FAKULTÄT FÜR !INFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

computer languages

A Modular Approach to Configuration Storage

Markus Raab

Technische Universität Wien
Institut für Computersprachen

Arbeitsbereich: Programmiersprachen und Übersetzer
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr. Franz Puntigam

Masterstudium:
Software Engineering &
Internet Computing

Kontext

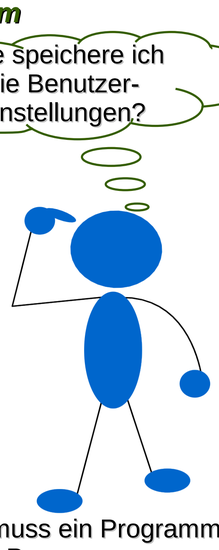
Lesen und Schreiben von **Konfiguration**

Hat viele Anforderungen

- Protokollieren
- Benachrichtigen
- Konfiguration prüfen
- Escape und Encode
- Konflikte erkennen
- Werte validieren
- Unnötige Updates vermeiden
- Verschiedene Konfigurationsformate anbieten
- Verhindern das ungültige Konfiguration gespeichert wird
- Verschiedene Betriebsysteme unterstützen

Problem

Wie speichere ich die Benutzer-einstellungen?



Derzeit muss ein Programmierer für jedes Programm den Umgang mit **Konfiguration selbst** implementieren.

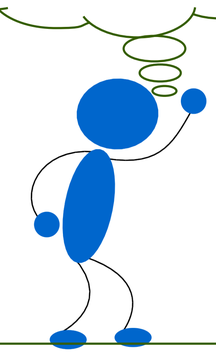
Bibliotheken helfen nur für Teilaufgaben (z.b. Parser) oder sind nicht portabel (z.b. WinReg).

Vollständige und portable Lösung?

Problem mit Ansatz

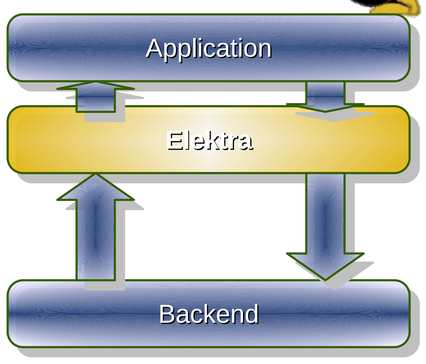
Mit Elektra 0.7 war es nicht möglich alle **Anforderungen** (siehe *Kontext*) zu berücksichtigen.

- Duplizierter Code
- Fehlende Flexibilität
- Features fehlen



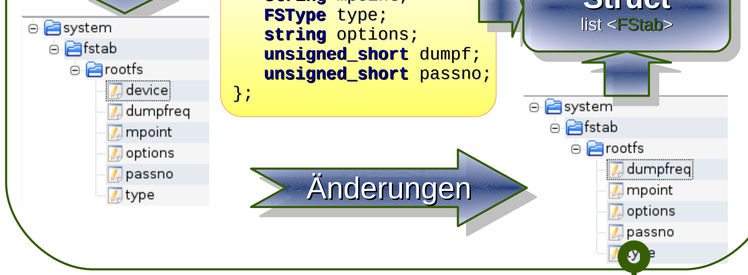
Modularität in Backends?

Existierender Ansatz



Struktur überprüfen

```
LABEL=stable /      jfs defaults,errors=remount-ro 0 1
proc         /proc  proc  defaults 0 0
LABEL=swap   none   swap  sw 0 0
LABEL=home   /home  jfs rw,suid,dev,exec,auto,nouser 0 2
```




```
struct FStab {
   string device;
   string mpoint;
   FSType type;
   string options;
   unsigned_short dumpf;
   unsigned_short passno;
};
```

Struct
list <FStab>

Änderungen

Ergebnis

Überprüfungen in zwei Phasen flexibler

1. **Phase** Anreichern von dynamischen Typinformationen
2. **Phase** Überprüfen der Typen mit *checker plugins*

