Python als Basis für komplexe GUI Anwendungen

Bernhard Herzog

Inhaltsverzeichnis

Einführung

Dieser Vortrag gibt eine Übersicht über den Einsatz der Programmiersprache Python zur Entwicklung umfangreicher und komplexer Applikationen mit grafischer Benutzeroberfläche. Insbesondere geht es um Python als Hauptentwicklungssprache, eventuell kombiniert mit anderen Sprachen wie C oder C++, und nicht darum, Python nur einzusetzen, um eine Skriptsprache in eine bestehende Applikation einzufügen.

Zur Erläuterung werden Beispiele aus zwei Projekten, Sketch [http://sketch.sourceforge.net/] und Thuban [http://thuban.intevation.org/] verwendet.

Sketch ist ein leistungsfähiges Vektorzeichenprogramm für GNU/Linux und eines der umfangreichsten in Python geschriebenen GUI-Programme (mit ca. 50000 Zeilen Pythoncode und 20000 Zeilen C-Code). Sketch läuft unter Unix-ähnlichen Systemen mit X11 und verwendet in der stabilen Version Tkinter (Pythons Tk-Anbindung) und in der Entwicklerversion GTK 2).

Sketch bietet dem Benutzer außer den üblichen Fähigkeiten eines interaktiven Zeichenprogramms die Möglichkeit, eigene Pythonfunktionen zu schreiben und in das Menü einzubinden. Des weiteren hat Sketch ein Plugin-System mit dem man Pythonmodule schreiben kann, die automatisch eingebunden werden und mit denen neue Import/Export Filter und sogar neue Objekttypen implementiert werden können.

Thuban ist ein Anwendung zum Betrachten von geographischen Daten. Die Daten können aus verschiedenen Quellen stammen, wobei zur Zeit Shapefiles und PostGIS-Datenbanken unterstützt werden. PostGIS unterstützung ist noch in der Entwicklung wird aber voraussichtlich im Juni 2003 in Thuban zur Verfügung stehen.

Thuban verwendet wxPython als GUI-Toolkit und läuft unter GNU/Linux (und vermutlich anderen Unix-ähnlichen Systemen) sowie unter MS Windows.

Python

Python ist eine dynamisch typisierte, objektorientierte Programmiersprache. Python wird seit über 10 Jahren beständig weiter entwickelt und erfreute sich besonders in den letzten Jahren wachsender Beliebtheit. Die aktuelle stabile Version ist 2.2.2. Python 2.3 ist zur Zeit (Mai 2003) in Beta.

Da Python als solches nicht Teil des Vortrags ist hier nur eine kurze Übersicht über die Eigenschaften der Sprache mit besonderem Augenmerk auf die Dinge, die für GUI-Programme interessant sind.

Dynamische Typisierung

Python ist dynamisch typisiert. Etwas vereinfacht ausgedrückt heißt das, daß Typinformationen an Objekte und nicht an Variablen geknüpft sind. Typüberprüfungen finden zur Laufzeit statt. Das hat mehrere Konsequenzen:

- Pythoncode ist generisch in dem Sinne, daß eine Funktion oder Methode mit allen Parametern funktioniert, die bestimmte implizite Schnittstellen implementieren. Welche Schnittstellen dies sind ist nur dadurch definiert, was die Funktion mit den übergebenen Objekten macht aber nicht wie sie deklariert ist. In dieser Hinsicht sind Python Funktionen vergleichbar mit C++ Templates.

- Python ist nicht so schnell wie kompilierte statisch typisierte Sprachen. Das ist in der Praxis im Allgemeinen kein grosses Problem, da meist nur relativ kleine Teile eines Programms wirklich laufzeitkritisch sind, und selbst diese häufig schnell genug sind.

  Wenn ein Programm dennoch zu langsam ist, kann man zunächst mit dem Profiler (ein Bestandteil der Standardbibliothek) feststellen, welcher Teil des Codes das Problem ist und dann entweder versuchen, bessere Algorithmen und/oder Datenstrukturen einzusetzen oder, wenn das nicht weiterhilft, den kritischen Teil in C oder C++ schreiben (s. unten).

