Proseminar Python - Python Bindings

Sven Fischer
Student an der TU-Dresden
Sven.Fischer@mailbox.tu-dresden.de

Abstract

Diese Arbeit beschäftigt sich damit, einen Einblick in die Benutzung von externen Bibliotheken in Python zu geben und führt den Leser in das Schreiben von eigenen Erweiterungen ein. Darüber hinaus wird darauf eingegangen, wie Projekte in anderen Programmiersprachen Python benutzen können. Weiterhin wird kurz auf verschiedene andere Möglichkeiten eingegangen, Python oder Pythons Benutzung zu erweitern und zu verändern: durch Kompilation, „Mischen“ mit oder Übersetzen in anderen Sprachen, oder spezielle Interpreter.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—Modules, packages

General Terms Languages, Documentation

Keywords Python, Extension, Embedding, Library

1. Einführung

Python wird mit einer umfangreichen Standartbibliothek ausgeliefert. Trotzdem gibt es Gründe, warum man Python um verschiedene externe Bibliotheken erweitern möchte. Ein Beispiel dafür wäre die Geschwindigkeit, die bei Python als interpretierter Sprache nicht immer den Anforderungen entspricht. Weiterhin gibt es genügend Software von Drittanbietern, welche man in Python nutzbar machen will - ohne sie in Python zu übersetzen. Darauf gehe ich in Abschnitt 2 ein.

Python ist eine flexible Sprache, was schon allein durch die dynamische Typisierung gegeben ist. Man kann vergleichsweise viel Funktion pro Codezeile realisieren, wäre es da nicht nützlich, diese Funktionalität - zusammen mit der großen Standardbibliothek - in einer bereits existierenden Applikation einfügen zu können? Abschnitt 3 beschäftigt sich damit Python in Projekte in anderen Sprachen einzubetten.

Zur besseren Unterscheidung sind beide Möglichkeiten in Abbildung 1 grafisch dargestellt. Im Grunde sind sich beide Verfahren sehr ähnlich: man führt jeweils 3 Schritte aus, zum *Erweitern* oder *Extending*:

1. Daten von Python nach C konvertieren
2. C Funktion mit konvertierten Werten aufrufen
3. Ergebnis von C zurück nach Python konvertieren

Zum *Einbetten* oder *Embedding* sind es die folgenden:

[Copyright notice will appear here once ’preprint’ option is removed.]

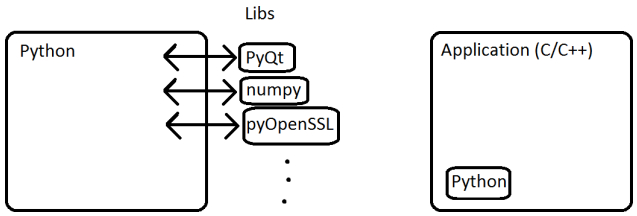

Abbildung 1. Vergleich der Möglichkeiten Python zu erweitern (links) und Python einzubetten (rechts).

1. Daten von C nach Python konvertieren
2. Python Funktion mit konvertierten Werten aufrufen
3. Ergebnis von Python zurück nach C konvertieren

Diese Konvertierung ist auch das größte Hindernis beim Verknüpfen von Python und C[1], zumindest ist es mit einigem Aufwand verbunden.

Es gibt einige Projekte, die sich mit dem Verändern und Erweitern der Sprache Python beschäftigen. Einen kurzen Überblick über einige Möglichkeiten gebe ich in Abschnitt 4. Dort gehe ich auf zwei Python-Interpreter neben dem in der Standard-Distribution enthaltenen ein und zeige Möglichkeiten auf, Python in andere Sprachen zu übersetzen, beziehungsweise direkt in Maschinencode zu kompilieren.

Zum Abschluss gebe ich noch einen kurzen Ausblick auf eine Sprache mit ähnlichen Eigenschaften und Anwendungsmöglichkeiten wie Python: *LUA*.

2. Extending Python

Python eignet sich als Skriptsprache mit hoher Ausdrucksfähigkeit für die Verwendung als „Glue-Language“ zwischen verschiedenen Bausteinen einer größeren Applikation. Dabei werden bereits existierende Teile, die in verschiedenen Programmiersprachen geschrieben sind, miteinander verknüpft.