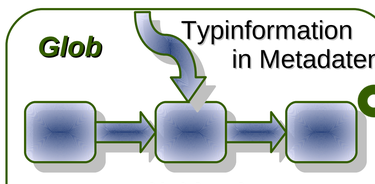
Überprüfung der Typen

```
enum FSType
{fat,ntfs,ext2,jfs,proc,swap};
string
unsigned_short
```

Basic CORBA Typsystem

Glob

Typinformation in Metadaten



```
/*/device
```

Weitere Überprüfungen

- ✔ Validation

```
[0-9]+x[0-9]+
```

- ✔ Network
- ✔ Filename

Hosts

```
#/etc/hosts file

# a new comment


192.168.0.23 first_entry
192.168.0.24 second_entry
alias2 alias3 alias4
alias5
192.22.22.3  third_entry


# Markus follows


192.168.1.55 markus
```

Simpleini

```
user/simpleini/example1 = @NULL
user/simpleini/example_empty_string =
@EMPTY
user/simpleini/example_null_value =
@NULL
user/simpleini/example_text = @@text
user/simpleini/equal = %3Dequal%3D
```

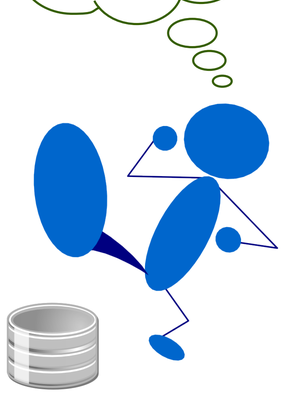
✔ Kommentare erhalten

✔ Reihenfolge bleibt gleich

Methode

Idee: Schnelle Entwicklung von Storage Plugins durch Verwendung von **Grammatiken**

Wow! Damit entwickle ich Storage Plugins im Nu!



```
{
   {system/hosts=""
        {file=/etc/hosts}
        {mode=644}
        {ctime=1273165467}
   }
}
```

```
query =  '{' >> pair >>
    *(pair) > '}';
pair  =  '{' >> key >
    '=' >> val >>
    *('{' >> metakey >
    '=' >>
metaval > '}') > '}';
```

Dump

```
kdbOpen 1
ksNew 3
   keyNew 10 8
   user/dumprootkey
   keyEnd

   keyNew 15 7
   user/dump/key1value1
   keyEnd

   keyNew 15 7
   user/dump/key2value2
   keyEnd

ksEnd
```

XML

```
<?xml version="1.0"
encoding="UTF-8"?>

<keyset xmlns=""
        xsi:schemaLocation=""
        parent="user/env/alias">

   <key basename="ls"
        type="string"
        value="ls -Fh"

        <comment></comment>
   </key>

   <key basename="vnc"
        type="string">
        <value>vncserver
       -geometry
        900x650</value>
        <comment></comment>
   </key>
</keyset>
```

PLUGINS

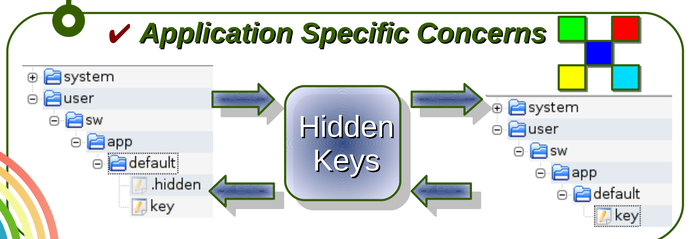
Resultate

- Plugins lösen An-forderungen von *Kontext*
- Separation of **Concerns**
- Wiederverwendbarkeit
- Backends zur Laufzeit **zusammenbauen**
- Nur funktionierende Backends durch **Kontrakte**
- Unix Philosophie

✔ *Character Reduction*

`"'val  x'"`	`value=abc xyz`
↓	↓
`\'val\tx\'`	`value%3Dabc%20xyz`

✔ *Application Specific Concerns*



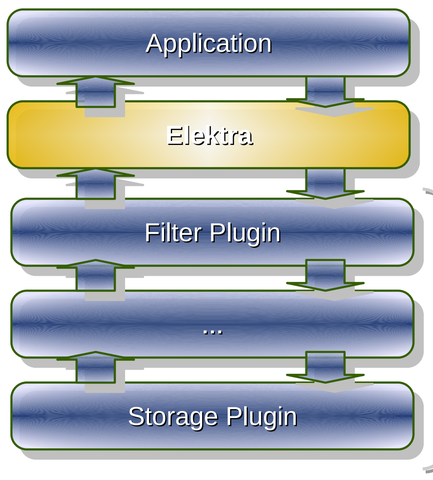
Methode

Zusicherung mit **Kontrakte**:

- **Semantik** des Backends
- **Reihenfolge** der Plugins

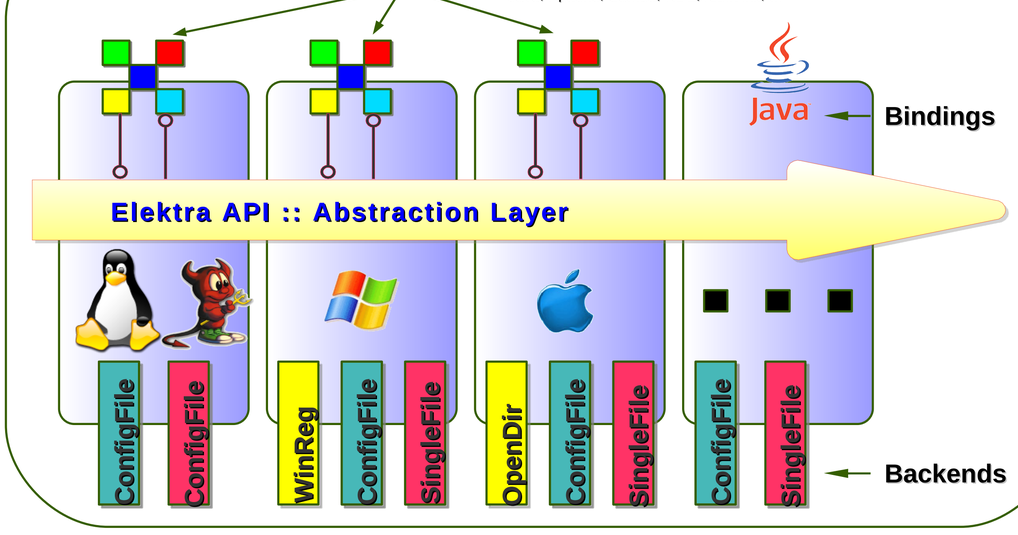
Zusammenspiel garantiert durch Kontrakte...

Modularer Ansatz



Ziel

Portable und nicht portable Software
Java, Apache, Samba, KDE, /sbin/init, ...



Methode

Idee: Schnelle Entwicklung durch Verwendung existierender Bibliotheken die für Teilaufgaben helfen:

Rad nicht neu erfinden!

① **Nickel** (INI Parser)

```
multi="line
inside quotes" ; with
comments after
escapes=\\\a\r\n\x11\b\05
6\t\\t\xj
\==\; not a comment
\[not a section\]=yeah
```

```
Ni_node current = NULL;
while ((current =
     Ni_GetNextChild(root, current)) != NULL)
{
     Key *k = keyNew(0);
     keySetName (k,
          Ni_GetName(current, NULL));
     keySetString (k,
          Ni_GetValue (current, NULL));
     ksAppendKey (returned, k);
}
```

② **D-Bus** für Notification

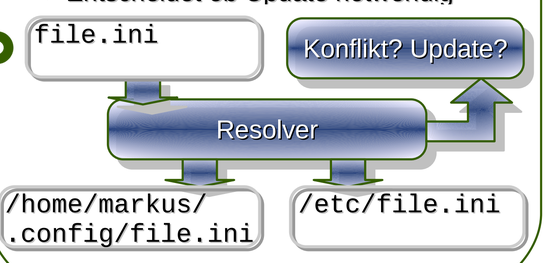
③ **Syslog** für Logging

④ **Libxml**

Resolver Plugin

Erledigt **nicht portable** Angelegenheiten

- Löst Dateinamen auf
- Findet Konflikte
- Entscheidet ob Update notwendig



Kontakt

http://www.libelektra.org

Markus Raab <elektra@markus-raab.org>