- Typfehler werden erst zur Laufzeit entdeckt. Typfehler bedeutet hier meist, daß ein Objekt nicht das vom Code erwartete Interface implementiert. In manchen Fällen kann es aber auch sein, daß speziell ein Integer erwartet wird und z.B. ein Float verwendet wurde.

  Für Programmierer, die von statisch typisierten Sprachen her kommen, ist dynamische Typisierung zumindest zu Anfang etwas erschreckend, da ihnen das gewohnte Sicherheitsnetz der Typprüfung zur Compilezeit fehlt. Wenn man allerdings gute Unittests hat, findet man diese Fehler in der Regel kurz nachdem der fehlerhafte Code geschrieben wurde. Man sollte auch bedenken, daß statische Typprüfungen nur einen Teil der möglichen Fehler eines Programms finden können und man daher auch in statisch typisierten Sprachen nicht auf (möglichst automatische) Tests verzichten sollte.

Automatische Kompilierung

Python muß nicht explizit übersetzt werden. Damit entfällt ein separater Übersetzungsschritt und man kann beim Programmieren nach einer Änderung sein Programm einfach sofort neu starten.

Flexible Standarddatentypen

Die eingebauten Datentypen Dictionary (eine Hashtabelle), Listen (eigentlich ein Array oder Vektor) sowie Strings (sowohl als Bytestrings als auch als Unicode Objekte) erleichtern viele Aspekte der Programmentwicklung.

Da diese Datentypen Standardbestandteil von Python sind, sind ihre Implementationen im Laufe der Zeit sehr gut optimiert worden. Dies gilt insbesondere für die Dictionaries, mit denen fast alle Namensräume implementiert. Ausserdem verfügt die Sprache über spezielle Syntax um die Objekte zu erzeugen und zu verwenden:

```
unicode_string = u"Unicode"

langer_string = """Ein langer String,
der sich über mehrere Zeilen
erstreckt"""
```

```
# eine Liste mit drei Elementen
liste = [1, 2, 10]

# Ein dictionary, das "a" auf 1 abbildet und das Tupel (1, 2) auf 4
dictionary = {"a": 1, (1, 2): 4}

# Eine "Listcomprehension" baut ein Liste aus einer Sequenz und
# einem Ausdruck (hier die ersten 10 Quadratzahlen):
quadratzahlen = [x ** 2 for x in range(10)]
```

Hohe Produktivität

Alle bisher aufgeführten Eigenschaften kombiniert führen zu einer hohen Produktivität der Programmierer, besonders im Vergleich zu C oder C++.

Mit der dynamischen Typisierung und der damit verbundenen, eher generischen Programmierweise reicht es meist, sich auf die Schnittstellen seiner Objekte und die Operationen auf ihnen zu konzentrieren. In statischen Sprachen muss man sich hingegen schon recht früh auf sehr bestimme Datentypen festlegen. Wo man in Python z.B. zunächst mal einfach an "Zahl" denkt muss man sich in C sofort für "int", "float" oder "unsigned int" oder ähnliches entscheiden. Spätere Änderungen ziehen dann häufig viele Änderungen an anderen Stellen nach sich.

Die leistungsfähigen und flexiblen Datentypen ermöglichen es, auf einer abstrakteren Ebene zu arbeiten und sorgen für relativ kompakten aber dennoch lesbaren Code und kurze Entwicklungszeiten, da man sich um viele Details nicht kümmern muss.

Die kürzeren Implementationszeiten ermöglichen es, im gleichen Zeitraum eine größere Zahl verschiedener Algorithmen und Datenstrukturen für die Lösung eines Problems auszuprobieren, so daß man häufig zu einer besseren Lösung gelangt, als mit C oder C++ im gleichen Zeitraum möglich gewesen wäre.

C-API

Python lässt sich leicht durch C oder C++ Code erweitern. Hauptanwendungsgebiete von solchen Erweiterungsmodulen sind Anbindungen an externe Bibliotheken oder Optimierung.

Die C-API ermöglicht alle üblichen Operationen auf Python-Objekten und erlaubt es, von Python aufrufbare Funktionen zu schreiben und auch neue Datentypen für Python zu definieren. Seit Python 2.2 ist es auch möglich, Typen zu implementieren, die man in Python wie eine Klasse beerben kann.

