Fachrichtung 6.2 – Informatik
Naturwissenschaftlich-Technische Fakultät I
Universität des Saarlandes

Eine GTK-Schnittstelle für Alice

Robert Grabowski
grabow@ps.uni-sb.de
Programming Systems Lab

Fortgeschrittenen-Praktikum

Angefertigt unter der Leitung von
Prof. Dr. Gert Smolka

Betreut von
Thorsten Brunklaus

16. Mai 2003

Abstract

In diesem Bericht wird die GTK+-Schnittstelle für Alice beschrieben. Mit GTK+ [1] lassen sich relativ einfach grafische Benutzeroberflächen (GUIs) in C erstellen. Durch die Schnittstelle ist dies nun auch dem Alice-Programmierer möglich.

Im Folgenden werden Entwurf und Implementierung der Schnittstelle vorgestellt. Es werden Probleme bei der Abbildung der GTK+-Bibliotheken auf Alice erörtert und Entwurfsentscheidungen diskutiert. Dabei wird auch auf allgemeine, nicht GTK-spezifische Konzepte beim Schnittstellen-Design hingewiesen.

Weiterhin werden die wichtigsten Punkte bei der Implementierung beschrieben und die Funktionsweise des Schnittstellen-Generators erläutert, bevor der Bericht mit einem Ausblick auf weiterführende Arbeiten schließt.

Inhaltsverzeichnis

1 Einleitung

Dieser Abschnitt stellt die Schnittstelle, GTK+ und das GnomeCanvas kurz vor. Es werden allgemeine Prinzipien der GUI-Programmierung und der Aufbau von GTK+ besprochen.

1.1 Die Schnittstelle

Als Grundlage für diese Arbeit dienen zwei bereits existierende Schnittstellen für GTK+ 1.2: eine Anbindung an Mozart, und eine an Alice auf der Mozart-VM [2]. Die hier besprochene Schnittstelle bindet GTK+ 2.x an Alice auf SEAM [3]. Sie ist in der Benutzung weitestgehend kompatibel mit der bereits existierenden Alice-Anbindung. Die größten Unterschiede liegen bei GTK+ selbst, bedingt durch den Versionssprung.

Alle erwähnten Schnittstellen sind sowohl unter Windows als auch unter Linux lauffähig. Benutzt werden die GTK+-Schnittstellen unter anderem von dem Inspector-Tool [4].

1.2 GTK+

GTK+ ist eine Bibliothekssammlung zum komfortablen Erstellen von grafischen Benutzeroberflächen. Ursprünglich im Jahr 1996 nur zur Verwendung im Grafikprogramm GIMP geschrieben[1], bildet es inzwischen die Grundlage sehr vieler Anwendungen, insbesondere des GNOME-Desktops. GTK+ ist komplett in C geschrieben, doch bei der Entwicklung wurde von Anfang an darauf geachtet, möglichst gute Voraussetzungen für die Anbindung an andere Programmiersprachen zu schaffen.

Ein weiterer, wichtiger Punkt für die Festlegung auf diese GUI-Bibliothek ist das GnomeCanvas, welches weiter unten vorgestellt wird.

[1] GTK steht für "GIMP Toolkit".

Konzepte

GTK+ basiert auf folgenden grundlegenden Konzepten:

- *Widgets.* Ein Widget ist ein grafisches Objekt, das ein Element der Benutzeroberfläche darstellt, wie Fenster, Schaltflächen und Menüs. Widgets sind in einem Pseudo-Klassensystem hierarchisch angeordnet und besitzen Attribute, die z.B. Aussehen und Verhaltensweisen beeinflussen.
- *Ereignisgesteuerte Hauptschleife.* Diese überprüft periodisch, ob Ereignisse vorliegen, wie z.B. Mausbewegungen oder Tastatureingaben. Für jedes eingetretene Ereignis ruft die Hauptschleife entsprechende Event-Handler auf.
- *Ereignisbehandlung.* Die Ereignisbehandlungsfunktionen, auch "Event-Handler" oder "Callbacks" genannt, führen beim Eintreten eines Ereignisses bestimmte Aktionen aus. Neben einer Reihe von vordefinierten Standard-Handlern bietet GTK+ dem Programmierer die Möglichkeit, eigene Callback-Funktionen zu registrieren und an bestimmte Ereignisse zu binden.
- *Speicherverwaltung.* Nicht mehr benötigte Objekte werden von GTK+ automatisch entfernt.

