Studienarbeit
Data structure-based Automatic User interface generation (DAU)

Sascha Silbe

31. August 2008

Inhaltsverzeichnis

Zusammenfassung

Im Rahmen dieser Studienarbeit wurde das Modul *Data structure-based Automatic User interface generation* (DAU) entwickelt, eine Abstraktionsschicht zwischen Applikation (View / Controller) und Widget Sets.

Ziel von DAU ist es, die selben Anwendungsprogramme verschiedenen Benutzern mit ihren speziellen Anforderungen und damit verschiedenen Benutzerschnittstellen / Widget Sets nutzbar zu machen. Dabei ist es möglich, neue Benutzerschnittstellen für bestimmte Benutzergruppen (z.B. Sehbehinderte) zu entwickeln, ohne die verwendeten Applikationen selbst anpassen zu müssen. Es muß lediglich ein passendes Backend für DAU geschrieben werden, das dann sofort von allen DAU-basierten Anwendungen genutzt werden kann.

1 Einleitung

1.1 Definitionen

Um Missverständnissen vorzubeugen, werden zunächst die folgenden Begriffe definiert:

Widget einzelnes Objekt, mit dem der Benutzer interagieren kann (z.B. Auswahl-Liste)

Widget Set Bibliothek, die Sammlung von Widgets (mit einheitlicher Schnittstelle) enthält (z.B. GTK)

Benutzerschnittstelle Vereinigung aller Objekte, die die Interaktion von Computer und Benutzer ermöglichen (z.B. Tastatur + Maus + X11 + GTK)

Applikation Programm, das eine vom Anwender direkt erwünschte Funktion bereitstellt und mit ihm interagiert (z.B. Editor)

Arbeitsumgebung (Desktop Environment) Sammlung von Programmen, die über gemeinsam verwendete Bibliotheken einheitliches Aussehen und gleichartige Bedienung erreichen und dabei ein konsistentes Gesamtsystem ergeben (z.B. KDE)

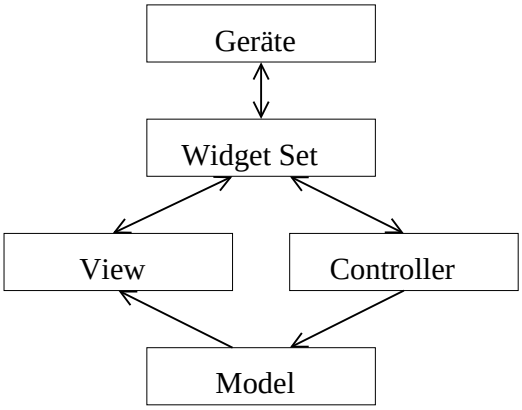

Abbildung 1: Bei der derzeit am weitesten verbreiteten Entwicklungsmethode für Anwendungsoberflächen werden zwar Darstellung (View) und Steuerung (Controller) von der „Geschäftslogik“ (Model) getrennt, die beiden Erstgenannten setzen jedoch direkt auf ein Widget Set auf und müssen damit zur Verwendung einer anderen Benutzerschnittstelle komplett ersetzt werden.

1.2 Motivation

Derzeit werden die meisten Anwendungsprogramme direkt mit einem einzelnen Widget Set verknüpft (Bild 1). Ein Austausch ist aufgrund inkompatibler Schnittstelle (Application Programming Interface, API) nicht möglich. Dies führt dazu, daß

1. Aussehen und Bedienung der Applikationen unterschiedlich ist (je nach verwendetem Widget Set)

2. viele Applikationen mehrfach programmiert werden, damit sie sich in die jeweilige Umgebung integrieren

3. Hilfestellungen oft nicht gegeben werden können, weil Hilfesuchender und Helfer verschiedene Anforderungen haben und dementsprechend andere Benutzerschnittstellen und in Folge andere Anwendungen verwenden (z.B. mutt für Kommandozeilen-Gurus und KMail für Gelegenheitsnutzer).

4. viele Programme gibt es nur für eine Art von Benutzerschnittstellen, diese sind für die Nutzer anderer Benutzerschnittstellen oft nicht zufriedenstellend zu bedienen (z.B. wget, KOrganizer)

Ziel von DAU ist es, die selben Anwendungsprogramme verschiedenen Benutzern mit ihren speziellen Anforderungen und damit verschiedenen Benutzerschnittstellen / Widget Sets nutzbar zu machen. Als (erwünschter)