Die C-API ist bis auf wenige Ausnahmen sehr konsistent und, etwas C und Python-Erfahrung vorausgesetzt, leicht zu verwenden. Die einzige schwierigere Hürde kann es sein, Pythons Garbage-Collector Mechanismus (reference counting) richtig zu verwenden.

Es zwar nicht wirklich schwierig, Erweiterungsmodule von Hand zu schreiben, aber es ist deutlich mühseliger als äquivalenten Python-Code zu schreiben, da z.B. Reference-Counting und Fehlerbehandlung explizit implementiert werden müssen. Um sich etwas Arbeit zu sparen, kann man in manchen Fällen diesen Code auch generieren lassen. Übliche Werkzeuge dazu sind u.a. SWIG [http://www.swig.org/] (Simplified Wrapper and Interface Generator) mit dem Schwerpunkt auf die Anbindung von C- und C++-Bibliotheken and diverse Interpreter, darunter Python, Perl und Tcl, sowie das neuere Pyrex [http://www.cosc.canterbury.ac.nz/~greg/python/Pyrex/] das mit einer Python-ähnlichen Sprache Erweiterungsmodule für Python erstellt.

Python Bibliotheken

Standardbibliothek

Eine Stärke von Python, unabhängig von der eigenlichen Sprache, ist die umfangreiche Standardbibliothek. Sie enthält Module für XML, Sockets, Internetprotokolle (http, ftp, xmlrpc, smtp, pop, nntp, u.a.), Parser/Converter

für diverse (internet-) Datenformate u.a. für Emails (MIME, RFC822, einige Mailbox formate), base64, hex, quoted printable.

GUI Toolkits

Es gibt Anbindungen von Python an eine ganze Reihe verschiedener GUI-Toolkits. Hier wollen wir vor allem diejenigen vorstellen, die für Freie Software am wichtigsten sind, nämlich Tkinter, PyGTK und gnome-python, PyQT und PyKDE sowie wxPython.

Tkinter

Tkinter ist die Anbindung an Tk. Der Hauptvorteil von Tkinter ist, daß es bestandteil der Standardbibliothek ist. Als Entwickler kann man also davon ausgehen, daß es entweder bereits beim Benutzer installiert (es ist zum Beispiel Standardbestandteil des Windows Installers von Python) oder es nicht schwer für den Benutzer ist, es nachträglich zu installieren (die meisten GNU/Linux-Distributionen enthalten ein Tkinter-paket).

Tkinter ist außerdem sehr stabil. Wenn ein Programm Tkinter nicht richtig benutzt, gibt es praktisch immer eine Python-Exception. Speicherzugriffsfehler und ähnliche Probleme sind extrem selten.

Das Angebot an GUI-Elementen, das Tkinter mitbringt ist aus heutiger Sicht etwas mager, so gibt es z.B. kein "Tree-View" für die Darstellung von Baumstrukturen. Es gibt aber Bibliotheken die solche Elemente in Python implementieren, z.B. PMW [http://pmw.sourceforge.net/].

Was von manchen als Nachteil empfunden wird, ist, daß Tkinter nicht nur an Tk selbst gebunden ist, sondern dazu einen kompletten Tcl-Interpreter verwendet. Sie sehen Tkinter daher als aufgebläht und langsam an. Wir können das so nicht nachvollziehen. Sketch, zum Beispiel, das es momentan in vergleichbarer Funktionalität sowohl in einer Tkinter- als auch einer PyGTK-Variante gibt, braucht mit GTK 2.2 deutlich mehr Speicher als mit Tk. Beide Versionen fühlen sich subjektiv auch etwa gleich schnell an.

wxPython

wxPython ist die Anbindung an wxWindows. wxWindows ist eine plattformunabhängige GUI Bibliothek die auf den normalen plattformspezifischen Bibliotheken aufsetzt und damit jeweils ein zur Plattform gehörendes Look&Feel hat.

wxWindows bietet einerseits Standardelemente wie Menüs, Dialoge, Buttons, etc. und auch komplexere Widgets wie etwa das Grid-Widget für Tabellen. wxWindows bietet darüber hinaus aber auch ein Applicationframework und plattformunabhängige Schnittstellen für Dinge, die nicht unmittelbar mit grafischen Benutzeroberflächen zu tun haben, wie etwa Drucken oder das Starten von Subprozessen.

