Konfigurative Code und Dokumentationsgenerierung mit dem Pythontool Cog

Max Noppel

Dez 2017

Was für ein bekloppter Titel ist das eigentlich?! (Konfiguration)

- Normalesweises dynamisch
- Umfangreich
- Verschachtelt
- Betriebsparameter
- Zusammenstellen des Systems[1]

[1] https://de.wikipedia.org/wiki/Konfiguration_(Computer)

Was für ein bekloppter Titel ist das eigentlich?! (Codegenerierung)

Als Kodegenerierung oder Codegenerierung wird die automatische Erzeugung von Quelltext in einer bestimmten Programmiersprache bezeichnet.[2]

[2] https://de.wikipedia.org/wiki/Codegenerierung

Was für ein bekloppter Titel ist das eigentlich?! (Dokumentationsgenerierung)

- Wie geil wäre das den?
- Exaktheit

Was für ein bekloppter Titel ist das eigentlich?! (Python)

Figure 1:
http://aisrtlnext-a.akamaihd.net/masters/270327/handlicher-python.jpg

Um was geht es

- C++, C
- Dokumentation
- Buildprozesse
- Python
- Preprozessoren
- TeX
- Validierung von Eingaben in den Compileprozess

Was ist überhaupt das Problem? (Embedded)

- Embedded ohne Dateisystem[3]
- Boilerplate Code über viele Dateien
- Viele Konfigurationsparameter[4]

[3] Hier ist nicht Speicher sondern ein Dateisystem gemeint. Ext4, Ntfs, Fat

[4] Zum Beispiel viele Pins, Datenbankeinträge, Interpreterkeywords, ...

Was ist überhaupt das Problem? (!embedded)

- Informationen liegen an vielen Stellen vor (redundant)
- Wartbarkeits und Übersichtsproblem
- Visualisierungen fehlen
- Dokumentation weicht vom Code ab und muss nachgezogen werden

Was ist überhaupt das Problem? (sonstiges)

- Der C/C++ Preprozessor ist nicht mächtig genug
- Es kann zur Compilezeit keinen Validierung der Eingaben gemacht werden

Was ist überhaupt das Problem?

DEMO

Was können wir dagegen tun?

Wir können:

- einen mächtigen Preprozessor einsetzen
- Konfiguration zur Compilezeit
- die Informationen nur einmal aufbewahren

Reicht das schon?

Reicht das schon?

Scheinbar!

What about python?

- mächtig
- einfach zu lernen und zu verstehen

cogapp[5]

- Download https://pypi.python.org/pypi/cogapp
- Extract
- `sudo python setup.py install`
- testen mit `python -m cogapp`

[5] https://nedbatchelder.com/code/cog/

Was kann das? (1)

- Es führt Pythoncode in Kommentarblöcken aus und erweitert die Dateien.
- Es dient quasi als Preprozessor.
- Mit der vollen Funktionalität von Python.

Wie sieht das aus? (1)

src/audiocontroller.cpp

```
...
//[[[cog
//import cog
//from py import record
//
//for r in record.getRecords():
//    cog.outl("extern Record rec%s;" % (r.name))
//]]]
extern Record recLevel;
extern Re...
//[[[end]]]
...
```

Wie sieht das aus? (2)

src/audiocontroller.cpp

```
...
//[[[cog
//import cog
//from py import record
//
//num = 0
//for r in record.getRecords():
//  num += 1
//
//cog.outl("#define NUM_RECORDS %d" % (num))
//]]]
#define NUM_RECORDS <num>
//[[[end]]]
...
```

Was kann das? (2)

Eingabe kann alles sein, was Python lesen kann

Was kann das? (3)

-> Alles!

- JSON, XML
- HTML Onlineservices
- ...

Was kann das? (4)

Man kann damit alles umsetzen, was Python kann

Was kann das? (5)

-> Alles!