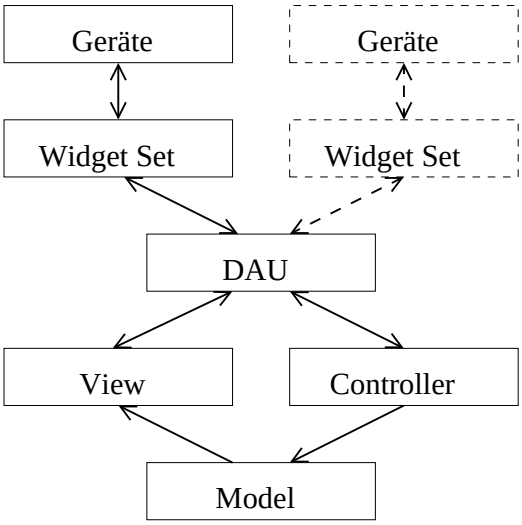

Abbildung 2: Bei Verwendung von DAU wird zwischen View/Controller und Widget Set eine weitere Abstraktionsschicht gesetzt. Damit ist es möglich, das Widget Set auszutauschen, ohne View und/oder Controller anpassen zu müssen. Neue Benutzerschnittstellen (z.B. eine spezielle Umgebung für Braille-Zeilen) können durch Entwicklung eines neuen Backends verwendet werden, ohne daß eine Änderung am Programm erforderlich ist (auch kein Neucompilieren o.ä.).

Nebeneffekt ist es möglich, neue Benutzerschnittstellen für bestimmte Benutzergruppen (insbesondere Behinderte) zu entwickeln, ohne dabei die verwendeten Applikationen anpassen zu müssen. Damit wären viele der heutigen Workarounds („Zugangshilfen“ wie Screenreader, Lupen und „klebrige Tasten“) nur noch selten nötig.

Ein weiteres, wenn auch derzeit eher untergeordnetes, Ziel ist Programme sowohl als Webapplikation (⇒ Thin Clients, Internetcafé, keine Installation nötig) als auch – wieder ohne Modifikation – auf dem Desktop (⇒ Offline-Nutzung, Datenschutz) nutzen zu können.

1.3 Was ist DAU?

DAU ist

- eine Abstraktion über verschiedenartige Benutzerschnittstellen (Bild 2)
- eine Trennung von Inhalt und Darstellung/Interaktion

Es lässt sich am ehesten mit HTML [W3Cb] + CSS [W3Ca] vergleichen. Analog zu HTML werden bestimmte Datenstrukturen und ihre Semantik

definiert. Wie diese von der jeweiligen Benutzerschnittstelle dargestellt werden und wie der Benutzer damit interagiert, ist nicht definiert. Es gibt – sowohl für Programmautoren als auch für Benutzer – die Möglichkeit, sog. Style Sheets zu definieren. Damit ist es möglich, Aussehen und Verhalten für einzelne Backends zu beeinflussen. Da Benutzer eigene Style Sheets angeben und auch andere Backends verwenden können, darf sich eine Applikation jedoch nicht auf ein bestimmtes Erscheinungsbild o.ä. verlassen.

DAU bezeichnet einerseits die Spezifikation einer Schnittstelle, andererseits aber auch die (Referenz-)Implementation. Im Rahmen der Studienarbeit sollte hauptsächlich überprüft werden, ob das Vorhaben praktisch umsetzbar und verwendbar ist. Eine umfassende Ausarbeitung der Spezifikation hätte den Rahmen bei weitem gesprengt; deshalb werde ich hier nicht streng zwischen Spezifikation und Implementation trennen.

Die derzeitige Implementierung verwendet Python, da damit eine schnelle und einfache Entwicklung möglich ist. Spätere Versionen könnten C oder (in eingeschränktem Umfang) C++ verwenden, um die Schnittstelle auch in anderen Sprachen bereitstellen zu können (z.B. via SWIG [SWI]).

1.4 Was ist DAU nicht?

DAU versucht nicht, alle möglichen Verwendungszwecke für Benutzerschnittstellen abzudecken, sondern lediglich die „Standardfälle“, mit Fokus auf Applikationen auf PCs. Ein einheitliches Erscheinungsbild wird ebenso wenig angestrebt (ganz im Gegenteil).

2 Beschreibung

2.1 Aufbau

DAU definiert eine Schnittstelle, die Anwendungen nutzen können, um eine Benutzerschnittstelle darstellen zu lassen. Verschiedene Backends implementieren diese Schnittstelle. Dabei ist es möglich, erweiterte Interaktionsmethoden (sog. Capabilities) anzubieten, die nicht von allen Backends unterstützt werden.

Die Schnittstelle basierte ursprünglich auf einem Paper von Tuomo Valkonen [Val04], ist jedoch deutlich modifiziert worden.

2.1.1 Application Programming Interface (API)

Die angebotene Schnittstelle orientiert sich hauptsächlich an Datenstrukturen mit grundlegenden Typen: logische Werte, Zahlen(bereiche), Zeichenketten, Listen und Structs (auch Record oder Verbund genannt). Zusätzlich gibt es (von Benutzer aktivierbare) Befehle sowie Leinwände.

Jedes Element hat dabei einen Namen, über den es identifizierbar ist. Die Namen sind hierarchisch aufgebaut; Structs und Listen dienen als Container für weitere Einträge. Elemente im Struct „std“ haben eine bestimmte, in der Spezifikation festgelegte Bedeutung (z.B. `std.seq.finish()` für Programmende) und können vom Backend ggf. anders behandelt werden (z.B. Dateiauswahl-Dialog für ein Zeichenfeld, das Dateinamen enthält). Über Style Sheets können auch die restlichen Elemente beeinflusst werden.