Aufbau

GTK+ 2.0 besteht aus den folgenden Bibliotheken:

- GLib – grundlegende Funktionen wie Klassensystem und Speichermanagement
- GDK – plattformunabhängige Abstraktion der Grafikfunktionen des Betriebssystems
- Pango – abstrahiert Schriftartenfunktionen des Betriebssystems
- ATK – stellt Funktionen für Eingabehilfen zur Verfügung
- GTK – enthält die eigentliche Sammlung von Widgets

In diesem Alice-GTK+-Binding werden nur Schnittstellen für die Bibliotheken GTK und GDK generiert, denn auf die anderen Bibliotheken muss in der Regel beim Programmieren von GTK+-Anwendungen nicht direkt zugegriffen werden.

GTK und GTK+

Obwohl die Sammlung der Bibliotheken "GTK+" heißt und "GTK" nur eine einzelne Bibliothek ist, wird im Folgenden aus Gründen der Vereinfachung und Lesbarkeit immer nur von "GTK" gesprochen, womit das gesamte Paket gemeint ist.

1.3 GnomeCanvas

Das GnomeCanvas ist ein Widget, das eine abstrakte Fläche zum Zeichnen von Texten, Bildern und primitiven Objekten darstellt. Es basiert wesentlich auf dem TkCanvas-Widget aus der Tcl/Tk-Bibliothek [5].

Die GnomeCanvas-Bibliothek ist nicht Teil von GTK+, sondern gehört bereits zum Gnome-Projekt. Eine Schnittstelle ist dennoch unbedingt erforderlich, da diverse Alice-Tools wie der Inspector zentral auf diesem Widget basieren.

2 Entwurf

Im Folgenden werden die Entwurfsentscheidungen erläutert, die bei der Entwicklung der Schnittstelle getroffen wurden. Neben grundsätzlichen Überlegungen zur Anbindung von C-Bibliotheken in Alice wird erklärt, wie gewisse GTK-spezifische Probleme speziell in dieser Schnittstelle gelöst wurden.

2.1 Alice-Schnittstellen für C-Bibliotheken

Eine C-Bibliothek besteht im Wesentlichen eine Reihe von Funktionen. Zusätzlich werden komplexe Datentypen deklariert, die bei der Verwendung mit den Funktionen sinnvoll sind. Ein Binding hat die Aufgabe, diese Funktionen und Datentypen dem Alice-Programmierer zur Verfügung zu stellen.

Funktions-Abbildung

Weil Bibliotheksfunktionen nicht direkt von Alice aufrufbar sind, wird für alle Funktionen ein Wrapper geschrieben, der in Alice sichtbar ist und der den eigentlichen Funktionsaufruf tätigt.

Ein Wrapper muss zudem ein Datenmarshalling durchführen, d.h. vor dem eigentlichen Funktionsaufruf Alice-Werte in C-Werte umwandeln und umgekehrt. Die genaue Arbeitsweise hängt natürlich davon ab, wie mit C-Datentypen in Alice umgegangen wird.

Datentyp-Abbildung

Es gibt grundsätzlich zwei Möglichkeiten für die Datentyp-Abbildung.

Eine Möglichkeit besteht darin, das gesamte C-Typsystem durch eine Reihe von abstrakten Alice-Typen darzustellen. In diesem Fall muss die Schnittstelle aber Operatoren für Werte dieser abstrakten Typen zur Verfügung stellen, wie etwa Zuweisungs- und Vergleichsoperatoren für C-Integers. Dieser Ansatz wird beispielsweise durch Matthias Blumes "Foreign Function Interface" für SML implementiert [6]. Zwar können auf diese Weise alle Aspekte des C-Typsystems sehr genau abgebildet werden, jedoch wird durch den indirekten Zugriff auf C-Werte die Programmierung unter Alice deutlich erschwert.

Als Alternative könnte man versuchen, für alle C-Typen ein möglichst ähnliches Alice-Pendant zu finden. Das erleichert zwar den Umgang mit der Schnittstelle, allerdings entsteht dadurch im Binding ein ineffizienter Overhead für die ständige Datenkonversion. Vor allem aber gibt es nicht immer ein passendes Typ-Äquivalent in Alice.