Wenn man ein großes Softwareprojekt erstellen will, wird man laut [Brooks], gewollt oder ungewollt, einen Prototypen programmieren, den man nur dazu verwendet logische Probleme aufzudecken und anschließend weg wirft. Python kann dabei helfen, die Kosten dieses Prototypen zu verringern beziehungsweise diesen schneller zu erstellen. Man kann auch das nachfolgende „ausgereifte“ System wieder in Python schreiben und dann aufgrund von Geschwindigkeitsvorteilen oder anderen Anforderungen wie Typsicherheit, die Anwendung ganz oder in Teilen in andere Program-

[1] Erweiterbar ist Python um dynamische Bibliotheken, die in beliebigen Sprachen geschrieben sein können. Einbettbar ist es im Grunde auch in Anwendungen, die in beliebigen Sprachen (C#, Java, ...) geschrieben wurden, allerdings bietet Python von Haus aus die C-API an, die die Zusammenarbeit mit C wesentlich erleichtert.

miersprachen auslagern. Wohlgemerkt muss man das nicht, sondern kann auch die finale Anwendung in 100% Python ausliefern. Wenn man sich die Möglichkeit einer Portierung offen lassen will, sollte man allerdings vorsichtig mit der Verwendung von speziellen Python-Konstrukten (wie Method-Patching) umgehen, oder man muss diese später in der Zielsprache geeignet umschreiben.

Listing 1. Benutzung einer Funktion fak aus einem importierten Modul modExample

```
1 import modExample
2 x = 3
3 r = modExample.fak(3)
```

Um Erweiterungen zu installieren reicht es meist, einen Installer bzw. Linux-Paket auszuführen, oder ein mitgeliefertes Setup-Skript zu starten (via `$ python setup.py install`). Erweiterungen sollten möglichst transparent zu benutzen sein, das heißt genauso wie ein normales Paket (siehe zum Beispiel Listing 1).

Das Problem am Erweitern von Python mit Bibliotheken liegt in den Unterschieden von Python zu anderen Programmiersprachen oder kompilierten Bibliotheken: Die Datentypen sind andere, Datentypen werden in Python teilweise automatisch konvertiert und es steht vor der Ausführung nicht fest, welchen Typ eine Variable an einem bestimmten Punkt hat. Es gibt Unterschiede in der Speicherverwaltung (Heap-Allokation und Reference Counting gegen teilweise Stack-Allokation und Speicherverwaltung „per Hand“) und in den Funktionsschnittstellen (benannte Parameter). Diese Unterschiede können entweder in der Bibliothek selbst (bzw. in C-Wrappern) oder in Python überbrückt werden, beide Wege werden vorgestellt.

2.1 ctypes

Mittels des Moduls ctypes ist ein Zugriff von Python auf dynamische Bibliotheken (als DLL- oder SO-Dateien) möglich, ohne Änderungen an der Bibliothek vornehmen zu müssen. Sämtliche Konvertierungen werden in Python gehandhabt.

Listing 2. Import und Aufruf der Microsoft Implementierung der C-Standard-Bibliothek

```
1 import ctypes
2 bibliothek = ctypes.CDLL("MSVCRT")
3 bibliothek.printf("Hallo Welt\n")
```

Listing 2 zeigt ein Beispiel, welches aber viele Probleme verschweigt: einfache Konvertierungen erledigt Python automatisch, sobald mehr Datentypen konvertiert werden müssen, als Strings oder Integer, muss man selbst Hand anlegen.

Listing 3. Aufrufen einer Funktion mit Float-Parameter und setzen des Rückgabetyps einer anderen

```
1 f = ctypes.c_float(1.337)
2 bib.callFloat(f)
3
4 bib.veclenDouble.restype = c_double
5 print bib.veclenDouble(c_double(1.5), c_double (2.7), ↩
      c_double (3.9))
```

Listing 3 zeigt am Beispiel, wie man einen Bibliotheks-Funktion aufruft, die einen Float- oder Double-Parameter erwartet. Es wird auch gezeigt, dass man, bevor man Rückgabewerte von Bibliotheksfunktionen verarbeiten kann, den Rückgabetyp explizit setzen muss.