Oberste Einheit der Schnittstelle sind die Leinwände (Canvas). Sie dienen als Container für die jeweilige Datenstruktur einerseits und die erweiterten Interaktionsmethoden (Capabilities) andererseits. Es können mehrere Leinwände gleichzeitig existieren (z.B. für verschiedene Bilder in einer Bildverarbeitung), was je nach Backend unterschiedlich dargestellt wird (z.B. mehrere Fenster bei GTK, Abschnitte auf der HTML-Seite bei Web).

Die definierten Klassen lassen sich in folgendem Vererbungsbaum darstellen:

Accessible(Implementation)

- Bool(Implementation): logische Werte (False, True)
- Integer(Implementation): Ganzzahlen (unbegrenzt)
- Double(Implementation): IEEE 754 double precision floating-point
- IntegerRange(Implementation): Ganzzahlen in einem bestimmten Intervall
- DoubleRange(Implementation): Fließkommazahlen in einem bestimmten Intervall
- String(Implementation): Zeichenketten
- Struct(Implementation)
  - Canvas(Implementation): Leinwand
- List(Implementation)
  - ModifiableList(Implementation): Liste mit veränderbarem Inhalt
  - ReadOnlyList(Implementation): Liste mit unveränderlichem Inhalt

- Command(Implementation): vom Benutzer aktivierbarer Befehl

Die Accessible-Klassen dienen zur Definition der Benutzerschnittstelle und werden nur an `CanvasImplementation.define()` übergeben. Als solche sind sie frei kopierbar (z.B. um mehrere DoubleRange-Elemente mit dem gleichen Intervall zu definieren) und enthalten keinerlei Möglichkeit zur Daten-Manipulation.

Die AccessibleImplementation-Klassen dagegen ermöglichen die Verwendung der Benutzerschnittstelle. Sie enthalten Methoden zum Abfragen und Setzen von Werten (`getv()`, `setv()`) sowie von Attributen (`geta()`, `seta()`). Listen enthalten zudem Methoden zum optimierten Zugriff (`isEmpty()`, `len()`, `append()`, usw.).

Attribute sind Eigenschaften von Elementen, auf die das Programm nicht verzichten kann (z.B. Callback bei Änderung; Verstecken oder Verschleiern der Eingabe für Passwörter). Im Gegensatz dazu sind Angaben in Style Sheets lediglich Entscheidungshilfen, um die Benutzerschnittstelle dem Benutzer optimal anzupassen.

CanvasImplementation enthält in der Funktion als Container für Datenstrukturen und Capabilities noch weitere Methoden:

- `define(name, type)`
  Definieren eines Elements vom Typ `type` mit dem Namen `name`.

- `defineConst(name, type)`
  Kurzform von `define()` + `seta('constant', True)`

- `get(name)`
  Liefert das Element (nicht den Wert) mit dem Namen `name` zurück.

- `getcap(capability)`
  Liefert ein Handle für die die beim Aufruf von `create_canvas()` angeforderte Capability `capability` zurück. Die Art des Handle hängt von der Capability ab.

- `realise()`
  Ende der Definitionsphase, Leinwand kann nun dem Benutzer präsentiert werden.

- `clear()`
  Bisher definierte Benutzerschnittstelle löschen, Leinwand jedoch nicht zerstören. Zur sofortigen Wiederverwendung der Leinwand mit neuer Benutzerschnittstelle (z.B. nächste Eingabemaske in einem Wizard) gedacht.

- `destroy()`
  Leinwand vollständig entfernen. Aufruf anderer Methoden ist danach nicht mehr erlaubt.

2.1.2 Initialisierung

Folgenden Zeitabschnitte lassen sich beim Programmablauf unterscheiden:

1. Globale Initialisierung, Auswahl des Backends, Lesen der Style Sheets
2. Definition der (ggf. ersten) Benutzerschnittstelle
3. Realisierung der (ersten) Benutzerschnittstelle
4. Übergabe der Kontrolle an DAU
5. Reaktion auf Benutzereingaben, incl. Beenden und Erstellen weiterer Benutzerschnittstellen
6. Programmende

Die globale Initialisierung erfolgt mit Hilfe der Funktion `init(name, datadir, args)`. Dabei werden Programmname, Datenverzeichnis (für Style Sheets) und Kommandozeilenparameter mitgeteilt. Entsprechend der Wünsche des Benutzers (über Umgebungsvariablen, ansonsten fest einprogrammierte Reihenfolge) und den Fähigkeiten der aktuellen Umgebung (z.B. X11, Textkonsole) wird ein passendes Backend ausgewählt und initialisiert.

Die Definition der Benutzerschnittstelle beginnt mit dem Aufruf der Funktion `create_canvas(name, capabilities, related)` zum Erzeugen der ersten Leinwand (`related` ist in diesem Fall `none`). `capabilities` gibt die gewünschten zusätzlichen Fähigkeiten an.