Das GTK-Binding geht einen Mittelweg zwischen beiden genannten Möglichkeiten: primitive Datentypen und Enumerationstypen werden auf ähnliche Alice-Typen abgebildet, während Verbundtypen in Alice abstrakt bleiben. Gründe für diese Entscheidung und Implementationsdetails werden in Abschnitt 3 erklärt.

Die automatische Schnittstellengenerierung

Sobald man sich auf eine bestimmte Abbildung festgelegt hat, liegt es nahe, das Binding anhand der Headerdateien der C-Bibliothek automatisch generieren zu lassen. Leider enthalten reine C-Deklarationen manchmal zuwenig semantische Informationen, um eine möglichst akkurate Alice-Schnittstelle erzeugen zu können.[2] Weil auch dies auch bei GTK der Fall ist

[2] So ist der C-Quellcode beispielsweise bei der Unterscheidung von Ein- und Ausgabeparametern nicht selbsterklärend.

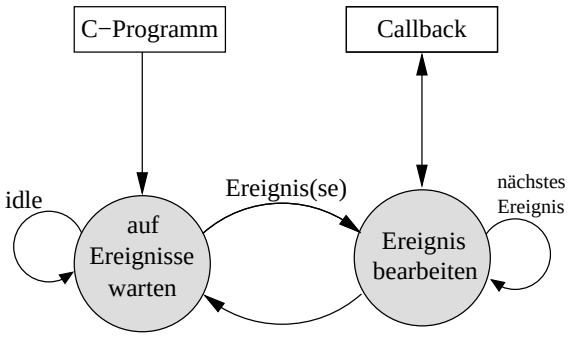

Abbildung 1: Ereignisbehandlung in C

und eine Annotierung mit Meta-Daten nicht vorliegt, macht der Generator an einigen Stellen bestimmte Annahmen, die zumindest für die GTK-Bibliotheken zutreffen. Außerdem kann für Einzelfälle eine gesonderte Code-Generierung festgelegt werden.

Die so generierte Alice-Schnittstelle ist im Wesentlichen eine 1:1-Abbildung der C-Bibliothek. Die GTK-Programmierung in Alice besteht daher zum größten Teil auf imperativen Funktionsaufrufen, die Seiteneffekte hervorrufen können. Der Ausblick am Ende dieses Berichts erläutert Ideen für eine höhere, funktionale Schicht.

Die direkte Abbildung auf unterer Ebene bringt noch ein weiteres Problem mit sich: Wenn die Bibliothek in Alice genauso benutzt wird wie in C, können unter Umständen fundamentale Alice-Konzepte wie die Nebenläufigkeit ausgehebelt werden. Daher müssen einige Teile der Schnittstelle auf jeden Fall von Hand geschrieben werden. Bei GTK sind dies die Hauptschleife, die Ereignisbehandlung sowie die Garbage Collection, welche in den nächsten Abschnitten erklärt werden.

2.2 Hauptschleife und Ereignisbehandlung

Im Folgenden wird kurz beschrieben, welche Prinzipien einem GTK-Programm unter C zugrunde liegen und warum eine direkte Übertragung nach Alice nicht möglich ist. Daraufhin wird erläutert, wie die Probleme der Ereignisbehandlung in Alice gelöst wurden.

GTK-Programmierung in C

Abbildung 1 zeigt den Kontrollfluss eines in C geschriebenen GTK-Programms. Dabei laufen die folgenden Schritte ab:

1. Es werden sogenannte “Callback-Funktionen” bei GTK registriert, die bei bestimmten Ereignissen reagieren sollen.

2. Das Hauptprogramm übergibt die Kontrolle an die GTK-Hauptschleife, welche erst zum Ende des Programms zurückkehrt.

3. Die Hauptscheife überprüft regelmäßig, ob Ereignisse vorliegen.

4. Wenn ein Ereignis eintritt (z.B. der Benutzer eine Schaltfläche drückt), werden die für dieses Ereignis registrierten Callback-Funktionen aufgerufen.

