

3. Praktikum: Modellierung von Informationssystemen

Andreas Krohn Benjamin Vetter Erik Andresen Jan Depke

6. Januar 2011

Inhaltsverzeichnis

1	Analyse	1
2	Design	2
2.0.1	Protokoll	2
3	Implementierung	9
4	Tests	9
4.1	Frontend	9
4.2	Backend	10
5	Komponenten/Schnittstellen	10
6	sonstiges...	10

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 kommunizieren über ein HTTP-Ähnliches Protokoll.

2.0.1 Protokoll

Das Frontend schickt:

GET Liste von bereits gewählten, 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).

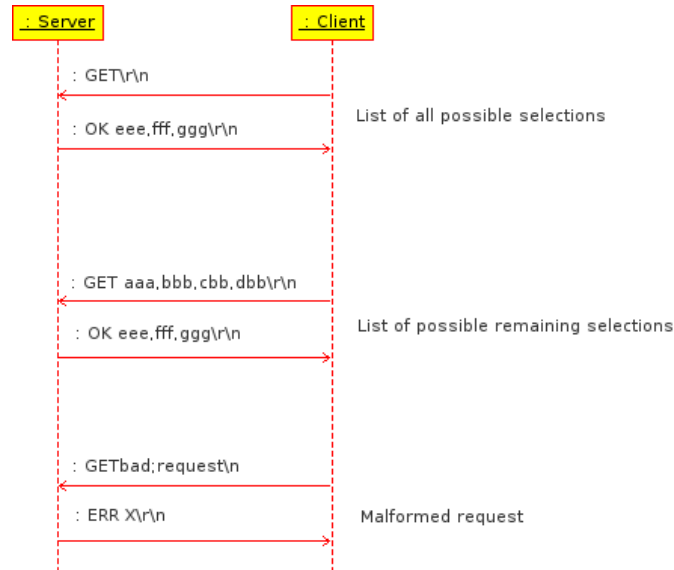


Abbildung 1: Netzwerkprotokoll Sequenzdiagramm

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? - abgearbeiteter 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)

Frontend Models

werden mittels OR-Mapper
(ActiveRecord) abgebildet

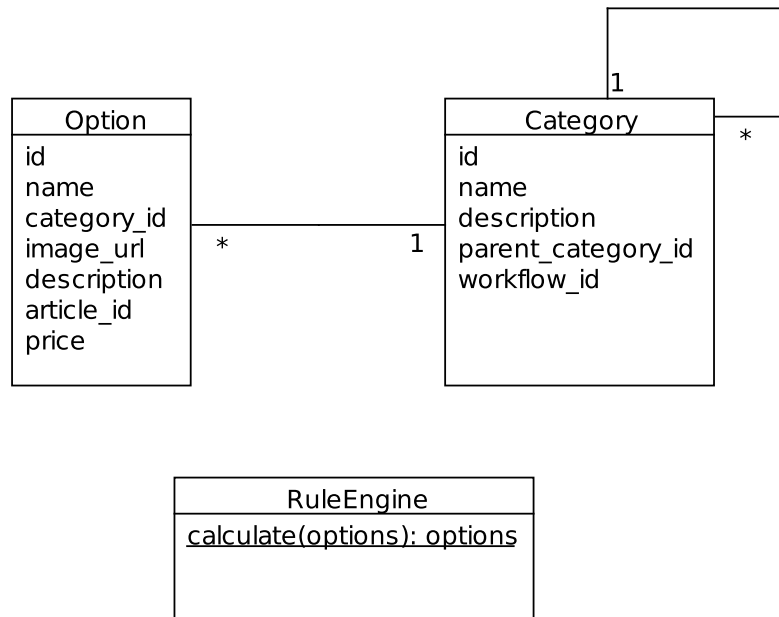


Abbildung 2: Modelle des Frontends

- 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.

Schritt 1

Aktueller Preis: \$0.00

Bitte wählen Sie: **Modell**

Das Modell bestimmt die Grundkonfiguration des Autos und beinhaltet serienmäßigen Füllfanz.



Bild	Auswahl	Name	Beschreibung	Preis
	Auswählen	AYGO		\$8,000.00
	Auswählen	Yaris		\$10,000.00

Schritt 2

Aktueller Preis: \$8,000.00

Bitte wählen Sie: **Karosserie**

Die Karosserie bestimmt die Geräumigkeit Ihres neuen Toyotas.

Bild	Auswahl	Name	Beschreibung	Preis
	Auswählen	3 Türer	Mit dem 3 Türer haben Sie 3 Türen. Tatsächlich!	\$10,000.00
	Auswählen	5 Türer	Mit dem 5-Türer haben Sie sogar 5 Türen.	\$13,000.00

Bereits gewählt:



Bild	Name	Beschreibung	Preis	Aktion
	Modell: AYGO		\$8,000.00	Löschen

Schritt 3

Aktueller Preis: \$10,000.00

Bitte wählen Sie: **Motor**

Mehr hilft mehr: Der Motor gibt Kraft.

Bild	Auswahl	Name	Beschreibung	Preis
	Auswählen	Diesel		\$11,000.00
	Auswählen	Benziner		\$12,000.00

Bereits gewählt:




Bild	Name	Beschreibung	Preis	Aktion
	Modell: AYGO		\$8,000.00	Löschen

Schritt 5

Aktueller Preis: \$12,000.00

Bitte wählen Sie: Felgen

Erst die Felgen machen das Auto prallig.

Bild	Auswahl	Name	Beschreibung	Preis
	Auswählen	Standard Felgen	Machen nicht so viel her	\$12,000.00
	Auswählen	MSW Felge	Und noch eine Felge	\$12,300.00
	Auswählen	AEZ Felge	Noch ne Felge	\$12,500.00
	Auswählen	Datz Felge	Hochwertige Felgen aus irgendeinem Material	\$12,600.00

Schritt 8



Aktueller Preis: \$12,550.00

Bitte wählen Sie: **Navigation**

Das Navi führt sie immer an den richtigen Ort

Bild	Auswahl	Name	Beschreibung	Preis
	Auswählen	Kein Navigation	Dass sie sich da mal nicht verirren	\$12,550.00
	Auswählen	Modion Navigation		\$12,630.00
	Auswählen	Mio Navigation		\$12,640.00






Bereits gewählt:

Bild	Name	Beschreibung	Preis	Aktion
	Modell: AYGO		\$8,000.00	Löschen
	Karosserie: 3 Türen	Mit dem 3 Türen haben Sie 3 Türen, Tatsächlich!	\$2,000.00	Löschen

Auswahl abgeschlossen!

Aktueller Preis: \$12,630.00

Bereits gewählt:

Bild	Name	Beschreibung	Preis	Aktion
	Modell: AYGO		\$8.000.00	Löschen
	Karosserie: 3 Türen	Mit dem 3 Türen haben Sie 3 Türen, Totschlecht	\$2.000.00	Löschen
	Motor: Benzinler		\$2.000.00	Löschen
	Felgen: MSW Felge	Und noch eine Felge	\$300.00	Löschen
	Farbe: Rot	Rot	\$0.00	Löschen

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 Frontend 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" -r"/usr/lib/ruby/1.8/rake/rake_test_loader.rb" -rtest/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/fun
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