Innerhalb dieser Leinwand kann nun eine Datenstruktur definiert werden, die der Interaktion mit dem Benutzer dient. Dazu werden Instanzen von Accessibles an die Methode `define()` übergeben, welche dann eine Instanz der jeweils passenden AccessibleImplementation-Klasse liefert:

```
canvas  = dau.create_canvas('main', [], None)
bool1   = canvas.define('ex.bool1', dau.Bool())
bool1.seta("changed", cb)
double1 = canvas.define('ex.dbl1', dau.Double()).seta("
    changed", cb)
```

In diesem Beispiel wird eine Datenstruktur mit einem logischen Wert (`ex.bool1`) und einem Fließkommawert (`ex.dbl1`) definiert. `define()` liefert eine Instanz der entsprechenden AccessibleImplementation-Klasse zurück. Mittels `seta("changed", cb)` wird DAU angewiesen, bei Änderungen der Daten durch den Benutzer die Funktion `cb()` aufzurufen.

Da `seta()`, `setv()`, etc. die verwendete Instanz als Rückgabewert liefern, ist eine Kurzschreibweise als Aufrufkette möglich, wie die Definition von `ex.dbl1` zeigt.

Die Realisierung der Benutzerschnittstelle wird mittels Aufruf von `realise()` erreicht. Damit wird die Benutzerschnittstelle fertiggestellt; Änderungen an der Datenstruktur selbst sind nun nicht mehr möglich, lediglich der Inhalt kann verändert werden. Erst nach diesem Schritt kann der Benutzer die Schnittstelle verwenden.

Die Übergabe der Kontrolle an DAU über die Funktion `mainloop()` ist nötig, da viele Widget Sets die Kontrolle über die Hauptschleife des Programms haben möchten. Die Funktion `mainloop()` kehrt erst bei Programmende zum Aufrufer zurück.

Die Nutzung der Schnittstelle ist nun möglich. Änderungen durch den Benutzer und Befehlsaufrufe werden der Applikation mittels der definierten Callback-Funktionen mitgeteilt, woraufhin das Programm selbst wieder Änderungen vornehmen kann. Manche Backends (derzeit nur GTK) können auch von nebenläufigen Prozessen (Threads) vorgenommene Modifikationen an den Benutzer weiterreichen (z.B. Fortschrittsbalken bei einem Installer). Beispiel für eine Reaktion auf Benutzereingaben:

```
def message(canvas, msg, finishfn):
  f = canvas.getcap('output-cstream')
  f.write(msg)
  canvas.define('seq.finish()', dau.Command()).setv(
      finishfn)
  canvas.realise()

def helloWorldMsg(canvas, command):
  userName = canvas.get('params.user').getv()
  canvas.clear()
  message(canvas, 'Hello '+userName+'!', finish)
```

```
canvas = dau.create_canvas("helloworld", ['output-
    cstream'], None)
canvas.define('params.user', dau.String()).setv(os.
    getenv('USER'))
canvas.define('seq.next()', dau.Command()).setv(
    helloWorldMsg)
```

Sobald der Benutzer den Befehl `seq.next()` aktiviert, wird die Funktion `helloWorldMsg()` aufgerufen. Diese holt den (möglicherweise vom Benutzer modifizierten) Wert von `parameters.user`, bereitet die Leinwand auf die Wiederverwendung vor, gibt über die Capability `output-cstream` einen Gruß an den Benutzer aus und lässt die Leinwand dann nach Definition eines Programmende-Befehls dem Benutzer präsentieren.

2.1.3 Backends

Derzeit stehen die folgenden Backends zur Verfügung:

- gtk
  Grafische Oberfläche auf Basis von GTK [GTK]. Derzeit umfangreichste Implementierung.

- stream
  Prototyp einer rein Textzeilen-basierten Benutzerschnittstelle. Könnte in Verbindung mit Braille-Zeile und/oder Sprachausgabe für Blinde gut geeignet sein. Prinzipbedingt nur Unterstützung für sehr einfache Capabilities.

- web
  Webserver. Darstellung und Interaktion erfolgen über einen normalen Webbrowser. HTML-Formular-basiert, deshalb Darstellung von programmgenerierten Modifikationen erst nach Benutzerinteraktion. Viele Anwendungen nehmen Änderungen nur als Reaktion auf Benutzeraktionen vor; für diese ist die derzeitige Implementierung ausreichend.

2.1.4 Capabilities

Folgende erweiterte Interaktionsmethoden (Capabilities) können genutzt werden:

- input-strings (gtk, stream, web)
  Eingabe von Textzeilen (Übertragung an Applikation nur zeilenweise).
- input-chars (nicht implementiert)
  Eingabe einzelner Zeichen (sofortige Übertragung an Applikation).
- output-cstream (gtk, stream, web)
  Ausgabe von Texten (z.B. Log-Meldungen).
- output-image (gtk, web)
  Anzeigen von Bildern. Unterstützte Formate sind Backend-spezifisch.
- output-cairo (gtk, web)
  Zeichenfläche auf Basis der Cairo-Bibliothek [cai]. Elementare Zeichenoperationen, Text.
- output-cairo-cmap (gtk, web)
  Wie output-cairo, jedoch mit Unterstützung für Benutzerinteraktion über rechteckige Bildelemente (analog zu den client-side image maps in HTML [W3Cb])

3 Beispiele

3.1 Hello world

Im Anhang A ist der Source Code eines einfachen „Hello World“-Programms aufgeführt. Bedienung und Programm sind trivial und teilweise bereits erläutert worden, deshalb wird hier nicht näher darauf eingegangen.