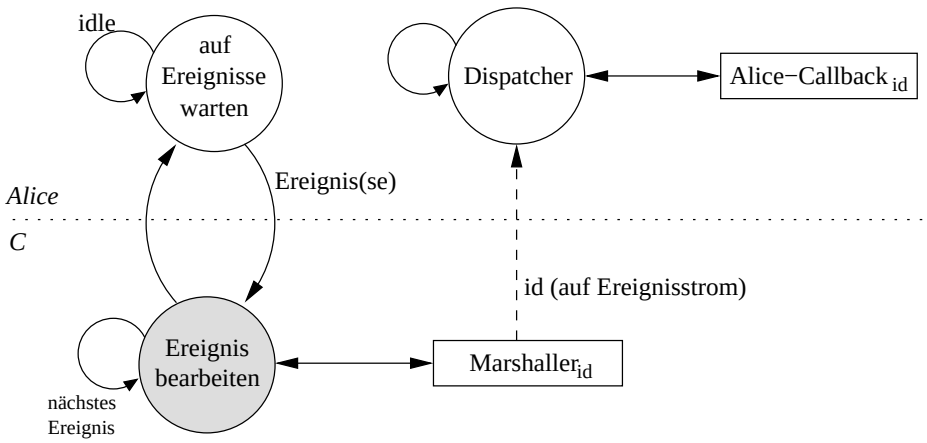

Abbildung 2: Ereignisbehandlung in Alice

Zu beachten ist dabei, dass die Ereignisse synchron abgearbeitet werden, d.h. erst wenn die Callback-Funktion zurückgekehrt ist, kann die Hauptschleife wieder neue Ereignisse abarbeiten.

Der naheliegende Ansatz, dieses Konzept unverändert auf Alice-Programme zu übertragen, muss aus zwei Gründen scheitern. Zum einen können Alice-Callback-Funktionen nicht bei GTK registriert werden, geschweige denn von C aus aufgerufen werden. Zum anderen sind Aufrufe von C-Funktionen für die Alice-VM atomar. Während die Programmkontrolle bei der Hauptschleife liegt, was fast immer der Fall ist, ist die virtuelle Maschine blockiert. Dadurch kann sie keine weiteren Alice-Threads mehr abarbeiten, und die Nebenläufigkeit ist nicht mehr sichergestellt.

Ereignisbehandlung in Alice

Abbildung 2 zeigt, wie die Probleme in Alice gelöst werden.

- Teile der Hauptschleife werden nach Alice verlagert.
- Alice-Callbacks-Funktionen werden indirekt über ein Ereignisstrom-Modell aufgerufen.

Die Hauptschleife wird in einen C- und einen Alice-Teil aufgeteilt.[3] In einem eigenen Alice-Thread wird dabei das Verhalten der GTK-Hauptschleife nachempfunden: Der Thread ruft periodisch GTK-Funktionen auf, die überprüfen, ob Ereignisse vorliegen bzw. diese abarbeiten. Durch diese Konstruktion kehrt der Programmfluss immer wieder zu Alice zurück, wodurch die VM nicht blockiert wird.

Die Realisierung des indirekten Aufrufs der Alice-Callback-Funktion ist etwas komplizierter:

1. Soll eine Funktion für ein Ereignis registriert werden, wird für sie eine eindeutige ID generiert. Bei GTK wird statt der Funktion ein in C geschriebener "generischer Marshaller" registriert, und zwar so, dass er bei jedem Aufruf die ID als Argument erhält.

[3] Diese Aufteilung ist kein "Hack", sondern explizit von den GTK-Entwicklern als Möglichkeit vorgesehen.

2. Wird das Ereignis ausgelöst, ruft GTK den Marshaller mit der ID auf. Dieser schreibt die ID auf den Ereignisstrom.

3. Auf Alice-Seite wartet ein "listener-Thread" auf Daten vom Strom. Dieser Thread kann anhand einer ankommenden ID die zugehörige Alice-Callback-Funktion herausfinden und aufrufen.

Es gibt einen wichtigen Unterschied zur Ereignisbehandlung bei einem C-Programm: Die Ereignisse werden asynchron abgearbeitet, d.h. die Hauptschleife kann bereits ein Ereignis bearbeiten, während ein anderes noch in einer Alice-Callback-Funktion abgearbeitet wird. Durch den Strom werden die Ereignisse werden aber trotzdem nacheinander in der richtigen Reihenfolge abgearbeitet. Für den GTK-Programmierer sind die Änderungen bei der Implementierung der Ereignisbehandlung nicht sichtbar.

2.3 Garbage Collection

GTK+ besitzt eine eigene Speicherverwaltung, die dafür sorgt, das GTK-Objekte freigegeben werden, sobald sie nicht mehr benötigt werden. Dies wird über einen Zähler realisiert, der festhält, wieviele Referenzen auf ein Objekt bestehen. Sobald der Zähler auf 0 sinkt, entfernt GTK das Objekt aus dem Speicher. Dieses Verfahren beruht wesentlich auf dem Algorithmus von Jones und Lins [7].

