objekts - syntaktisch wiederum mit dem Schlüsselwort intern. Ist etwa der HighlightState als intern vereinbart,

```
intern HighlightState  (#NotHigh #High),
```

dann kann man in Canvas etwa das Flag r1.#High auf der linken Seite einer Regel abfragen. Gemäß dem Axiom, daß ein Objekt nur durch Ereignisse aktiv beeinflußt werden kann, darf dieses Flag jedoch nicht von außen gesetzt werden - also auf der rechten Seite einer Regel auftauchen.

Durch die Festlegung der Schnittstellen von Objekten über das interne Bekanntmachen einiger Eingabeereignisse und einiger Flags ist ein klares Konzept der Kooperation von Objekten definiert. Diese folgt der Idee, daß die „Bedienung" eines Objektes durch den menschlichen Benutzer intern nachgebildet wird: Der Benutzer kann Ereignisse „auf" einem Objekt auslösen und einige der Zustandsgrößen (Flags) in ihrer Veränderung beobachten. Genauso können nun andere Objekte Ereignisse auslösen und einige der Zustandsflags in ihren Regeln abfragen. Dadurch entsteht in der Beschreibung eine strukturgebende Symmetrie zwischen interner und externer Kooperation. Gleichgültig, ob ein Objekt letztlich „direkt" vom Benutzer bedient wird oder über ein Mittlerobjekt erreicht wird, bleibt die Denkweise beim Entwickeln eines Objektes gleich.

3.4 Datenfluß

Vielfach wird bei Ereignissen, die der Benutzer auslöst, nicht nur das Ereignis selbst, sondern auch „kleine" Zusatzinformation transportiert. Ein Mausklick etwa trägt typischerweise die Koordinaten des getroffenen Punktes mit sich. Diese Beobachtung machen wir zum generellen

Prinzip für die Kooperation von Objekten untereinander und mit dem Benutzer. So mag es etwa von Bedeutung für das Objekt Rectangle sein, bei welchen x- und y-Koordinaten das Ereignis iHit auftritt. Wir erweitern daher das „reine" Eingabeereignis iHit um die Angabe, daß auch ganzzahlige Werte x und y mitgeliefert werden: Picking (iPos {int x, y}).

An diesem Punkt kümmern wir uns nicht darum, **wie** diese Werte entstehen - wir legen auch nicht fest, daß iHit durch eine Mausoperation realisiert wird. Wir sagen nur **daß** das Ereignis iHit unter den Namen x und y ganze Zahlen „mitliefert". Als Datentypen stehen hier Standardtypen, aber auch der Typ oid zur Verfügung, was für *Objektidentifikator* steht.

Um Datenflußwerte festzuhalten, geben wir auch den Feldern - wie den Ereignissen - die Möglichkeit, Werte zu speichern. Damit ergibt es sich nun, daß die Ereignisse und Flags, die „links" in einer Regel auftreten, Daten „mitbringen" können, die dort in der Regel zur Verfügung stehen. Sie können in mehrfacher Weise verwendet werden:

1. Als Test, ob eine Regel schalten kann durch einen einfachen Vergleich der Werte etwa mit einer Konstanten.
2. Zum Setzen der Zusatzinformation eines Flags oder erzeugten Ereignisses auf der rechten Seite der Regel.
3. Zum zielgerechten Ansprechen von Objekten, deren Objektidentifikation bekannt ist, um diesen Ereignisse zu senden.

Bis zu diesem Punkt der Definition des Kontrollmodells ist eine sinnvolle, in sich geschlossene Einheit entstanden, die einen wesentlichen Gestaltungsspielraum aufspannt, der die Definition der Benutzungsschnittstelle und ihre gute Benutzbarkeit direkt beeinflußt. Nicht behandelt wurden bislang die Aspekte:

- **Visuelles Design** (wie sehen die Objekte aus?)
- Mit welchen **Interaktionstechniken** agiert der Benutzer ?
- Wie ist die **Applikation** angeschlossen ?
- Wie sieht die **Softwarestruktur** der Benutzungsschnittstelle mitsamt Anwendung aus ?

Für die Anreicherung des Gesamtmodells um diese Informationen schafft jedoch gerade das Kontrollmodell ideale Voraussetzungen.