3.2 Kassensystem mit Barcode-Scanner

Eines der beiden bisher mit DAU entwickelten Programme ist ein einfaches Selbstbedienungs-Terminal mit Barcode-Scanner, das sich mittlerweile im produktiven Einsatz befindet.

Steuerung und Identifikation des Benutzers erfolgt dabei wahlweise über Tastatur oder Barcode-Scanner; Produkte werden nur über letzteren identifiziert. Um die Barcodes vom Scanner zu lesen (Anschluß über serielle Schnittstelle), werden Threads verwendet. Über Style Sheets wurde das Layout des „Aktions“-Canvas optimiert, der Dreh- und Angelpunkt für den Einkauf bzw. die Pfandrückgabe darstellt (Bild 3).

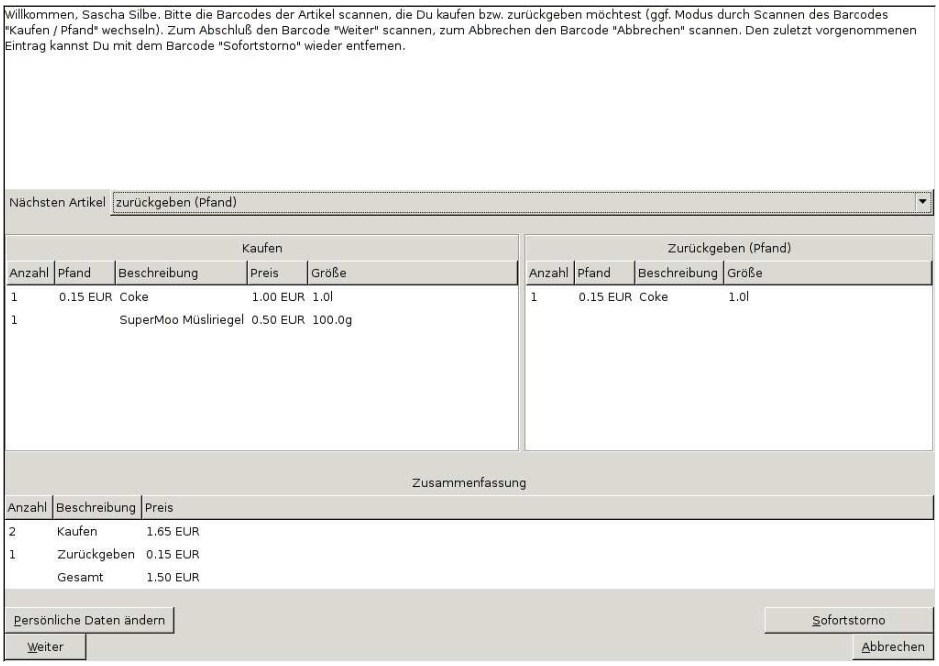

Abbildung 3: Kassensystem auf Basis von DAU: der „Aktions“-Canvas für Kauf und Pfandrückgabe

3.3 GPS-Navigationssystem

Das zweite DAU-nutzende Programm ist ein Navigationssystem für PCs mit GPS-Empfänger (Bild 4). Dargestellt werden können Rasterkarten von Expedia sowie Vektorkarten von OpenStreetMap [osm] (beide via `output-cairo` oder – falls unterstützt – `output-cairo-cmap`). Die Navigationsfunktion ist kommerziellen Navigationssystemen noch unterlegen, aber für Testzwecke ausreichend.

Die Datenbankabfrage findet in einem eigenen Thread statt, damit die Position weiterhin aktualisiert werden kann. In einer kleinen Testreihe (mit Expedia-Rasterkarten) war die CPU-Auslastung meßbar besser als die des Konkurrenz-Produkts GPSDrive. [gps]

4 Vergleich mit anderen Systemen

Sowohl Literatur als auch World Wide Web zeigen verschiedene mit DAU verwandte Projekte und Techniken. [Lau95] gibt eine Übersicht über einige bekannte Verfahren. Im Folgenden werden diese (und weitere) vorgestellt und in ihrer Zielsetzung mit DAU vergleichen. Mehrere weitere Systeme werden

