Freie Universität Berlin

# ↪ tarent

Bachelorarbeit:
Entwicklung eines Konfigurationsvalidierers (extern)

Christian Cikryt
Freie Universität Berlin

21.07.2011

# Überblick

Motivation

Verwandte Arbeiten / Evaluation

Vorgehen

Technologie-Evaluation

Algorithmus

# Problembeschreibung

- Portalsysteme mit komplizierter Konfiguration.
- Ziel: Ausfall durch Fehlkonfigurationen vermeiden.
- Idee: Überprüfung der Konfiguration auf Gültigkeit.
- Beschreibung gültiger Konfiguration nötig.

# Anforderungen an die Lösung

- Beschreibungsmöglichkeiten:
  - Parameter mit Typüberprüfung.
  - Abhängigkeiten unter Parametern.
  - Clustersysteme.
  - Portalinstanzen.
- Einsatzgebiete: Test-, Entwicklungs- und Kundensysteme.
- JVM-Lösungen bevorzugt.

# Konfigurationsmanagementsysteme

- Beispiele: puppet, cfengine, Smartfrog.
- Zweck: Konfigurationen verteilen.
- Konfigurationsüberprüfung müsste davor ansetzen.
- Erweiterung möglich, aber:
  - Für Kundensysteme ungeeignet.
  - Einsteigshürde für Entwicklung und Nutzung.
  - Keine Festlegung firmenweit möglich.

# Verwandte Arbeit

„Automated Model-based Configuration of Enterprise Java Applications“:

- Fokus: Featureauswahl nicht Konfigurationsparameter.
- Zugreifen auf Constraint Satisfaction Algorithmus.
- keine wiederverwendbaren Softwareartefakte.

# Abgrenzung zu Constraint Satisfaction Problem



- Identisch: Wertebereich mit Constraints.
- Unsere Wertemenge ist prinzipiell unendlich.
- Unsere Fragestellung: Ist eine vorgegebene Lösung gültig?
- Zusätzlich: Ist die Beschreibung überhaupt erfüllbar?

# Plan

Eigenentwicklung:

- Beschreibungsmodell aus Anforderungen.
- Beschreibungssprache evaluieren und entwickeln.
- Algorithmus zur Überprüfung der Erfüllbarkeit.
- Konfigurationsvalidierung.

- Parameterbeschreibung (Typ, Standardwert, Instanz spezifisch,...).
- Differenzierung zwischen Artefakten und Systemen.
- Optionale Funktionalität.
- Verschiedene Ausprägungen von Komponenten.

# Vorteile einer internen DSL

- Keine Einarbeitung in eine neue Technologie.
- Ausnutzung der Fähigkeiten und Integration der Hostsprache:
  - IDE-Unterstützung.
  - Fehlerbehandlung.
- Folge: Evaluation der Fähigkeiten.

# Programmiersprache

- JVM-Lauffähigkeit erforderlich.
- Javas DSL-Features ungenügend.
- Scala aufgrund der stärksten Ähnlichkeit als Alternative.
- Ausschluss von JRuby, Clojure und Groovy wegen mangelnder Kenntnisse.

# Scala als Programmiersprache der Wahl

- DSL-Fähgikeit (implicits, Operatorüberladen,...).
- Ausdrucksstärke (funktionale Elemente).
- Aufgeräumte Syntax.
- Firmenwunsch:
  - Kleines, internes Projekt.
  - Geschaffen für die Anforderungen.
  - Kenntnisse vorhanden.

# Evaluation der internen DSL

- Scala-Interpretation zu langsam.
- Kompilation verkompliziert Buildprozess in allen Projekten => größere Hürde für Akzeptanz.
- Vorteile der internen Sprache leider nicht nutzbar.
- Für interne Tests wiederverwendbar.
- Evaluation externer Sprache als Konsequenz.

# Evaluation der externen DSL

- Ausschluss von XML, YAML, JSON und UML scheiden wegen starrer Struktur.
- Parserkombinatoren im Vergleich zu Parsergeneratoren:
  - Ineffizient.
  - Für komplexe Fälle ungeeignet.
  - Unterlegene Fehlerbehandlung.
- ANTLR wegen sehr guter Dokumentation und IDE-Unterstützung als ausgereifter Parsergenerator.
- xText viel aufgrund mangelhaftem Buildmanagement und der Eclipsekopplung weg.

# Zusammenfassung Technologien

- Scala.
- ANTLR.

# Ausschluss von unerfüllbaren Beschreibungen



- Keine hilfreichen Fehlermeldungen bei fertigen Constraintsolvern.
- Konflikte nur bei verschiedenen Aussagen über denselben Parameter.
- Alle bedingten Parameteraussagen prüfen.
- Neue Erkenntniss für den nächsten Prüfdurchgang verwenden.