PyGTK und gnome-python

PyGTK ist die Anbindung an GTK+. Es ist Teil von gnome-python, der Anbindung an die Gnome-Bibliotheken. GTK selbst ist im wesentlichen nur ein Toolkit, d.h. es besteht aus den normalen GUI-Elementen sowie einigen Standarddialogen. Drucken und andere darüber hinausgehende Dienste können über gnome-python genutzt werden.

Zu den von pygtk unterstützten Plattformen gehört neben GNU/Linux und anderen Unix-ähnlichen Systemen auch MS Windows.

PyQt und PyKDE

PyQt und PyKDE sind, wie die Namen schon vermuten lassen, die Anbindungen and Qt und KDE. Ähnlich wie wxWindows bietet auch Qt die Möglichkeit zum Drucken und andere Features, die nicht direkt an die GUI gebunden sind.

Ein Nachteil von Qt gegenüber den anderen hier erwähnten Toolkits ist, daß es nur in der Version für das X Window System Freie Software ist. Insbesondere ist die Version für MS Windows nicht frei. PyQt folgt im wesentlichen dem gleichen Lizenzmodell wie Qt und ist nur für die GPL-Version von Qt Freie Software

Auswahl der "richtigen" Bibliothek

Welche der vielen Bibliotheken für graphische Benutzeroberflächen, die mit Python genutzt werden können, man verwenden sollte hängt natürlich immer vom Projekt ab. Daher können wir hier nur recht allgemeine Hinweise geben.

Ein Applikation-Framework ist für ein Python-Programm nicht unbedingt wichtig. Viele Teile eines Applikation-Frameworks, wie zum Beispiel Klassen um das Model-View-Controller-Konzept zu implementieren sind in Python recht einfach selbst zu schreiben (s.u.).

Schnittstellen zu nicht-GUI Funktionalitäten wie etwa Datenbanken sind in der Regel schon unabhängig von GUI-Toolkits für Python verfügbar und spielen daher aus der Sicht eines Pythonprogrammierers eigentlich keine Rolle. wxPython zum Beispiel verzichtet daher auch auf eine Anbindung der Datenbankschnittstelle wxODBC.

Eine Ausnahme bildet hier das Drucken, da man zumindest unter Windows dem Benutzer die Standarddialoge zur Druckerauswahl und -konfiguration anbieten sollte. In dieser Hinsicht sind wxWindows und Qt interessant.

Entwicklungsstrategien für Python GUI-Programme

Trennen von Oberfläche und Applikation

Die wichtigste Strategie bei der Entwicklung von GUI Applikationen im Allgemeinen ist eine weitgehende Trennung der eigentlichen grafischen Benutzeroberfläche von der Applikationslogik.

Diese Trennung ist in gewisser Weise eigentlich nur ein Spezialfall einer grundlegenderen Strategie, nämlich die Kopplung zwischen verschiedenen Teilen eines Softwaresystems so lose wie möglich zu gestalten.

Vorteile einer solchen Trennung:

- Konzentration der Logik an einer Stelle
- Einfacheres Testen

  Dialoge und andere GUI-Elemente lassen sich meist nur schwer automatisch testen. Wenn die GUI von der Applikationslogik gut getrennt ist und damit die GUI-Schicht sehr dünn lässt sich ein Großteil der Applikation automatisch testen.

- Einfacherer Wechsel des GUI Toolkits

  Ein Wechsel des Toolkits ist zwar sicherlich ein eher seltenes Ereignis im Leben eines Programms, aber auch nicht so selten.