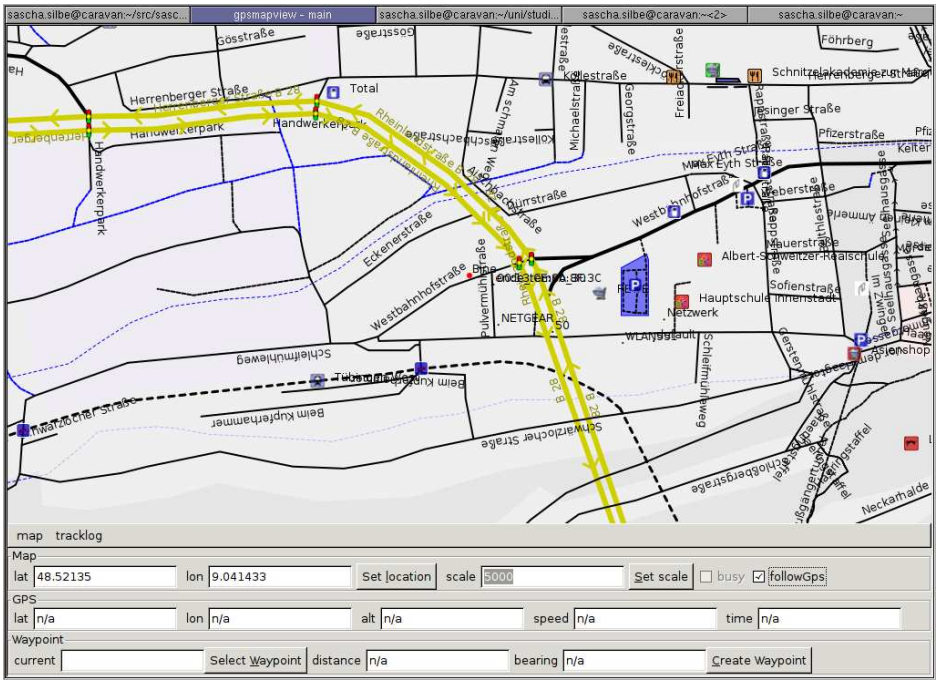

Abbildung 4: GPS-Navigationssystem auf Basis von OSM und DAU

wegen Ähnlichkeit zu bereits vorgestellten [PEGM94] oder offensichtlich unterschiedlichem Ansatz [BVVB93] nicht weiter betrachtet .

4.1 Java Server Faces

Java Server Faces (JSF, [jsf]) ist ein Java Beans-basiertes Framework für Webapplikationen. In Java Server Pages eingebettete UI components werden mit Datenmodellen (Beans) verbunden, die eine Art Mischung zwischen Accessibles und Capabilities von DAU sowie Layout-Containern darstellen.

JSF ist klar auf Web-Applikationen fokussiert, die UI components stellen lediglich eine geringfügige Abstraktion über HTML-Formulare dar. Da die Beans nur auf einem Java EE Server laufen, ist ein Einsatz auf dem Desktop selbst bei Beschränkung auf Bedienung per Webbrowser nicht ohne weiteres möglich.

4.2 Coupling Application Design and User Interface Design

In [dBFMdM92] wird eine Methode zur halbautomatischen Erzeugung von Benutzerschnittstellen für Datenmodelle beschrieben. Mit Hilfe von D2M2edit

erzeugte Datenmodelle werden analysiert und daraus an Hand von Regeln Vorschläge für Benutzerschnittstellen erstellt. Ein UI-Designer kann dann die passenden Widgets auswählen und die Benutzerschnittstelle erzeugen lassen.

Während de Baar in seinem Paper einerseits einen Schritt weitergeht und die Datenstruktur aus einer formalen Beschreibung des Datenmodells erzeugt, ermöglicht DAU andererseits die vollautomatische Erzeugung von Benutzerschnittstellen zur Laufzeit. Insbesondere ist es möglich, Backends zu verwenden, die bei Programmerstellung nicht zur Verfügung standen.

4.3 GENIUS

Janssen, Weisbecker und Ziegler [JWZ93] erzeugen aus einem erweiterten Entity-Relationship-Datenmodell, Definition von Partitionen des Datenmodells (sog. Views) und Interaktionsmustern (sog. Dialogue Nets) mit Hilfe eines Expertensystems (Knowledge Base besteht Objekten des Widget Sets sowie Regeln zur Auswahl und Layout der Widgets) passenden Code zur Steuerung des Widget Sets.

Ähnlich dem im vorherigen Abschnitt beschriebenen System setzt GENIUS direkt auf einer formalen Beschreibung des Datenmodells auf, die in diesem Fall jedoch eher auf Datenbank-basierte Systeme ausgerichtet ist. Die Verwendung des (regelbasierten) Expertensystems lässt sich – wenn auch zur compile-time statt zur run-time – mit den Style Sheets von DAUvergleichen. Allerdings zielt das System eher auf die Erzeugung einer einzelnen, speziell angepassten Benutzerschnittstelle ab. Die Auswahl zur Laufzeit ist nicht vorgesehen und für bei der Programmerstellung nicht bekannte Backends ist keine Code-Erzeugung möglich. Zudem ist eine Anpassung an die Bedürfnisse eines einzelnen Benutzer nur durch Modifikation und Neuübersetzung des Programms nötig.

