Freie Universität Berlin

↪ tarent

Bachelorarbeit:
Entwicklung eines Konfigurationsvalidierers (extern)

Christian Cikryt
Freie Universität Berlin

02.02.2012

Überblick

Motivation

- Zusammenarbeit mit *tarent solutions GmbH*
- Zielumgebung: dortige Portalsysteme
  - Viele verschiedene Komponenten, auch von Drittanbietern
  - ⇒ Konfiguration der Gesamtsysteme komplex und fehleranfällig
- Unterscheidung zwischen Konfigurations- und Programmierfehlern schwierig
  - ⇒ Behebung zeitaufwendig und kostenträchtig

Ziele

- Erkennen von Konfigurationsfehlern vor dem Einspielen
- Überprüfung der Konfigurationen auf Gültigkeit
  - ⇒ Wann ist eine Konfiguration gültig?
  - ⇒ Wissen vorhanden, aber verstreut und nicht automatisch auswertbar
  - ⇒ Beschreibung von Validitätskriterien nötig
- Jede Komponente liefert eigene Beschreibung mit
- Im Folgenden: Entwicklung der Beschreibungssprache

Grenzen dieses Ansatzes

- Nur bekannte Anforderungen überprüfbar
- Realistisch nicht alle Anforderungen und Verknüpfungen abbildbar
- Konflikt zwischen Erlernbarkeit und Mächtigkeit

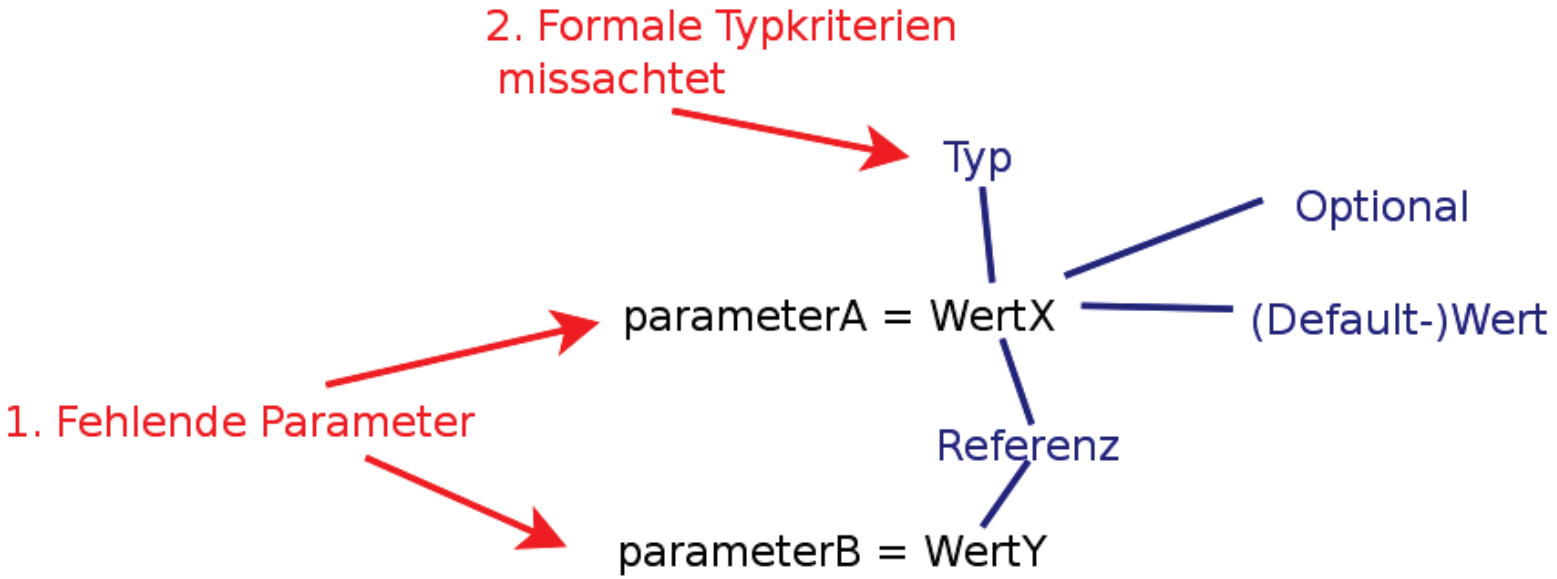
⇒ Orientierung an die häufigsten Fehlerklassen