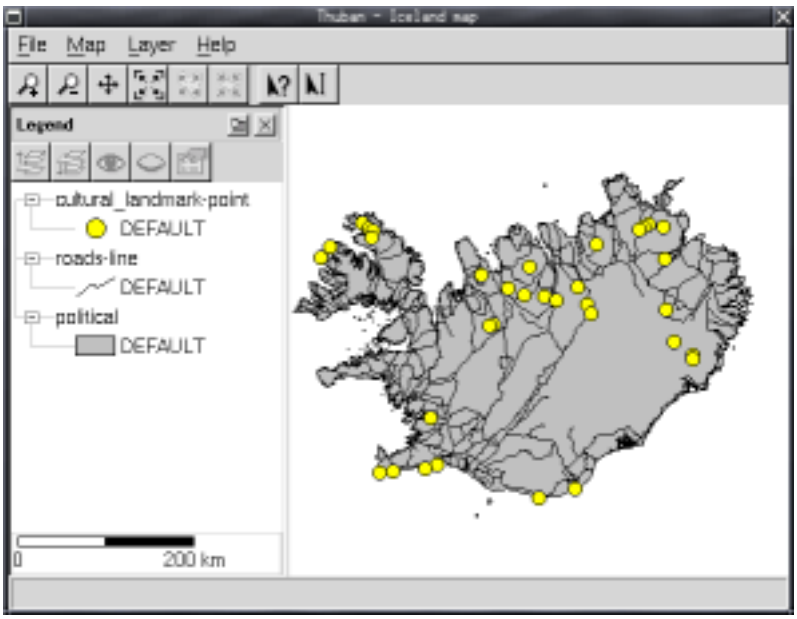
  Sketch zum Beispiel verwendete am Anfang eine Pythonanbindung an Xt (die X-Toolkit Intrisics) und die Athena Widgets und wechselte nach einer Weile zu Tkinter, das in der stabilen Version immer noch verwendet wird. Die Entwicklerversion verwendet mittlerweile GTK2. Bei der Portierung war eine noch weitergehende Trennung von GUI und Logik ein sehr wichtiger Schritt.

Model-View-Controller

Das Model-View-Controller (MVC) Konzept und seine Geschwister Model-View-Presenter und die Observer Design-Pattern sind wohl die bekanntesten Konzepte bei der Trennung von GUI und Logik. Die Observer-Pattern ist allerdings nicht unbedingt auf GUIs bezogen.

Die Kernidee hinter MVC ist, daß die Logik im Modell implementiert ist, der View lediglich den Zustand (eines Teils) des Modells darstellt und der Controller Benutzereingaben wie z.B. Mausclicks entgegennimmt und in Aufrufe von Methoden des Modells umsetzt.

In Thuban zum Beispiel ist das Modell die Karte mit den verschiedenen Ebenen von Daten. Es gibt mehrere Views, unter anderem die Karte im Hauptfenster und die Legende:



Der Maßstab unten links ist natürlich auch ein View, aber nicht auf die Karte selbst, sondern auf den Skalierungsfaktor der bei der Darstellung der Karte im rechten Teil des Fensters verwendet wird. In einem Programm gibt es also meist nicht nur ein Modell sondern mehrere und Views können selbst wieder Modelle sein.

Das Modell muss dabei natürlich die Views immer dann informieren, wenn sich sein Zustand ändert, so daß sich die Views auf den neuesten Stand bringen können. Um die Kopplung zwischen Modell und View lose zu halten, sollte das Modell aber nicht wissen müssen, welche Views dies sind. In Sketch und Thuban wird diese lose Kopplung durch einen in Python implementierten Benachrichtigungsmechanismus erreicht.

Das Konzept hinter diesem Benachrichtigungsmechanismus ist, daß es meist einige Objekte gibt, die Nachrichten senden und viele Empfänger solcher Nachrichten. Die Sender sind normalerweise von einer gemeinsamen Basisklasse, `Publisher` abgeleitet. `Publisher`-Instanzen veröffentlichen gewissermassen Nachrichten und die Empfänger abonnieren sie indem sie mit den Methoden `Subscribe` und `Unsubscribe` ihr Abo verwalten. Eine Publisherinstanz verschickt nachrichten über ihre `issue` Methode.