Wie man sieht bedarf es einiger Umstände um beliebige Bibliothken ansprechen zu können. Trotzdem ist es noch recht einfacher Python-Code, den man auch noch mal hinter einem einfachen, in Python programmierten Wrapper verstecken könnte. Probleme könnten auch dabei auftreten, dass ctypes direkt im Speicher arbeitet und man zwar nicht in C programmiert, sondern in Python, aber trotzdem sehr viel über C wissen muss, zum Beispiel benötigt man Kenntnisse über Memorymanagement, Alignment, Datentypen, Casting. Ein genaueres Eingehen würde hier den Rahmen sprengen, deshalb verweise ich auf [Python Docs] unter „The Python Standard Library: Generic Operating System Services“.

2.2 Module selbst schreiben

Falls man keine fertige Bibliothek ansprechen möchte, sondern zum Beispiel Funktionalität in ein externes Modul auslagern will, auf dessen Erstellung man Einfluss nehmen kann, kann man die C-API benutzen ([Python Docs] unter „C API Reference Manual“). Wie der Name schon sagt, müssen Erweiterungen unter Benutzung dieser API in C geschrieben werden.

Listing 4. Benutzung des Moduls Spam

```
1 import spam
2 status = spam.system("ls -l")
```

Im Gegensatz zu ctypes lässt sich der benutzende Python Code genau so schreiben, wie der für eingebaute Module: Listing 4 zeigt, wie unser Beispiel am Ende zu benutzen sein soll. Es soll eine Schnittstelle zu C erstellt werden, die (nur) die Bibliotheksfunktion `int system(char*)` aufruft, ähnlich der bereits im Modul `os` vorhandenen Funktionalität.

Listing 5. spammodule.h

```
1  #include <Python.h>
2
3  static PyObject *spam_system(PyObject *self, PyObject *↩
       args){
4
5      const char *command;
6      int         sts;
7
8      if (!PyArg_ParseTuple(args, "s", &command))
9          return NULL;
10
11     sts = system(command);
12     return Py_BuildValue("i", sts);
13 }
```

Dazu benötigt man zuerst eine Quellcode-Datei, nennen wir sie `spammodule.c`. `Python.h` erlaubt Zugriff auf alle benötigten Funktionen und Makros für den Zugriff auf die Python Laufzeitumgebung, wobei alles mit `Py` beziehungsweise `PY` anfängt. Listing 5 zeigt die komplette Datei. Im Prinzip müssen alle Funktionen, die aus Python aufrufbar sein sollen, einen Zeiger auf ein `PyObject` zurückgeben. Dieses Objekt wird dann in Python als Rückgabewert verwendet. Jedes Datum was aus Python kommt oder zu Python geht muss ein `PyObject` sein, welches die Basisklasse[2] für alle Daten von/zu Python darstellt. Der Parameter `self` ist ein Zeiger auf das aufrufende Python-Objekt bzw. Python-Modul für Modul-Funktionen, `args` ist ein Tupel aus Argumenten, von denen jedes seinerseits wieder ein Zeiger auf ein `PyObject` ist. `PyArg_ParseTuple` extrahiert aus `args` mittels eines Typ-Strings die Argumente und macht sie über bereitgestellte Zeiger verfügbar. Im Fehlerfall gibt es `false` zurück und setzt noch globale Exception-Variablen (wie die meisten Funktionen aus `Python.h`). Wenn man Python einen Fehler bei der Ausführung signalisieren will, muss man `NULL` aus seiner C-Funktion zurückgeben. `Py_BuildValue` nutzt wieder einen Typ-String, um C-Daten in Python-Daten zu verwandeln.

[2] Klasse ist nicht das richtige Wort, besser eine Struktur, welche in C eine Klasse imitieren soll

Listing 6. Method-Table mit einer Methode names system

```
static PyMethodDef SpamMethods[] = {
    //...
    {"system",  spam_system, METH_VARARGS,
     "Execute a shell command."},
    //...
    {NULL, NULL, 0, NULL}
};
```

Ist diese Funktionalität programmiert, muss man Python noch bekanntmachen, dass die neue Funktion überhaupt existiert und wie diese aufzurufen ist. Dazu erstellt man einen Method-Table wie in Listing 6, welche den Methoden-Namen, einen Funktions-Zeiger und neben einer Anweisung, wie Argumente übergeben werden, einen Doc-String enthält.