Die häufigsten Fehlerklassen

1. Fehlende Parameter
2. Verstoß gegen formale Typkriterien
3. Nicht geänderte Platzhalterwerte
4. Inkonsistenzen: Abhängigkeiten zwischen Parametern missachtet

Anforderungen an die Beschreibungssprache

- Konfigurationen sind Parameter-Wert-Paare



- Abhängigkeiten zwischen Parametern: Optionale Funktionalität, Ausprägung

Ausprägungen und optionale Funktionalitäten

- Ausprägungen: Beispiel verschiedene Datenbanktypen

```
Postgres {
   db.url of type URL
}
Oracle {
   db.url of type URL
}
MySQL {
  db.hostname of type Hostname
  db.port defaultsTo '3306'
}
```

- Optionale Funktionalität: Beispiel Virenscanner

```
Virusscan {
  install.dir of type Dir,
  autoupdate of type Bool
}
```

Mögliche Syntax für die Beschreibungssprache

Freie Universität Berlin

```
component.id {
  parameters {
    in config.file {
      parameter.id[Bool] defaultsTo 'false' optional,
      parameter2.id[URL] = 'http://${Hostname}:${Port}',
      parameter3.id[URL] = parameter2.id # 'describing comment'
    }
  }
  flavors {
    db1 :
      activatedWhen parameter.id = 'true'
      parameters {
        db.url[URL] in config.file2
      }
    db2 :
      activatedWhen ...
  }

  functionality virusscan {
    activatedWhen ...
  }
}
```