5 Analyse

Die Arbeit am Prototyp hat gezeigt, daß die Erstellung dieser Abstraktionsschicht zwar eine umfangreiche, aber dennoch machbare Aufgabe ist. Fehler in den verwendeten Bibliotheken [Gno] bzw. getesteten Applikationen [KDE, Mozb, Moza] haben einige wertvolle Stunden gekostet und die Optik ist durchaus verbesserungsfähig, aber die realisierten Applikationen (Kasse, Navigationssystem) zeigen, daß selbst auf den ersten Blick recht spezielle Systeme problemlos von DAU Gebrauch machen können.

5.1 Pro & Kontra

Vorteile:

- Backends müssen während Programmerstellung nicht bekannt sein
- Erstellung eines neuen Backends ermöglicht sofortige Nutzung aller DAU-basierenden Applikationen mit der neuen Benutzerschnittstelle, somit ideal zur Entwicklung neuer Systeme z.B. für Personen mit Behinderungen
- Anpassung an die individuellen Bedürfnisse des einzelnen Benutzers möglich
- das gleiche Programm lässt sich sowohl als Webapplikation als auch als Desktopapplikation betreiben (derzeit nur eingeschränkt)
- bei vergleichbarem Programmieraufwand Unterstützung von mehreren Benutzerschnittstellen statt eines einzelnen Widget Sets
- einheitliches Look&Feel für alle DAU-basierenden Applikationen beim selben Benutzer
- Verwendung der selben Anwendung von unterschiedlichen Benutzern mit divergierenden Ansprüchen möglich, dadurch Synergieeffekte

Nachteile:

- kein (garantierter) Einfluß auf Darstellung und Benutzerinteraktion
- kein einheitliches Aussehen bei unterschiedlichen Benutzern
- (noch) keine automatische Erzeugung des UI code aus einer formalen Beschreibung
- Übersetzung zwischen Datenmodell und Benutzerschnittstelle zur Laufzeit kostet Ressourcen, dadurch möglicherweise für eingebettete Systeme nicht geeignet
- bisher nur wenige Backends vorhanden

5.2 Einsatzgebiete

DAU bietet sich derzeit besonders für kleinere Anwendungen an, die keine speziellen Anforderungen an die Benutzerschnittstelle haben oder sich diese Anforderungen in überschaubarer Zeit implementieren lassen (z.B. OpenGL). Mittelgroße Projekte wie z.B. ein Mail User Agent (MUA) könnten bei ausreichender Geduld oder Implementierungseifer ebenfalls profitieren. Insbesondere dürfte es langfristig zu einer grösseren Verbreitung beitragen.

Nicht geeignet ist DAU für Fälle, in denen Layout und Bedienweise genau festgelegt werden sollen. Hier ist die direkte Verwendung eines plattformübergreifenden Widget Sets wie GTK oder QT sinnvoller. In Bezug auf die Benutzerschnittstelle Performance-kritische Applikationen wie z.B. Webbrowser sind derzeit ebenfalls besser beraten, auf andere Lösungen zurückzugreifen, da die bisher einzige unterstützte generische Zeichenfläche (Cairo) – auch ohne DAU – leider noch recht langsam ist.

6 Zusammenfassung

Alleinstellungsmerkmal von DAU ist die an den individuellen Benutzer angepasste Generierung einer Benutzerschnittstelle zur Laufzeit sowie die (noch nicht vollständig implementierte) Möglichkeit, das selbe Programm sowohl als Web- als auch als Desktop-Applikation verwenden zu können.

Zwar ist noch einiges zu tun, bis das System für „die breite Masse“ tauglich ist, dennoch zeigte sich, daß das Vorhaben mit vertretbaren Aufwand umzusetzen und DAU bereits jetzt sinnvoll verwendbar ist.

7 Ausblick

Obwohl DAU bereits im Produktiveinsatz ist, gibt es noch sehr viele Verbesserungs- und Erweiterungsmöglichkeiten. Insbesondere das Web-Backend ist derzeit sehr eingeschränkt. Neben offensichtlichen Ergänzungen wie AJAX zur Unterstützung nebenläufiger Applikationen und feingranularer Rückmeldungen an die Anwendung (derzeit werden Änderungen erst nach Aktivierung von Submit- oder Command-Buttons übertragen) dürfte eine interessante Aufgabe der Umbau zu einer Multi-Session-fähigen Lösung, also einer „echten“ Webapplikation werden. Weiterhin sind bessere Algorithmen zur Layout-Bestimmung und Unterstützung für speziellere GTK-Widgets wie Datei- oder Schriftarten-Auswahl angebracht. Um das gesetzte Ziel der Abstraktion über verschiedene Benutzerschnittstellen tatsächlich (nicht nur probeweise)

zu erreichen, sind weitere Backends (QT, SAA, etc.) nötig. Eine Namenskonvention für häufig verwendete Datenelemente nebst dazugehörigen Default-Stylesheets werden sich ebenfalls erst mit der Zeit entwickeln.