- Summen
- Produkte
- Listen
- Bedingungen
- Schleifen
- Eingabenvalidierung

Der Buildprozess generell

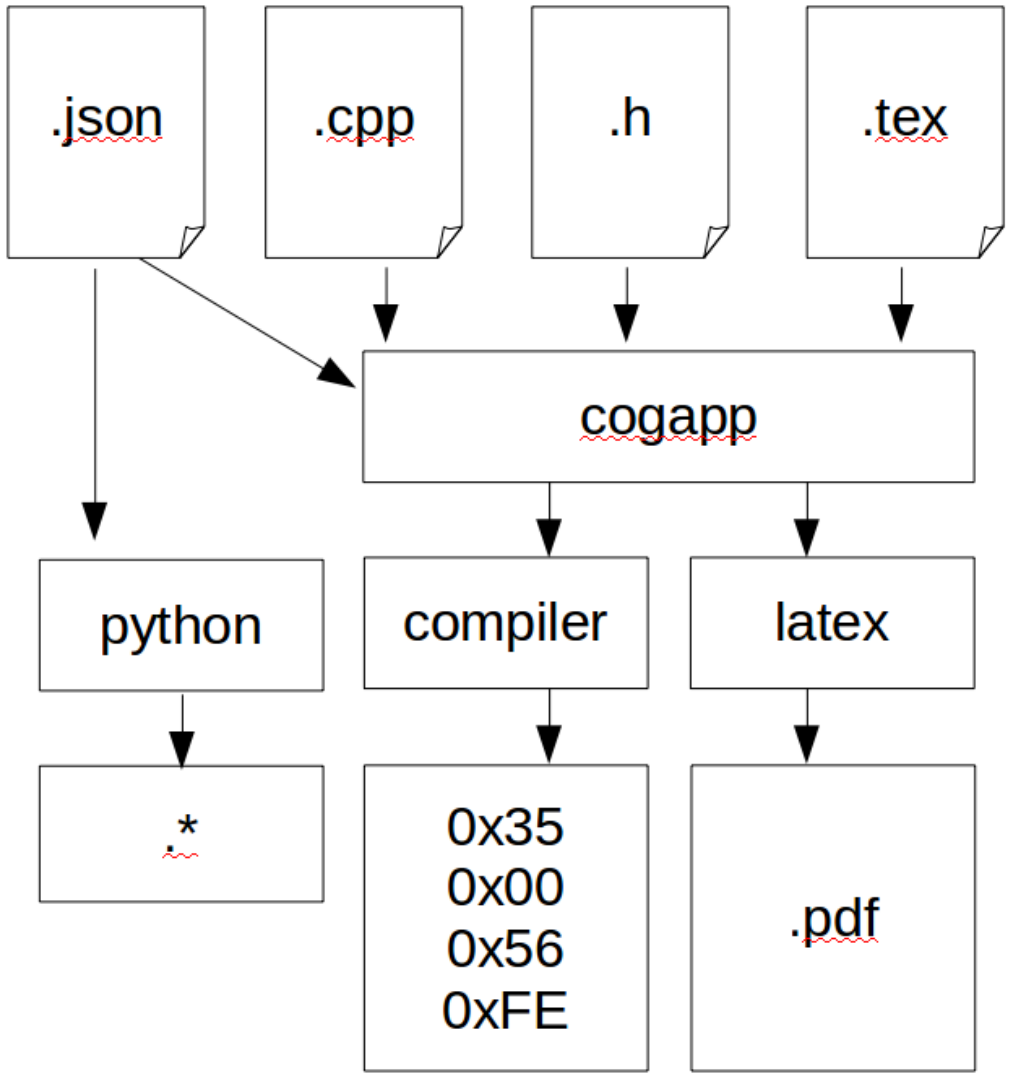

Figure 2: Buildprozess

Sprechen wir über den Buildprozess

- Pythonmodule zum Parsen der Eingabedateien
- Pythonmodule zum Validierungen der Eingabedateien
- Kopieren, dann coggen oder einfach nur coggen
- GNU Make
- LaTeX

Eingabe parsen (1)

py/record.py

```
import os
import json

with open("config"+os.sep+"records.json") as recordsFile:
    recordsJSON = json.load(recordsFile)
    recordsFile.close()

def getRecords():
  recordList = []
  for record in recordsJSON:
    recordList.append(Record(record))
  return recordList
```

Eingabe parsen (2)

py/record.py

```
class Record:
  def __init__(self, jsonObj):
    self.name = jsonObj['name']
    self.minValue = jsonObj['minValue']
    self.maxValue = jsonObj['maxValue']
    self.description = jsonObj['description']

    if self.minValue > self.maxValue:
      raise Exception("MinValue muss kleiner sein als der M
```

Eingabe validieren (1)

```
def verifyUnique():
    nameList = []
    for r in record.getRecords():
        nameList.append(r.name)
        if( len(nameList) != len(set(nameList))):
            raise Exception("Duplicated name. Name must be
    print "Verify unique recordsname: SUCCESS"
```

Erst kopieren?

- Vorteile
  - SCM
  - Übersichtlicher
- Nachteile
  - Code nicht direkt erkennbar
  - Preprocessorergebnis nicht ersichtlich
  - Probleme mit IDEs, Autovervollständigung usw.

GNU Make[6] (1)

- cog als Preprozessorschritt in Buildprozess einfügen

Makefile

```
INFO = config/records.json
INFO += py/record.py
INFO += py/verify_records.py

$(COGDIR)/$(SRCDIR)/%.cpp: $(SRCDIR)/%.cpp $(INFO)
    @ cp $< $@
    @ $(PYTHON) -m cogapp -r $@
```

[6] https://www.gnu.org/software/make/

GNU Make (2)

Makefile

```
verify:
    $(PYTHON) py/verify_records.py
```

LaTeX

doc/audiocontroller.tex

```
%[[[cog
%import cog
%from py import record
%
%for r in record.getRecords():
%    cog.outl("\\subsection{%s}" % (r.name))
%    cog.outl("Minimaler Wert: %d\\\\" % (r.minValue))
%    cog.outl("Maximaler Wert: %d\\\\" % (r.maxValue))
%    cog.outl("Beschreibung %s\\\\" % (r.description))
%]]]
\subsection{Level}
Minimaler Wert: 0
Maximaler Wert: 11
Beschreibung Hier kann man die Lautstaerker einstellen.
%[[[end]]]
```

Wie sieht das aus? (3)

DEMO

Welche Möglichkeiten eröffnen sich uns zusätzlich?

Welche Möglichkeiten eröffnen sich uns zusätzlich?

- Validierung von Eingaben zur Compilezeit
- Visualisierungen
- Eigene Grammatiken, Sprachen die als ASTs abgelegt werden, DSLs[7]
- GUIs
- Interdisziplinäre Zusammenarbeit (Hardware, Software, Design,. . .)

[7] Domain Specific Languages

Fazit

Vorteile

- erzwungen exakt, über Commits
- pflegeleicht
- refactoring
- umfangreich

Nachteile

- Hohe Initialkosten
- Komplexer
- Mehr Sprachen
- Mehr Code

Ende

```
return 0;
```