4 Anbindung der Präsentation

4.1 Der Gestaltungsspielraum

Nach der Definition des Kontrollmodells liegen einige Informationen über die visuelle Präsentation und die Eingabetechniken bereits fest:

- Welche Objekttypen mit welchem Zustandsraum müssen überhaupt dargestellt werden?
- Welche Benutzerereignisse diesen Objekten vom Benutzer „geschickt" werden; mit welchen Zusatzinformationen müssen diese behaftet sein?

Nicht festgelegt ist im Kontrollmodell,

- Wie sehen die Objekttypen genau aus (s. Abschnitt 4.2 Objektgestaltung)?
- Wann genau sind welche Objekte sichtbar (s. Abschnitt 4.3 Objektsichtbarkeit)?
- Wie werden die Benutzereignisse „auf" den Objekten ausgelöst (s. Abschnitt 4.4 Benutzerereignisse)?

4.2 Objektgestaltung

Um die Gestalt eines Objekttyps festzulegen, steht dem Entwickler der im Kontrollmodell definierte Zustandsraum des Objekttyps zur Verfügung. Wir unterscheiden explizit bei der Zuordnung eines „Objektaussehens" die *Gestalt* und die *Projektion* dieser Gestalt auf ein Darstellungsmedium - etwa eine zweidimensionale Graphikfläche. Den Objekttypen des Kontrollmodells selbst ordnen wir eine - mit den Daten des Zustandsraums parametrisierte - Gestalt zu. Dabei muß die Gestaltform eines Objekttyps so gewählt, daß die Darstellungsfunktion, mit der diese Gestalt später dem Benutzer gezeigt werden soll, diese auch darstellen kann. Mögliche Gestaltformen bei Einsatz **konventioneller zweidimensionaler Graphik** sind die dort typischen Elemente wie Kreise, Linien, Rechtecke, Bitmaps, Kombinationen hiervon, aber auch flächige Widgets aus Interaktionsbibliotheken; ist die Darstellung die perspektivische Projektion einer **dreidimensionalen (virtuellen) Welt**, dann sind räumliche Objekte, wie Kugel, Kegel, Quader, Flächen und wiederum Kombinationen hiervon Kandidaten für Objektgestalten.

Die Gestaltdefinition des Objekttyps kann beliebige Komponenten des Zustandsraums - aber auch nur diese Information ! - benutzen, um die Gestalt zu beeinflussen. Bei dieser Festlegung wird also entschieden, ob eine intern gespeicherte Zustandsinformation dem Benutzer nach außen sichtbar gemacht wird, oder nicht. Unser Beispiel-Objekttyp mit dem Zustandsraum

```
HighlightState  (#NotHigh #High)
BorderState     (#Normal #Thick)
Colour          (#Red #Blue #Yellow #Green)
```

etwa könnte als 2D-Gestalt ein Rechteck besitzen, dessen Farbe vom Feld Colour gesetzt wird, dessen Rand, je nach gesetztem BorderState-Flag normal oder fett ist, und dessen HighlightState als Information zur inversen Darstellung ausgewertet wird. Wir könnten aber auch einen 3D-Quader mit ähnlicher Auswertung der Felder für eine 3D-Welt festlegen.

4.3 Objektsichtbarkeit

4.3.1 Darstellende Objekte

Die oben angesprochene Projektionen oder Darstellungsfunktionen, die Objektgestalten erst auf einem entsprechenden Medium sichtbar machen, werden ebenfalls Objekten zugeordnet. Darstellende Objekte werden entsprechend ausgezeichnet; neben ihrer Gestalt, die wie üblich definiert wird, haben sie die Eigenschaft, daß sie ganz oder teilweise dem Darstellen anderer Objekte dienen. Dieses Phänomen findet sich vielfach in graphischen Benutzungsschnittstellen, Paradebeispiel sind natürlich alle denkbaren Varianten von **Fenstern**, die typischerweise auf einer rechteckigen Zeichenfläche die Gestalt anderer Objekte zeigen.

In Ergänzung zum bislang definierten Kontrollmodell fügen wir die Information, welches Objekt welche anderen Objekte zeigt, in Form sogenannter shown-by-Klauseln hinzu. So könnten wir etwa mit

```
OBJECT Canvas;
    OBJECT r1: Rectangle SHOWN BY Canvas;
    OBJECT r2: Rectangle SHOWN BY r1;
...
ENDOBJECT
```

spezifizieren, daß das Rechteck r1 von Canvas gezeigt wird, während das Rechteck r2 von Rechteck r1 dargestellt wird. Auch hier enthält das Kontrollmodell lediglich die Information **daß** diese Darstellungsbeziehungen existieren - nicht wie sie genau aussehen.