Listing 7. Init-Methode

```
PyMODINIT_FUNC initspam(void) {
    (void) Py_InitModule("spam", SpamMethods);
}
```

Weiterhin muss man dann diesen Method-Table an Python übergeben. Das macht man, wie in Listing 7 zu sehen, mit einer Funktion, die den Namen des zu importierenden Moduls trägt und mit dem `PyMODINIT_FUNC` Makro definiert wird. Diese Funktion wird aufgerufen, wenn das entsprechende Modul in Python per `import spam` importiert wird. Im Funktionskörper wird dann das Modul initialisiert indem es von der Funktion `Py_InitModule` in `sys.modules` und die Methodenliste in das neue Modul `spam` eingefügt wird.

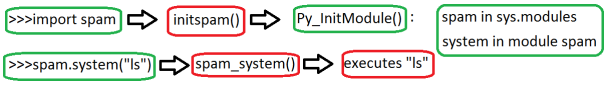

Abbildung 2. Extension Callgraph

In Abbildung 2 wird der Ablauf eines Aufrufs grafisch dargestellt.

Dies war nur ein Ausschnitt aus dem Programmieren von Erweiterungen mittels der C-API, auf verschiedene Möglichkeiten und Probleme sei dennoch hingewiesen: Schlüsselwort-Parameter zu übergeben ist möglich, es gibt keine void-Funktionen, man muss stattdessen `Py_None` zurückgeben. Man muss das komplette Reference-Counting in C mittels `Py_INCREF(x)` und `Py_DECREF(x)` selbst realisieren. C++ Programmierung ist möglich, wenn man ein paar Sachen beachtet.

2.3 SWIG

Wir haben grade zwei Möglichkeiten gesehen, Python zu erweitern, doch beide sind sehr komplex und benötigen viele Detailkenntnisse. *SWIG* [SWIG] ist ein Generator für Interfaces zwischen C/C++ und diversen anderen Sprachen wie Python, Perl, PHP, Tcl, Ruby und weiteren. Er verpackt in unserem Falle normalen C Code in C-API Code und ein Python Modul, der generierte Code führt Konvertierungen transparent durch. Man benötigt allerdings eine Interface-Datei die beschreibt, welche Funktionen in welchem Modul verfügbar gemacht werden sollen. Ein solcher Workflow ist in Abbildung 3 gezeigt, Beispiele sind in Listings 8 und 9 aufgeführt.

Listing 8. example.c

```
#include "example.h"

int cube( int n ) {
  return n * n * n;
}
```

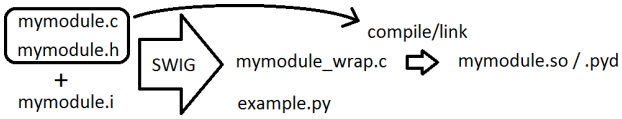

Abbildung 3. SWIG Workflow

Listing 9. example.i

```
%module example
%{
#define SWIG_FILE_WITH_INIT
#include "example.h"
%}

int cube( int n );
```

2.4 Beispiele

Für viele verschiedene existierende Software-Projekte und Bibliotheken gibt es inzwischen Python-Bindings, sodass man ctypes nicht benutzen und auch nicht selbst C-Wrapper schreiben muss. Hier stelle ich zwei Beispiele kurz vor: *PyQt* und *mod_python*.

Listing 10. PyQt Sample Window by [PyQt Tut]

```
import sys
from PyQt4.QtGui import QLabel, QApplication

if __name__=='__main__':

    App = QApplication(sys.argv)

    Label = QLabel( "Hello World!" )
    Label.show()

    App.exec_()
```

Abbildung 4. Qt Sample Window Shot

PyQt [PyQt] ist ein Binding für das bekannte Qt-Framework [Qt]. Python lässt sich damit nicht nur um eine ausgewachsene GUI-Bibliothek erweitern, sonder Qt beinhaltet auch Unterstützung für Netzwerk, Unicode, Xml, Datenbanken, etc. PyQt kann sogar den QtDesigner nutzen, welcher normalerweise auf das Generieren von C++ Code beschränkt ist. Listing 10 zeigt, wie einfach man ein Hello-World-Fenster wie in Abbildung 4 erstellen kann.

