3. Praktikum: Modellierung von Informationssystemen

Andreas Krohn Benjamin Vetter Erik Andresen Jan Depke

5. Januar 2011

Inhaltsverzeichnis

1 Analyse

Welcher konkreter Konfigurator soll re-implementiert werden?

http://carconfig.toyota-europe.com/

Welche Schwachstellen sollen in der neuen Fassung vermieden werden?

- Auswahl des Modells soll in dem Konfigurator selbst möglich sein

- Wizard - Schrittweises Konfigurieren des Autos
- Menü rechts oben.. Sprechendere Namen, nähere Angaben → Pro Option eine Seite im Wizard

Welche Fahrzeuggrundtypen gibt?

Name
iQ
AYGO
Yaris
Urban Cruiser
Auris
Verso
Avensis
RAV4
Prius
Land Cruiser
Land Cruiser V8

Was ist konfigurierbar?

Welche Komponenten sind miteinander verbaubar?

2 Design

Architekturmodell

Wir haben eine Trennung von Frontend (Präsentationsschicht für Anwender) und Backend (Regelsystem) vorgenommen. Die Komponenten kommunzieren über ein HTTP-Ähnliches Protokoll.

2.0.1 Protokoll

Das Frontend schickt:

```
GET Liste von bereits gewaehlten, alphanumerischen Options-IDs
```

z.B.

```
GET yaris,three_doors
```

Das Backend antwortet, indem ein *OK* und eine Liste von Options-IDs zurückgegeben wird, die kompatibel mit den empfangen Options-IDs sind.

```
OK benziner,becker_radio,jvc_radio,...
```

Nach der Request-Methode (GET, OK, ERR) folgt ein Leerzeichen (ASCII-Wert 0x20). Elemente in der Liste werden durch ein Komma (ASCII-Wert 0x2C) getrennt. Jeder Request wird durch ein Carriage Return vor einem Newline abgeschlossen. (ASCII 0x0D 0x0A).

Abbildung 1: Netzwerkprotokoll Sequenzdiagramm

Arbeitspakete und Zuständigkeiten

Das Frontend wurde von Andreas Krohn und Benjamin Vetter bearbeitet. Das Backend wurde von Jan Depke und Erik Andresen bearbeitet.

Regelsystem

- Es gibt eine Sequenz von Kategorien (Chassis → Reifen → Lack → . . .)
- Es gibt eine Menge aktuell gewählter Optionen (und - implizit? - abgearbeteter Kategorien), die aktuelle *Konfiguration*
- Zu einer Konfiguration liefert das Regelwerk eine Menge noch verfügbarer Kategorien sowie jeweils wählbarer Optionen.

Klassenmodell / DB

- Es gibt eine Klasse *Option* (mögl. Ausprägungen: Chassis „Yaris“, Michelin Reifen „XYZ“, Metallic-Lackierung „Schwarz“. . .)
- Eine Option hat einen Identifier und gehört zu einer *Kategorie* (z.B. Radio, Dachfenster, Bereifung)

Abbildung 2: Modelle des Frotends

- Pro Option gibt es eine Liste von Identifiern, mit denen sie kombinierbar ist.

Das Frontend (Ruby on Rails) umfasst die Modelle *Option* und *Category*.

Die Modelle werden mittels OR-Mapper des Frameworks (ActiveRecord) in einer SQLite-Datenbank gespeichert und bedürfen daher keiner weiteren Erläuterungen. Zusätzlich gibt es ein Modell *RuleEngine*, dass für die Kommunikation mit dem Backend zuständig ist.

Der funktionale Teil des Frontends besteht aus der Klasse *WorkflowController*, der ein Wizard implementiert mit dem der Anwender konfrontiert wird und sein Auto konfigurieren muss (vgl. Screenshots).

GUI

Im folgenden sind einige Screenshots unserer webbasierten GUI zu sehen.

3 Implementierung

Für das Frontend wurde **Ruby on Rails 2.3.5** mit einer SQLite-Datenbank verwendet. Die GUI ist dementsprechend webbasiert. Das Backend (Regelsystem) wurde in Java realisiert.

4 Tests

4.1 Frontend

Für das Frotend wurden Unit-Tests und Functional-Tests erstellt (vgl. /frontend/test/unit). Die Unit-Tests testen die Abfragen an das Regelsystem. Unit-Tests für die Modelle *Option* und *Category* wurden nicht über die automatisch generierten Tests hinaus erstellt, da die Models keinen applikationsspezifischen Code aufweisen. Die Functional-Tests testen den *WorkflowController* (vgl. /frontend/test/functional).

Das Test-Protokoll für das Frontend:

```
$ rake test
/usr/bin/ruby1.8 -I"lib:test" "/usr/lib/ruby/1.8/rake/rake_test_loader.rb" "test/unit
```

```
Loaded suite /usr/lib/ruby/1.8/rake/rake_test_loader
Started
....
Finished in 0.049247 seconds.

4 tests, 4 assertions, 0 failures, 0 errors
/usr/bin/ruby1.8 -I"lib:test" "/usr/lib/ruby/1.8/rake/rake_test_loader.rb" "test/func
Loaded suite /usr/lib/ruby/1.8/rake/rake_test_loader
Started
....................
Finished in 0.264087 seconds.

20 tests, 31 assertions, 0 failures, 0 errors
/usr/bin/ruby1.8 -I"lib:test" "/usr/lib/ruby/1.8/rake/rake_test_loader.rb"
```

4.2 Backend

Für das Backend existiert ein Python-Skript nettest/nettest.py das alle möglichen Kombinationen in mehreren Threads durchtestet.

5 Komponenten/Schnittstellen

Regelengine

Aktion:

Parameter: Liste von (bereits gewählten) Optionen

Rückgabe: Liste von wählbaren Optionen

Datenbank

6 sonstiges...

- Modelldatenbank
- Teile und Konfigurationsoptionen (Farbe, etc..) in DB
- Kombinierbarkeit/Konfigurierbarkeit/Regeln in XML-Dateien (die dann Modelle/Teile.. referenzieren)
- Verbaubarkeitsregeln in gesondertem Editor?
- Ausblick: Workflow/Ablauf konfigurierbar