Das Problem dabei ist, das der Zähler zunächst nur bibliotheksintern verwaltet wird. Nachdem eine GTK-Funktion eine Referenz auf ein Objekt geliefert hat und diese Referenz in Alice sichtbar ist, kann es passieren, dass GTK das Objekt intern nicht mehr benötigt und daher entfernt wird, weil die Bibliothek keine Kenntnis von der Alice-Referenz besitzt. Wird die Alice-Referenz dann weiterhin benutzt, kann das fatale Folgen haben.

GTK bietet jedoch Funktionen an, mit denen der Programmier selbst den Referenzzähler eines Objekts ändern kann. Die Schnittstelle macht sich diese Möglichkeit zunutze und erhöht den Zähler automatisch:

- Sobald eine GTK-Funktion eine Objekt-Referenz liefert und es noch keine Referenz von Alice auf das Objekt gibt, wird der GTK-Referenzzähler um eins erhöht.

- Sobald unter Alice letzte Referenz auf das Objekt verschwindet, wird dessen Zähler wieder um eins verringert.[4]

Damit ist gewährleistet, dass der GTK-Referenzzähler eines Objekts mindestens 1 ist, solange eine Referenz auf das Objekt in Alice sichtbar ist, was GTK am Freigeben des Objekts hindert.

3 Implementierung

In diesem Teil wird der Aufbau der GTK-Schnittstelle und die Funktionsweise des Generators kurz erklärt. Daraufhin wird beschrieben, wie die C-Datentypen konkret in Alice abgebildet wurden, und welchem Schema die Wrapper für die Bibliotheksfunktionen folgen.

3.1 Allgemeiner Aufbau der Schnittstelle

Abbildung 3 zeigt den allgemeinen Aufbau der GTK-Schnittstelle. Sowohl die generierte Schnittstelle als auch der handgeschriebene "Kern" besitzen einen in C implementierten

[4] Die Schnittstelle greift auf den Finalisierungsmechanismus von Alice zurück, um festzustellen, ob noch Alice-Referenzen auf ein Objekt bestehen.

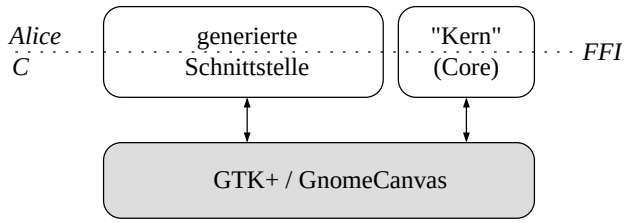

Abbildung 3: Aufbau der Schnittstelle

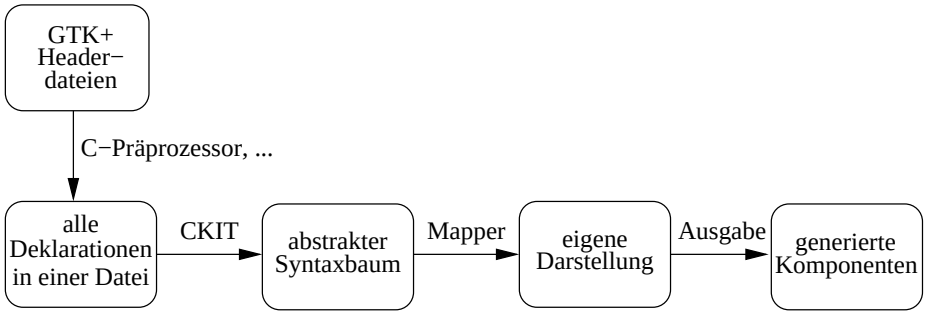

Abbildung 4: Generierungs-Phasen

Teil, der die eigentliche Kommunikation mit der Bibliothek abwickelt, als auch eine in Alice implementierte Schicht, die von der unteren Ebene abstrahiert und u.a. eine höhere Typsicherheit bietet. Für den Zugriff auf die C-Ebene von der Alice-Seite und umgekehrt sorgt das Foreign Function Interface (FFI) von SEAM.