Standard CGI:	23 requests/s
Mod_python publisher:	476 requests/s
Mod_python handler:	1203 requests/s

Tabelle 1. Vergleich von CGI und Python mit eingeschaltetem Publisher-Handler und nur mit Standard-Handler ([mod_python])

Mod_python [mod_python] ist ein Modul für den Apache Webserver [Apache]. Apache führt das Request-Handling in verschiedenen Phasen durch, wie „Request lesen“, „Header parsen“, „Zugriffsrechte prüfen“, usw. Mit mod_python kann man nun jede dieser Phasen durch eigenen Code ersetzen. Dabei könnte man darüber streiten, ob es sich um eine Möglichkeit der Erweiterung von Python handelt (es wird möglich, Webseiten zu generieren),

oder ob Python in den Apache Webserver eingebettet wird (Apache ruft Python auf). Mod_python hat verschiedene Vorteile gegenüber traditionellen CGI-Handlern, allen voran die Geschwindigkeit, siehe dazu Tabelle 1. Dies wird unter anderem dadurch erreicht, dass der Python Interpreter ständig im Hintergrund läuft und nicht jedes mal neu gestartet wird, wie bei einem CGI-Handler. Weiterhin werden Bibliotheken nur einmal geladen und es besteht die Möglichkeit, Datenbanken-Verbindungen über mehrere Aufrufe hinweg zu erhalten, was wiederum Geschwindigkeit bringt.

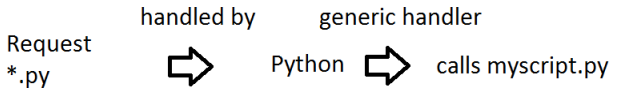

Abbildung 5. Kontrollfluss zwischen Apache und mod_python

Der typische Kontrollfluss zwischen Apache und mod_python ist in Abbildung 5 dargestellt.

Listing 11. mod_python Xml Config für eigenen Handler by [mod_python Tut]

```
1  <Directory /mywebdir>
2      AddHandler mod_python .py
3      PythonHandler myscript
4      PythonDebug On
5  </Directory>
```

Listing 12. mod_python Code für eigenen Handler by [mod_python Tut]

```
1  from mod_python import apache
2
3  def handler(req):
4
5      req.content_type = "text/plain"
6      req.write("Hello World!")
7
8      return apache.OK
```

In Listings 11 und 12 ist ein Beispiel dargestellt, wie man einen Request empfängt und eine Hello-World-Antwort zurückgibt. In der Konfigurationsdatei werden zuerst alle Aufrufe von Dateien mit der Endung „py“ im Verzeichnis „/mywebdir“ von Python gehandhabt. Dann wird Python gesagt, dass das Modul „myscript“ für das bearbeiten der Anfragen zuständig ist. Im entsprechenden Modul gibt es eine Funktion `handler`, welche für jede Anfrage aufgerufen wird. Dabei ist das `req` Objekt gleichzeitig zum Lesen der Anfrage und zum Schreiben der Rückgabe gut.

3. Embedding Python

Wenn bereits eine große Applikation vorhanden ist, welche nicht in Python geschrieben ist, und man diese um die Flexibilität und Erweiterbarkeit von Python ergänzen will, so kann man Python in andere Programme einbetten. Man muss damit bestehende Anwendungen nicht in Python übersetzen, um Pythons Features zu nutzen und kann auch dem Endbenutzer die Möglichkeit geben, die Anwendung nach seinen Vorstellungen in Python zu verändern.

Die Möglichkeit Python (wie viele andere Sprachen) per Kommandozeile aufzurufen ist die einfachste Variante, aber zugleich beschränkt, weil Daten nur umständlich von Python und der Applikation gemeinsam benutzt werden können. Deshalb möchte ich hier die Möglichkeit vorstellen, Python direkt in andere Applikationen einzubetten.

3.1 How To

Die Anwendung muss gegen die richtige Version der Python DLL beziehungsweise SO linken und natürlich die Python Header Dateien einbinden. Einen Beispielaufruf kann man in Listing 13 sehen. Dabei wird der Python Interpreter gestartet und initialisiert. Er läuft solange, bis `Py_Finalize()` aufgerufen wird. Dazwischen wird in Python ein einfacher Aufruf ausgeführt, wie man ihn auch in der Python Kommandozeile ausführen könnte.

Listing 13. Python Aufruf

```
1  #include "python.h"
2  //...
3  Py_Initialize();  //Initialize Python
4  PyRun_SimpleString("from time import time,ctime\n"
5                     "print 'Today is',ctime(time())\n"↩
                          );
6  Py_Finalize();
```

Man sieht bereits an der Header-Datei und den Funktionsnamen, dass die selbe API zum Einsatz kommt, wie beim Erweitern von Python. Man kann Daten an Python entweder durch den Aufruf-String übergeben, oder zum Beispiel eine Funktion auf einem Python Objekt per `PyObject_CallObject()` aufrufen, und die gewünschten Parameter übergeben.