Als Beispiel hier eine Methode aus Thubans `Map` Klasse. Ein Kartenobjekt hat in Thuban ein Projektion, durch die definiert ist, wie die Oberfläche der Erde auf eine Ebene abgebildet wird. Die `SetProjection` Methode weist eine neue Projektion zu:

```
def SetProjection(self, projection):
   self.projection = projection
        self.issue(MAP_PROJECTION_CHANGED, self)
```

Hier wird also zunächst einfach die Projektion an die entsprechende Instanzvariable gebunden und dann eine `MAP_PROJECTION_CHANGED` Nachricht verschickt. Die Nachricht hat einen Parameter, nämlich die Karte selbst so daß Empfänger Nachrichten von verschiedenen Karten auseinanderhalten können.

Für die Empfängerseite hier ein Beispiel aus der View-Klasse:

```
def SetMap(self, map):
        if self.map is not None:
           self.map.Unsubscribe(MAP_PROJECTION_CHANGED,
                           self.projection_changed)
   self.map = map
        if self.map is not None:
           self.map.Subscribe(MAP_PROJECTION_CHANGED,
                    self.projection_changed)
```

Die `SetMap` Methode weist dem View eine neu darzustellende Karte zu. Zuerst muß also die Benachrichtigung bei der eventuell vorhandenen alten Karte abbestellt werden (mit `Unsubscribe`) und nachdem die Instanzvariable gesetzt wurde wird die Nachricht der neuen Karte abonniert. Der Code hier ist recht stark vereinfacht, da es unter anderem noch einige andere Nachrichten gibt, auf die der View reagieren sollte.

Der erste Parameter von `Subscribe` und `Unsubscribe` ist der Nachrichtentyp. Der zweite ein aufrufbares Objekt -- hier eine gebundene Methode. Wenn eine Nachricht gesendet wird, wird das aufrufbare Objekt mit den Parametern der Nachricht aufgerufen.

Für (noch) nicht Pythonprogrammierer bedarf der Begriff "gebunden Methode" vielleicht noch einer Erläuterung. Ein Methodenaufruf sieht in Python sehr ähnlich aus wie in Java oder C++:

```
objekt.methode(arg1, arg2)
```

Bei der Ausführung dieses Methodenaufrufs wertet Python zunächst den Ausdruck `objekt.methode` aus. Das liefert, wenn `"methode"` tatsächlich der Name einer Methode des Objekts `objekt` ist, ein gebundenes Methodenobjekt. Ein solches Objekt enthält eine Referenz auf das Objekt, an das es gebunden ist, und eine Referenz auf das Funktionsobjekt, das die Methode implementiert. Beim Aufruf des Methodenobjekts wird dann die Funktion mit dem Objekt als erstem Parameter aufgerufen.

In unserem Fall ist die Methode folgende:

```
def projection_changed(self, *args):
   self.FitMapToWindow()
```

Das, heißt, wir ignorieren alle Argumente (`args`, ein Tuple das alle Parameter der Methode bis auf self enthält, wird in der Methode nirgends mehr verwendet) und passen dann die Skalierung der Kartendarstellung so an, daß die ganze Karte ins Fenster passt. `FitMapToWindow` sorgt auch dafür, daß die Karte neu gezeichnet wird.

Nun kann es sehr leicht passieren, daß durch eine Benutzeraktion mehrere Nachrichten vom Modell verschickt werden, die alle ein Update des Fensters verlangen. Wenn das Neuzeichnen sehr aufwändig ist, wie hier im Falle von Thuban, wo hinter einer Karte viele Megabytes an Daten stehen können, ist es wichtig, das Zeichnen etwas aufzuschieben, bis das Programm wieder auf Ereignisse des Fenstersystems wartet. Die meisten Toolkits bieten hierfür sogenannt Idle-Events an.

Undo

Gute Anwendungen sollten den Benutzer nach Möglichkeit davor bewahren, nicht zu behebende Fehler zu machen. Eine der aus Anwender angenehmsten Lösungen ist es, zu Erlauben, Änderungen rückgangig zu machen.