3.2 Generator

Die Generierung läuft in mehreren Phasen ab (siehe dazu auch Abbildung 4):

- Mithilfe des C-Präprozessors und weiterer, auf Textebene arbeitender Programme werden die Deklarationen in allen Header-Dateien der GTK-Bibliotheken in eine Datei zusammengefasst.
- Die C-Deklarationen werden mithilfe der CKIT-Bibliothek[5] geparst und in einem abstrakten Syntaxbaum dargestellt.
- Die CKIT-Darstellung ist für unsere Zwecke aufgrund der reichhaltigen Annotationen sehr unhandlich und ineffizient. Daher wird sie in eine eigene Zwischenrepräsentation umgewandelt, welche genau diejenigen Informationen enthält, die für die Generierung notwendig sind.
- Anhand dieser Repräsentation kann schließlich der eigentliche Schnittstelle (Alice- und C-Code) generiert werden.

[5]Das CKIT [8] kann beliebige C90-konforme C-Programme parsen. Es ist Bestandteil der SML/NJ-Suite.

C-Typ	Alice-Typ	C-Typ	Alice-Typ
`void`	`unit`	`char*`	`string`
(ganzzahliger Typ)	`int`	t`[]`	t `vector`
(Fließkomma-Typ)	`real`	t`*`	`object`
`gboolean`	`bool`	`GList*`/`GSList*`	`object list`

Tabelle 1: Abbildung primitiver Datentypen

3.3 Abbildung der Datentypen

Für welche konkreten Elemente kann das Binding überhaupt eine Schnittstelle liefern? Neben den primitiven Datentypen von C sind dies die in der Bibliothek deklarierten Funktionen, Enumerationstypen, Verbundtypen, Typ-Aliasnamen und externen Variablen.

Primitive Datentypen. In diesem Binding werden primitive C-Datentypen in der Regel auf möglichst ähnliche primitive Alice-Typen abgebildet. Einen Überblick gibt Tabelle 1.

- `object` ist dabei ein abstrakter Datentyp, der einen C-Zeiger repräsentiert. Es ist ohne die GTK-Bibliothek nicht möglich, ein Wert vom Typ `object` zu konstruieren oder zu dereferenzieren, lediglich ein Vergleich ist möglich. Mehr ist auch nicht nötig, denn für Zeiger auf Verbundtypen gibt es Zugriffsfunktionen, und Zeiger auf andere Werte werden als Ausgabeparameter betrachtet (siehe unten).

  Alternativ hätte für C-Zeiger auch ein abstrakter, polymorpher Datentyp "α `pointer`" deklariert werden können. Bedingt durch das Pseudo-Klassensystem von GTK muss der Programmierer aber sehr häufig Typumwandlungen von Zeigern vornehmen, so dass die Schnittstelle unhandlicher geworden wäre, die Typsicherheit sich aber nicht verbessert hätte.[6]

- `GList` und `GSList` sind GTK-Datenstrukturen für die Verwaltung von Listen. Die spezielle Behandlung ist notwendig, da diese Datenstrukturen in der GLib-Bibliothek definiert sind, für die kein eigenes Binding erzeugt wird.

- Bei Strings, Arrays und Listen kann sich in C der Zustand/Inhalt ändern, ohne dass diese Änderungen auf Alice-Seite reflektiert werden. Anhand der Konzeption von GTK kann jedoch davon ausgegangen werden, dass diese Typen nur als "Container" für den Datentransport benutzt werden, und etwaige Änderungen durch eine GTK-Funktion in einem neuen "Container" zurückkommen.

Enumerationstypen. Diese werden auf Alice-Datatypes mit nullstelligen Konstruktoren abgebildet. Dadurch wird eine neue Typsicherheit eingeführt, die unter C nicht vorhanden war, da dort alle Enumerationskonstanten im Grunde Integer-Werte sind. Beispiel:

```
typedef enum {              datatype GtkWindowType =
  GTK_WINDOW_TOPLEVEL,        WINDOW_TOPLEVEL
  GTK_WINDOW_POPUP          | WINDOW_POPUP
} GtkWindowType;
```

Eine direkte Implementierung als Alice-Integerkonstanten (Variablen) wäre zwar in Bezug auf die Laufzeit etwas günstiger als die hier vorliegende Konstruktor-Integer-Konversion.

[6] Die GTK-Funktionen prüfen ohnehin zur Laufzeit, ob die übergebenen Zeiger den richtigen Typ haben.