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<component id="component.id">
  <parameters location="config.file">
    <parameter id="parameter.id" type="Bool" default="true" optional="true"/>
    <parameter id="parameter2.id" type="URL" value="http://${Hostname}:${Port}" />
  </parameters>
  <flavors>
    <flavor id="db1">
      <activatedWhen>
        <hasValue parameter="parameter.id" value="true" />
      </activatedWhen>
      <parameters>
        <parameter id="db.url" type="URL" location="config.file2" />
      </parameters>
    </flavor>
    <flavor id="db2">
      <activatedWhen>
        ...
      </activatedWhen>
    </flavor>
  </flavors>
  ...
</component>
```

- Gute Werkzeugunterstützung (Syntaxvervollständigung)
- Aber:
  - Geschwätzig, überflüssige Syntaxelemente
  - Starre, unspezifische Struktur
  - Ungeeignet für die Erstellung durch Menschen

Skizze der Beschreibung in YAML

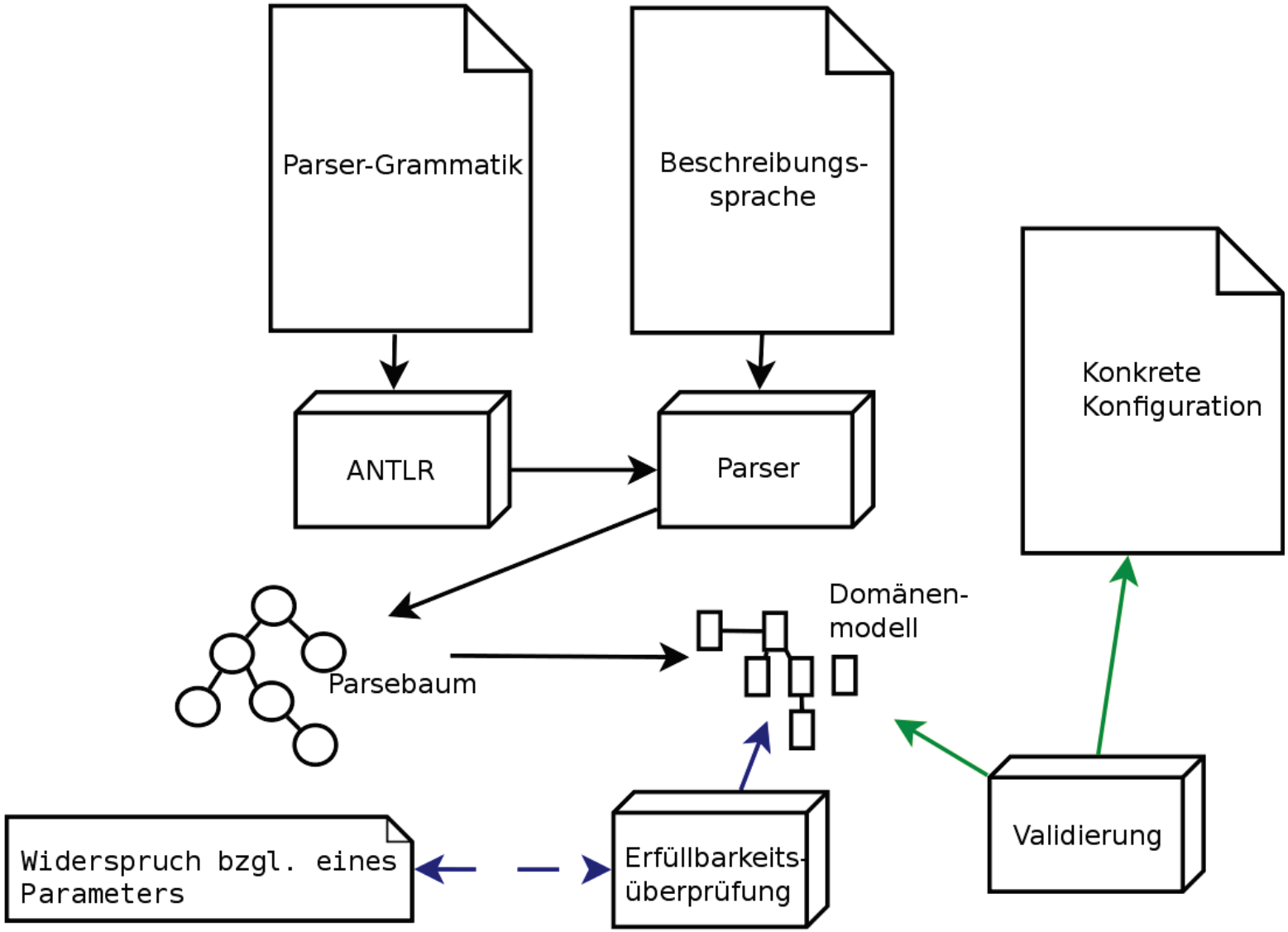

```
component: component.id
  parameters:
    location: config.file
      - id: parameter.id
        type: Bool
        default: true
        optional

      - id: parameter2.id
        type: URL
        value: "http://${Hostname}:${Port}"
  ...
```

- Menschenfreundliche Variante von XML
- Aufgeblähte Struktur
- Schablonenparameter erfordern weitere Schachtelungsebene oder zusätzlichen Parser

Entwicklung einer eigenen Sprache

- Ergebnis der Evaluation: Schaffen einer eigenen Sprache
- ANTLR als Parsergenerator
  - Ausgereift
  - Generierung von Java-Code
  - IDE mit Debugger
  - Kontextfreie Grammatik als Eingabe

Erfüllbarkeitsüberprüfung und Validierung



Das wurde erreicht

- Beschreibungssprache, um Konfigurationsanforderungen zu definieren
- Erweiterbar für künftige Anpassungen (Nutzerfeedback)
- Beschreibungen am Einsatzort und automatisch auswertbar
- Fordert aktuelle Konfigurationskriterien eindeutig zu formulieren
- Validierungswerkzeug: *Konfigurator*
- Evaluation der Lösung durch Anwendertests

Fragen?