Durch die SHOWN-BY-Klauseln eröffnen wir uns die Möglichkeit, schon bei der Definition des Kontrollmodells festzulegen, welche Abhängigkeiten in der graphischen Darstellungshierarchie bestehen. Dadurch können wir diese auf Benutzungsprobleme hin abzuprüfen, etwa ob vom Kontrollmodell her erlaubte Benutzerereignisse sich auch nur auf „sichtbare" Objekte beziehen, oder ähnliches.

4.3.2 Darstellungsfunktionen

Haben wir ein Objekt als „darstellend" im Kontrollmodell festgelegt, müssen wir eine Darstellungsfunktion dafür auswählen. Die gewählten zu zeigenden Gestalten müssen dabei zu dieser Darstellungsfunktion „passen". So können wir zum Beispiel festlegen,

- daß eine Teilfläche des Objektes Canvas sich wie ein Fenster verhält (also eine zweidimensionale Zeichenfläche aufspannt, die man z.B. scrollen und pannen kann), oder
- eine solche Fläche als Kamera auf eine 3D-Szene agiert, die man dann im Objekt Canvas einstellen, fokussieren, zoomen etc. kann.

Die *Darstellungsfunktion* beinhaltet die Information,

- wie **Koordinaten** der Objektgestalten in solche der Präsentation „umgerechnet" werden,
- wie mit **Farben** und Oberflächen etc. umgegangen wird, und
- wie gleichzeitige Darstellungen **mehrerer** Objektgestalten behandelt werden.

Die Koordinatentransformation beinhaltet Effekte wie Zooming und Panning im 2D-Fall, aber auch Blickpunkt oder Blickrichtung bei dreidimensionalen Szenerien. Auch Verzerrungen, wie Fish-Eye-View, oder verschiedene semantische Sichten können hierauf abgebildet werden. Die Abbildung mehrerer Objektgestalten besitzt Eigenschaften, die nicht aus der Definition der einzelnen Gestalten heraus abgeleitet werden können. Ein Beispiel aus dem Bereich 2D-Fenster wäre etwa der Umgang mit überlappenden Teilobjekten, die durch „z-Werte" unterschieden werden, wobei das Fenster - also das darstellende Objekt - diese z-Staffelung verwalten muß.

Die Darstellungsfunktion kann nun wiederum - genau wie die Gestalt der Objekte - aus dem Zustandsraum des darstellenden Objektes heraus parametrisiert werden. So könnte der Zoomfaktor, die Koordinate der linken oberen Ecke einer Zeichenfläche, oder die Position von Lichtquellen und/oder Kamera einer dreidimensionalen Szenerie dadurch beeinflußt werden. Zusätzlich kann der Zustandsraum auch die eigentliche - nicht-darstellende - Gestalt eines solchen Objekttyps beeinflussen. Dazu können im Fall von Fenstern etwa die Farbe und Dicke der Umrandung, oder auch der Text im Fenstertitel zählen. Für die Präsentation gilt, daß das Kontrollmodell den Rahmen aufspannt, innerhalb dessen sich die visuelle Gestaltung abspielt; dabei handelt es sich um strukturelle Eigenschaften, die sinnvoll abgespalten werden und somit separat bereits Entwurfsgegenstand sein können.

4.4 Benutzerereignisse

Nachdem definiert ist, wie die Objekte gestaltet sind und durch welche anderen Objekt sie wie dargestellt werden, kann man definieren, welche Benutzerereignisse in welcher Weise „auf" diesen Objekten ausgelöst werden können. Auch hier ist das Kontrollmodell eine präzise Vorgabe dafür, **welche** als extern angegebenen Ereignisse mit **welchen** zugehörigen Datenflüssen realisiert werden müssen, ohne zu präjudizieren wie diese Zuordnung aussieht. Die technische Umsetzung der Benutzungsschnittstelle - etwa als 2-D-Fensteroberfläche - bedingt das Vorhandensein bestimmter Ereignistypen, die auf den Basiselementen der Umgebung basieren - etwa Mausklick, Doppelklick, Entry-Events, Exit-Events etc. Diese „technischen" Ereignisse

müssen nun den „logischen" Ereignissen des Kontrollmodells zugeordnet werden. Entsprechend der zweigeteilten Darstellungskonzeption (Gestalt, Darstellungsfunktion) muß hierbei ebenfalls zweistufig vorgegangen werden: Die auftretenden technischen Ereignisse beziehen sich zunächst nur auf die darstellenden Objekte. Dieses Objekt muß analysieren,