Es würde dann aber die Typsicherheit fehlen, zudem sind z.B. pattern matchings über Variablen nicht möglich.

Verbundtypen. Bei Verbundtypen[7] liegt es nahe, sie als Alice-Records zu repräsentieren. In C können jedoch Verbunde nicht direkt, sondern immer nur als Verbundzeiger an Funktionen übergeben werden. Änderungen, die GTK über die Zeiger an den Verbund-Daten vornimmt, können im Allgemeinen nicht in den Alice-Records widergespiegelt werden. Stattdessen bietet die Schnittstelle den abstrakte Zeiger-Typ `object`, der für die Benutzung von Verbunden nicht nur notwendig, sondern auch ausreichend ist. Zudem muss der Programmierer GTK-Verbunde nie selbst erstellen oder entfernen, sondern kann entsprechenden Bibliotheksfunktionen benutzen. Die Schnittstelle muss also keine eigene Konstruktoren und Destruktoren zur Verfügung stellen. Allerdings erfordert die GTK-Programmierung bisweilen einen direkten Zugriff auf Felder eines Verbunds. Daher werden für jeden Verbundtyp Feldzugriffsfunktionen generiert.

Typaliasnamen und externe Variablen. Diese werden von der Schnittstelle nicht behandelt. Eine Abbildung von Typaliasnamen auf type-Deklarationen in Alice würde erst richtig Sinn machen, wenn man statt `object` einen polymorphen "α `pointer`" benutzen würde (siehe oben). Ein Zugriff auf externe Variablen ist bei der GTK-Programmierung nicht nötig.

3.4 Wrapper

Für jede Funktion wird ein Wrapper generiert, der das Datenmarshalling durchführt, d.h. Werte eines Alice-Typs in Werte eines C-Typs konvertiert und umgekehrt. Der Alice-Funktionstyp folgt dabei der C-Funktionsdeklaration; mehrere Argumente werden als Tupel übergeben. Die Funktionsnamen sind nach der "lower CAML case"-Konvention umgewandelt. Beispiel:

```
      void gtk_scale_set_draw_value (GtkScale*, gboolean)
⟹    val scaleSetDrawValue : object * bool -> unit
```

Ein- und Ausgabeparameter

Manche GTK-Funktionen nehmen Zeiger auf primitive Typen entgegen:

```
void gtk_entry_get_layout_offsets (GtkEntry*, int*, int*)
```

Mittels solcher Zeiger kann die Funktion zusätzliche Ausgabewerte zurückliefern. Da in Alice mehrere Rückgabewerte (als Tupel) möglich sind, betrachtet die Schnittstelle alle Zeiger mit Ausnahme von Verbundzeigern, Strings und Listen als Ausgabeparameter und erstellt daher den folgenden Wrapper:

```
⟹    val entryGetLayoutOffsets : object -> int * int
```

Möglicherweise greift die GTK-Funktion aber zusätzlich noch lesend auf einen solchen Wert zu. Aufgrund fehlender semantischer Informationen kann der Generator nicht wissen, ob es sich um einen kombinierten Ein- und Ausgabe-Parameter handelt. Daher wird für eine solche Funktion ein weiterer Wrapper generiert, der jeden Ausgabeparametern zusätzlich als Eingabeparameter betrachtet:

[7] Das sind in C "structs" und "unions".

```
⟹  val entryGetLayoutOffsets' : object * int * int -> int * int
```

Es liegt dann am Programmierer, die richtige Variante zu benutzen.

Funktionen mit variablen Argumentlisten

Im Bereich der Sprachanbindungen ist dies einer der problematischsten Punkte. Leider macht auch GTK intensiv von diesem "Feature" Gebrauch.

Funktionen mit variablen Argumentlisten können beliebig viele Argumente erhalten, und jedes Argument kann von einem beliebigen Typ sein. Die Übergabe der Argumente von Alice an die Schnittstelle lässt sich dabei noch halbwegs elegant gestalten. Auf Alice-Seite wird eine Liste vom Typ `arg list` übergeben, wobei `arg` die variablen Typen repräsentiert:

```
datatype arg = INT of int | BOOL of bool | STRING of string | ...
```

Das eigentliche Problem besteht darin, dass zur Übersetzungszeit *der Schnittstelle* Anzahl und Typen der Argumente bekannt sein müssen. Sie werden aber frühestens zur Übersetzungszeit des Alice-Programms bekannt, welches die Schnittstelle benutzt.