Was allerdings, wenn man nicht jedes Datum übergeben, sondern von Python aus direkt auf die Daten der umgebenden Applikation zugreifen möchte? Man kann dann ähnliche Wege gehen, wie in Abschnitt 2 beschrieben und sich auch wieder von SWIG helfen lassen: man erhielte damit eine Applikation mit eingebettetem Python, welches wiederum auf die Applikation (oder andere Bibliotheken) zugreifen kann.

3.2 Beispiele

Als Beispiel für den erfolgreichen Einsatz von Python als eingebettete Programmiersprache möchte ich hier das rundenbasierte Strategiespiel Civilization IV anführen. Durch den Einsatz von Python sind Features realisiert worden, für die sonst eine eigene Skriptsprache hätte entwickelt werden müssen: Änderungen am gesamten Interface sind möglich, man kann den Kartengenerator beeinflussen und sogar komplexe Events steuern, die den Spielablauf verändern können. Das alles passiert voll in das umgebende Spiel eingebettet, ohne dass der Benutzer etwas von der Einbettung merkt.

4. Implementieren/Kompilieren/Übersetzen

Neben den in den Abschnitten 2 und 3 genannten Möglichkeiten möchte ich hier einen kurzen Überblick über einige andere Möglichkeiten geben, Python zu verändern oder zu erweitern und auf zwei dieser Möglichkeiten, Jython und Stackless, näher eingehen.

Freeze verpackt Python-Code und den Python-Interpreter in eine ausführbare Datei. Es kann dazu verwendet werden, Python Anwendungen leichter verteilbar zu machen. Freeze ist kein Compiler und erhöht demzufolge auch nicht die Geschwindigkeit!

Shedskin portiert eine Untermenge von Python automatisch zu C++ Code. Damit kann man (eingeschränkt durch die Untermenge) Python programmieren, aber alle Vorteile einer kompilierten Sprache genießen.

Psyco ist ein Optimierungsprogramm für Python-Code. Es funktioniert ähnlich wie ein JIT: es schreibt, basierend auf den Laufzeitdaten der Anwendung, für alle möglichen Typen der Daten spezialisierten Code. Psycos Ideen werden im Interpreter PyPy genutzt. Dies macht Python Code 2x-100x schneller, wobei der Speedup im Durchschnitt eher bei 4x liegen dürfte.

Pyrex ist Python um C-Datentypen erweitert. Dadurch gewinnt man ein wenig Kontrolle über die Datentypen und durch die anschließende Kompilierung erhält man Geschwindigkeitsvorteile.

4.1 Jython

Ein Python-Interpreter neben der Referenz-Implementierung CPython ist Jython. Jython ist eine Kombination aus Java und Python, es kompiliert Python zu Java-Bytecode und lässt diesen dann auf einer Java-VM laufen. Daneben kann man von Python aus direkt auf Java-Klassen zugreifen und sogar von diesen erben. Erweiterungen von Python in C werden allerdings noch nicht unterstützt. Der Python Standard wird zu 100% implementiert, jedoch gibt es Abweichungen beim Laufzeitverhalten, zum Beispiel wird kein Reference-Counting genutzt, sondern die normale Java Garbage Collection. Ein großes Projekt, was Jython als Skriptsprache einsetzt, ist IBM Websphere.

Listing 14. Verwendung von Java Klassen in Jython

```
1  from java.util import Random
2  r = Random()
3  r.nextInt()
4  # ergibt z.B.
5  #501203849
6
7  for i in xrange(5):
8      print r.nextDouble()
9  #ergibt z.B.
10 #0.435789109087
11 #0.0702903104743
12 #0.962867215318
13 #0.674547069552
14 #0.434106849824
```

Im Listing 14 ist ein einfaches Beispiel dargestellt, wie man Java Klassen verwenden kann.