- ob es „selbst" gemeint ist (etwa wenn ein Fensterrand mit der Maus „verzogen" wird),
- oder ob es das Ereignis an eines (und wenn ja, an welches) der dargestellten Objekte weiterreichen soll (etwa wenn eine Rechteck in der Zeichenfläche des Fensters angeklickt wird). In diesem Fall muß das technische Ereignis, das auf der Darstellung eingetreten war, „umgerechnet" werden in ein Ereignis auf der Objektgestalt des getroffenen Objektes.

Ähnliche Vorgänge werden in modernen Fenstersystemen ständig durchgeführt: Das Anklicken eines Bildschirmpixels (mit absoluten Bildschirmkoordinaten), welches aber in der Zeichenfläche eines Fensters liegt, wird umgerechnet auf ein Anklicken einer Zeichenflächenposition in **deren** Koordinatensystem. Dieser Transformationsprozeß ist Eigenschaft der Darstellungsfunktion für die darstellenden Objekte. Zusätzlich zur Gestaltdefinition muß dann noch vereinbart werden, welche Ereignisse auf den Objektgestalten definiert sind, und welchen „logischen" Ereignissen aus dem Kontrollmodell sie zugeordnet werden.

5 Anbindung der Applikation

Die Kopplung zwischen Applikation und Benutzungsschnittstelle muß bei modernen direktmanipulativen Schnittstellen sehr eng sein, um effizientes *semantisches Feedback* leisten zu können. Die Applikation wirkt in das Kontrollmodell hinein; umgekehrt wird die Applikation vom Kontrollmodell angestoßen, um semantische Berechnungen durchzuführen. Wir bilden diese Mechanismen ab auf ein ereignisorientiertes Konzept, wie es im Kontrollmodell ohnehin schon vorhanden ist.

5.1 Die Schnittstelle zur Applikation: Applikationsereignisse

Typischerweise erfolgt durch ein Benutzerereignis der Anstoß zum Aufruf einer Applikationsroutine. Wir gehen davon aus, daß die Applikation sich als Sammlung von Routinen präsentiert - im Sinne der üblichen *Callback*-Strukturen. Der Aufruf einer Applikationsroutine korrespondiert zu **zwei** Ereignissen im Kontrollmodell: dem **Aufruf** der Routine (und Versorgung mit Parametern) und der **Fertigmeldung** der Routine (mit Rücklieferung von Ergebnissen). Um dies im Kontrollmodell einzufangen, führen wir den Ereignistyp „Applikationsereignis" ein. Ein Applikationsereignis definiert einerseits einen Namen für das Ereignis und legt fest, welche Routine mit welchen Daten aufgerufen wird und welche Daten diese Routine zurückliefert. Der Aufruf einer Routine wird dann durch Erzeugen des Aufrufereignisses (auf der rechten Seite von Regeln) im Kontrollmodell spezifiziert; die Rückmeldung der Applikationsroutine mit den Ergebnissen kann auf der linken Seite von Regeln als Ereignis abgefangen werden. Durch diese Konzeption ist grundsätzlich Nebenläufigkeit zwischen Benutzungsschnittstelle und Applikation modellierbar. Es ist eine Eigenschaft des Kontrollmodells, ob es nach Absetzen einer Berechnungsfunktion der Applikation auf das Ergebnis „wartet" oder nicht.

Es zeigt sich, daß das Kontrollmodell gut geeignet ist, um die Schnittstelle zwischen Oberfläche und Applikation zu definieren: Hat man sich auf die zu verwendenden Applikationsereignisse geeinigt, kann separat an Oberfläche und Applikation gearbeitet werden. Dem Applikationsentwickler liegen dann Funktionsprototypen als Vorgaben für seine Arbeit vor; für den

Entwickler der Benutzungsschnittstelle ist eindeutig definiert, **wie** diese Applikationsfunktionen angebunden sind.