Bei der Implementation der Undo-Funktionalität muss für jede Benutzeraktion festgehalten werden, was sich am Dokument in welcher Weise ändert, um beim Zurücknehmen der Änderungen den vorherigen Zustand wieder herstellen zu können. Was das "Dokument" ist hängt von der Art des Programms ab. In einer Tabellenkalkulation ist es z.B. eine Sammlung von Tabellen

Wir wollen uns nun anschauen, wie Sketch dieses Problem löst. Das Dokument in Sketch ist eine ganze Vektorzeichnung. Die Zeichnung besteht aus übereinander gezeichneten Ebenen die wiederum aus verschiedenen Arten von Objekten bestehen. Die Objekte können Text, Rastergraphiken, Linien, und so weiter sein, aber auch Gruppen die wiederum solche Objekte enthalten. Darunter können auch Objekte sein, die nicht direkt Teil von Sketch sind, weil sie durch Plugins implementiert werden.

Sketch muß nun in der Lage sein, Änderungen an all diesen Objekten festzuhalten und rückgangig machen zu können.

Der Lösungsansatz besteht darin, die Objekte selbst dafür sorgen zu lassen daß die richtige Undo-Informationen vorhanden sind. Das geschieht in Sketch dadurch, daß Methoden, die die Objekte verändern diese Informationen zurückgeben.

Als Beispiel hier eine Methode aus einem Plugin-Objekt das regelmässige Vielecke implementiert. Dies Objekte besitzen einen Radius (mathematisch betrachtet der Radius des Umkreises) den der Benutzer definieren kann. Dabei wird die `SetRadius`-Methode aufgerufen:

```
def SetRadius(self, radius):
    # undoinfo erzeugen
    undo = self.SetRadius, self.radius

    # Instanzvariable setzen
    self.radius = radius

    # interne Datenstrukturen updaten...

    # undoinfo an den Aufrufer zurückgeben.
    return undo
```

Die Undo-Informationen, in Sketch meist einfach Undoinfo genannt, ist einfach ein Tupel aus einem aufrufbaren Objekt, gefolgt von eventuellen Parametern. Das aufrufbare Objekt ist hier wieder eine gebundene Methode. Die Radiusänderung zurückzunehmen bedeutet, wieder den alten Radius zu setzen, also verwenden wir die `SetRadius`-Methode selbst und den alten Radius als parameter.

Der Aufrufer wird in der Regel die Undoinfo Objekte die er bekommt in einem neuen Undoinfo-Objekt zusammenfassen und an seinen Aufrufer übergeben. Eine Ausnahme bilden die Funktionen, die Benutzeraktionen implementieren. An diesen Stellen wird das Undoinfo Objekt an das Dokument übergeben.

Wenn die Aktion, die durch ein Undoinfo-Objekt repräsentiert wird, zurückgenommen wird, wird einfach das aufrufbare Objekt mit den Parametern aufgerufen.

Wenn man sich das Beispiel genauer anschaut, erkennt man, daß hier beim Ausführen des Undo wieder ein Undoinfo-Objekt erzeugt wird. Diesesmal kann man es verwenden, um das Undo selbst rückgängig zu machen, man kann es also für das "Redo" verwenden. Daher ist es in Sketch eine Auflage für das aufrufbare Objekt im Undoinfo-Objekt, beim Aufruf wiederumg ein Undoinfo-Objekt zurückzugeben.

Ein Nachteil dieses Konzepts ist, daß der Aufrufer für die richtige Behandlung der Undoinfo-Objekte verantwortlich ist. Besser wäre es wahrscheinlich, wenn die Objekte selbst die Undoinfo-Objekte an das Dokument übergeben. In Sketch wäre das in jedenfalls kein großes Problem, da die meisten Objekte eine Referenz auf das Dokumentobjekt besitzen, zu dem sie gehören.

Schlußbemerkungen

Es gibt noch viele andere Beispiele aus Sketch oder Thuban, die die Verwendung von Python für GUI-Porgramme demonstrieren können:

- Plugins, um Applikationen durch Zusatzmodule zu erweitern
- Benutzerdefinierte Pythonskripte

  In einem Programm, das selbst in Python geschrieben ist, haben die solche Skripte automatisch Zugriff auf alle interessanten Objekte des Programms, so daß ein sehr leistungsfähiges Skriptinterface mit wenig Aufwand implementiert werden kann.

Auf diese Möglichkeiten auch noch einzugehen, würde den Rahmen des Vortrags sprengen, aber da sowohl Thuban als auch Sketch Freie Software sind, können Sie natürlich in den Quelltext schauen und daraus für ihre eigenen Programme lernen.