Alles in allem ist noch viel zu tun, bis DAU wirklich für eine große Zahl von Anwendern tauglich ist. Nichtsdestotrotz ist der erste Schritt erfolgreich getan und die nächsten sind bereits in Vorbereitung.

A Quelltext zu Hello world

```
#!/usr/bin/env python
import sys
import os
import os.path
import dau

def message(canvas, msg, finishfn):
  f = canvas.getcap('output-cstream')
  f.write(msg)
  canvas.define('seq.finish()', dau.Command()).setv(
     finishfn)
  canvas.realise()


def helloWorldMessage(canvas, command):
  userName = canvas.get('parameters.user').getv()
  canvas.clear()
  message(canvas, 'Hello '+userName+'!', finish)

def finish(canvas, command):
  print "Finished."
  sys.exit(0)


myName = sys.argv[0]
params = sys.argv[1:]
dau.init(os.path.basename(myName).replace('.py', ''),
    os.path.join(os.path.dirname(myName), 'dau-data'),
    params)
canvas = dau.create_canvas("helloworld", ['output-
    cstream'], None)
user = canvas.define('parameters.user', dau.String()).
    setv(os.getenv('USER'))
canvas.define('seq.next()', dau.Command()).setv(
    helloWorldMessage)

canvas.realise()
dau.mainloop()
```

Literatur

[BVVB93] Bodart, Francois, Jean M. V, Jean M. V und Franois Bodart: *Encapsulating knowledge for intelligent automatic interaction objects selection*. Seiten 424–429. ACM Press, 1993.

[cai] *Cairo 2D graphics library homepage*. `http://cairographics.org/`, [Online; Zugriff am 18. Juli 2008].

[dBFMdM92] Baar, Dennis J. M. J. de, James D. Foley, Kevin E. Mullet und Charles A.van der Mast: *Coupling application design and user interface design*. Technischer Bericht DUT-TWI-92-03, Delft, The Netherlands, 1992. `http://citeseer.ist.psu.edu/baar91coupling.html`.

[Gno] Gnome Bugzilla: *GTK+ bug #314926: GtkSpinButton is not listening for GtkEntry changes*. `http://bugzilla.gnome.org/show_bug.cgi?id=314926`, [Online; Zugriff am 12. März 2008].

[gps] *Homepage von GPSDrive*. `http://www.gpsdrive.de/`, [Online; Zugriff am 20. Juli 2008].

[GTK]

[jsf] *JavaServer Faces Technology Homepage*. `http://java.sun.com/javaee/javaserverfaces/index.jsp`, [Online; Zugriff am 20. Februar 2008].

[JWZ93] Janssen, Christian, Anette Weisbecker und Jürgen Ziegler: *Generating user interfaces from data models and dialogue net specifications*. In: *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, Seiten 418–423, New York, NY, USA, 1993. ACM Press, ISBN 0-89791-575-5.

[KDE] KDE Bugzilla: *Konqueror bug #148343: SVG images in HTML pages served via HTTP are cut off*. `https://bugs.kde.org/show_bug.cgi?id=148343`, [Online; Zugriff am 12. März 2008].

[Lau95] Lauridsen, Ole: *Abstract specification of user interfaces.* In: *CHI '95: Conference companion on Human factors in computing systems*, Seiten 147–148, New York, NY, USA, 1995. ACM, ISBN 0-89791-755-3.

[Moza] Mozilla Bugzilla: *Firefox bug #288276: The width and height of SVG embeded by reference isn't overriden.* `https://bugzilla.mozilla.org/show_bug.cgi?id=288276`, [Online; Zugriff am 15. März 2008].

[Mozb] Mozilla Bugzilla: *Firefox bug #321531: SVG in ¡object¿ doesn't size right.* `https://bugzilla.mozilla.org/show_bug.cgi?id=321531`, [Online; Zugriff am 15. März 2008].

[osm] *Homepage von OpenStreetMap.* `http://www.openstreetmap.org/`, [Online; Zugriff am 20. Juli 2008].

[PEGM94] Puerta, Angel R., Henrik Eriksson, John H. Gennari und Mark A. Musen: *Beyond Data Models for Automated User Interface Generation.* In: *In Proc. British HCI'94*, Seiten 353–366. University Press, 1994.

[SWI] *Homepage von SWIG (Simplified Wrapper and Interface Generator).* `http://www.swig.org/`, [Online; Zugriff am 20. Juli 2008].

[Val04] Valkonen, Tuomo: *Vis/Vapourware Interface Synthesiser*, April 2004. `http://modeemi.fi/~tuomov/vis/vis-paper.pdf`, [Online; Zugriff am 3. Januar 2008].

[W3Ca] W3C World Wide Web Consortium: *Cascading Style Sheets, level 2 - CSS2 Specification.* `http://www.w3.org/TR/1998/REC-CSS2-19980512`, [Online; Zugriff am 19. Juli 2008].

[W3Cb] W3C World Wide Web Consortium: *HTML 4.01 Specification.* `http://www.w3.org/TR/1999/REC-html401-19991224`, [Online; Zugriff am 19. Juli 2008].