5.2 Objektorientierte Strukturierung der Applikation

Die Angabe von Applikationsereignissen in Objekten bzw. Klassen legt es nahe, die Applikation in Analogie zum Kontrollmodell zu strukturieren. Verwendet man ein Applikationsereignis in der Definition einer Klasse oder eines Objektes, kann man dies als Vorgabe dafür betrachten, daß es in der eigentlichen Applikation eine entsprechende Klasse bzw. ein entsprechendes Objekt gibt, das eine derartige Funktion als Methode besitzt. Das schließt nicht aus, das in der Applikation zusätzliche, für die Benutzungsschnittstelle unsichtbare Klassen und Objekte existieren. Die Objektstruktur des Kontrollmodells liefert aber eine sinnvolle Vorgabe für die objektorientierte Strukturierung der Applikation.

6 Werkzeuglandschaft

In Anlehnung an das bereits existierende Prototypwerkzeug KAP [8] ist eine Werkzeuglandschaft in Arbeit, die mit verschiedenen dedizierten Editoren und angeschlossenen Generierungskomponenten die Erzeugung von Prototypen, aber auch Teile der endgültigen gesamten Applikation erlauben wird. Zur Zeit ist die Definition der textuellen Version von ODSN abgeschlossen, und wir arbeiten an einem Simulator. Bis auf eine Generatorkomponente ist eine vergleichbare Werkzeuglandschaft im Tool KAP implementiert; allerdings für das nicht objekt-orientierte und auch in anderer Hinsicht „einfachere" DSN/2.

Literatur

[1] B. d'Ausbourg, G. Durrieu, P. Roche: Deriving a formal model of an interactive system from ist UIL description in order to verify and test its behaviour. In: Proceedings of the 3rd Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'96), Namur, 1996.

[2] J. Acott et al: A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. In: Proceedings of the 3rd Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'96), Namur, 1996.

[3] M.B.Curry, A.Monk: Dialogue modelling of graphical user interfaces. Technical Report, Department of Psychology, University of York, 1991.

[4] M. Green: A Survey of Three Dialogue Models. In: ACM Transactions on Graphics 5 (1986), 3, 245-275

[5] P. Griebel, M.Pöpping: Motifation - Ein Benutzungsschnittstellen-Entwicklungssystem. In: Informatik Fachberichte 293, Springer Verlag, 1991, 445-454.

[6] C. Janssen: Dialognetze zur Beschreibung von Dialogabläufen in graphisch-interaktiven Systemen. In: K.-H. Rödiger (Hrsg.): Software-Ergonomie '93 - von der Benutzungsoberfläche zur Arbeitsgestaltung. Stuttgart, Teubner, 1993, 67-76.

[7] K. Kespohl, G. Szwillus: Prototyping Device Interfaces with DSN/2. In: Proceedings of the 3rd Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'96), Namur, 1996.

[8] K. Kespohl, G. Szwillus: KAP - A Prototyper for Technical Device Interfaces. In: CHI'96 Conference Companion, Formal Demonstration, 1996

[9] B. Myers: User-Interface Tools: Introduction and Survey. In: IEEE Software (1989), 15-23.

[10] D.R. Olsen: Propositional Production Systems for Dialogue Description. In: Human Factors in Computer Systems, CHI'90 Conference Proceedings, 1990, 57-63

[11] E. Schlungbaum, T. Elwert: Modellierung von graphischen Benutzungsoberflächen im Rahmen des TADEUS-Ansatzes. In: H.-D. Böcker (Hrsg.): Software-Ergonomie '95 - Mensch-Computer-Interaktion. Anwendungsbereiche lernen voneinander. Stuttgart, Teubner, 1995, 331-348

[12] G. Szwillus: Ein objektorientiertes Kontrollmodell für graphische Benutzungsschnittstellen. In: Softwaretechnik'96, 1996.

[13] H. Uhr et al: OBJECTION: Entwicklung anwendungsspezifischer interaktiver Graphik. Software-Ergonomie '97 - Usability Engineering: Integration von Mensch-Computer Interaktion und Software-Entwicklung, Dresden, 1997

Adresse des Autors:

Gerd Szwillus
Universität - GH Paderborn, Fachbereich Mathematik/Informatik (17)
Fürstenallee 11, 33102 Paderborn
szwillus@uni-paderborn.de