Es gibt in C keine plattformunabhängige Möglichkeit, variable Argumentlisten zur Laufzeit zu bauen und einer Funktion zu übergeben. Die GTK-Schnittstelle löst das Problem momentan auf eine Art, die auf x86-Plattformen unter Windows und Linux keine Probleme zu bereiten scheint. Eine sauberere Lösung bieten spezielle low-level-Bibliotheken [9], die das plattformabhängige dynamische Übergeben von variablen Argumenten abstrahieren.

4 Ausblick

Weiterführende Arbeiten könnten sowohl "in die Tiefe" gehen und die GTK-Schnittstelle ausbauen, als auch "in die Breite" und die automatische Generierung verallgemeinern.

Funktionale GUI-Programmierung

Die hier vorgestellte Schnitstelle bietet eine Anbindung auf niedriger Ebene. Durch die 1:1-Abbildung sieht die GTK-Programmierung unter Alice ähnlich aus wie unter C. Diese unter Alice unelegante Programmierweise kann nur mit einer funktionalen, abstrahierenden Schicht über der eigentlichen GTK-Schnittstelle umgangen werden.

Einen Anfang dafür bildet die GtkBuilder-Struktur [10]. Das Layout ganzer Fensterinhalte kann als Wert mittels Konstruktoren repräsentiert werden. Daraufhin genügt der Aufruf einer einzigen Funktion, welche diesen Wert anschließend in eine Reihe von GTK-Befehlen übersetzt.

Wünschenswert wäre es, spezielle Features der Programmiersprache Alice gleich zu nutzen. So könnten z.B. ähnliche Mitteilungsfenster in einem Funktor zusammengefasst werden, und dieser könnte wiederum als Package im Netz zur allgemeinen Verfügung stehen. Weitere Ideen für eine funktionale GUI-Programmierung bietet das Fudgets-Konzept [11].

Allgemeines Generierungstool

Auch wenn der Generator in der jetzigen Form bereits möglichst unabhängig von GTK arbeitet, macht er dennoch einige GTK-spezifische Annahmen, die eine Anwendung für andere Bibliotheken erschweren.

Daher wäre es nützlich, diese Annahmen zu parametrisieren, um einen allgemeinen Generator zu erhalten, der für beliebige C-Bibliotheken eine Anbindung an Alice generieren kann.

Ausgehend von dieser Schnittstelle hat Sven Woop ein solches allgemeines Generierungstool bereits entwickelt, und mit den entsprechenden Parametern konnte damit ebenfalls eine voll funktionsfähige GTK-Schnittstelle erzeugt werden.

Literatur

[1] GTK+ Homepage
`http://www.gtk.org/`

[2] The Mozart GTK+ Binding.
`http://www.mozart-oz.org/documentation/add-ons/gtk/`

[3] Thorsten Brunklaus, Leif Kornstaedt. *A Virtual Machine for Multi-Language Execution*. Universität des Saarlandes, 2002.

[4] Bernadette Blum, Marvin Schiller. *Ein Browser für Alice*. Fortgeschrittenenpraktikum, Universität des Saarlandes, 2003.
`http://www.ps.uni-sb.de/~schiller/inspector.html`

[5] John K. Ousterhout. *An X11 Toolkit Based on the Tcl Language*. In: Proceedings of the 1991 Winter USENIX Conference, University of California, Berkeley, 1991.

[6] Matthias Blume. *No-Longer-Foreign: Teaching an ML compiler to speak C "natively"*. University of Chicago, 2001.
`http://people.cs.uchicago.edu/~blume/papers/nlffi.pdf`

[7] Richard Jones, Rafael Lins. *Garbage Collection. Algorithms for Automatic Dynamic Memory Management.* John Wiley & Sons, 1996.

[8] CKIT Homepage
`http://www.smlnj.org/doc/ckit/`

[9] libffi Homepage
`http://sources.redhat.com/libffi/`

[10] GtkBuilder von Thorsten Brunklaus. Enthalten in den Demonstrationsprogrammen, die zu den beiden GTK-Schnittstellen für Alice gehören.

[11] Magnus Carlsson, Thomas Hallgren. *Fudgets – Purely Functional Processes with applications to Graphical User Interfaces*. PhD Thesis, Chalmers University of Technology, Göteborg, 1998.
`http://www.cs.chalmers.se/~hallgren/Thesis`