Listing 15. Ein Java-AWT Fenster mit Jython by [Jython Tut]

```
1  from java import awt
2
3  class SpamListener(awt.event.ActionListener):
4      def actionPerformed(self,event):
5          if event.getActionCommand() == "Spam":
6              print 'Spam and eggs!'
7
8  f = awt.Frame("Subclassing Example")
9  b = awt.Button("Spam")
10
11 b.addActionListener(SpamListener())
12 f.add(b, "Center")
13 f.pack()
14 f.setVisible(1)
```

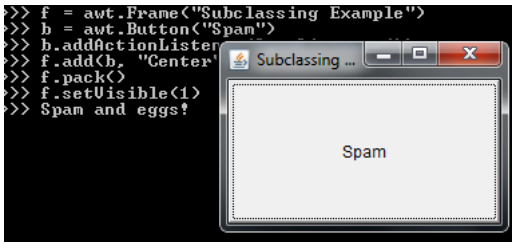

Abbildung 6. Screenshot AWT Fenster mit Jython

Um die Einfachheit der Zusammenarbeit mit Java zu verdeutlichen, ist in Listing 15 ein Quelltext dargestellt, der in Jython ein Java-AWT Fenster erstellt und in Jython von der Java Klasse `awt.event.ActionListener` ableitet, um bei Button-Klick eine Aktion auszuführen. Das erstellte Fenster ist in Abbildung 6 zu sehen.

4.2 Stackless

Stackless ist nicht nur ein Python Interpreter, sondern eine eigene Python-Distribution, Stackless ergänzt Pythons Sprachschatz um einige Features, die paralleles Programmieren erleichtern. Darüber hinaus führt Stackless sogenannte Microthreads ein, das sind vom Interpreter (nicht vom Betriebssystem) gemanagte Threads, die sich dadurch viel schneller wechseln lassen und weniger Speicherplatz verbrauchen. PyPy implementiert wiederum viele Features von Stackless.

Ein prominentes Beispiel für den Einsatz von Stackless ist das Online Rollenspiel *Eve Online*. Der Server dieses Spiels muss viele Tausend Nutzeraktionen gleichzeitig verarbeiten, da meist 25 000 Nutzer oder mehr online sind, in Hochzeiten sind es über 50 000. Dabei hilft Stackless durch seine Unterstützung für Parallelisierung: viele kleine Aufgaben sind auf der Basis von Microthreads realisiert. Darüber hinaus wird Python wegen seiner hohen Produktivität von den Entwicklern geschätzt.

5. Vergleich mit LUA

Zum Vergleich werde ich hier kurz einige Details einer anderen häufig genutzten Sprache hervorheben: *LUA*. Diese wird in diversen Computerspielen eingesetzt, wie zum Beispiel *World of Warcraft*, *Supreme Commander* oder *Civilization V*. Wie Python wird sie dabei zur Realisierung der einfachen Anpassbarkeit der Applikationen benutzt. Die Syntax von Lua ist einfacher als die von Python. Die Standardbibliothek ist ebenfalls kleiner, genau wie der Speicherbedarf, wodurch sich die Sprache gut zum Einbetten in andere Programme eignet. Da in Lua nicht jedes Ding ein Objekt ist, ist sie in manchen Punkten schneller als Python. Lua hat ebenfalls eine C-API, auch diese ist simpler als die von Python. Man kann natürlich in der kleinen Standardbibliothek und der einfacheren Syntax sowohl Nachteile als auch Vorteile sehen.

Literatur

[Brooks] Frederick P. Brooks: The Mythical Man-Month

[Python Docs] *Python Docs* http://docs.python.org/

[PyQt] *PyQt* http://www.riverbankcomputing.co.uk/software/pyqt/intro

[SWIG] *SWIG* http://www.swig.org/

[Qt] *Qt* http://qt.nokia.com/

[PyQt Tut] *PyQt - Intro* http://www.harshj.com/2009/04/26/the-pyqt-intro/

[mod_python] *mod_python* http://www.modpython.org/

[Apache] *Apache Webserver* http://httpd.apache.org/

[mod_python Tut] *mod_python Tutorial* http://www.modpython.org/live/current/doc-html/tut-what-it-do.html

[Galileo] *Galileo OpenBook - Python* http://openbook.galileocomputing.de/python/

[ShedSkin] *Shed Skin* http://shed-skin.blogspot.com/

[Psyco] *Psyco* http://psyco.sourceforge.net/

[Stackless] *Stackless* http://www.stackless.com/

[Jython] *Jython* http://www.jython.org/

[Jython Tut] *Jython Tutorial* http://wiki.python.org/jython/UserGuide#a-short-example

[EVE] *EVE Online* http://www.eveonline.com/