

Entwickler-Schulung 11.04.2014

kivitendo

Bernd Bleßmann



Themen

- MVC
- Rose
- Controller

- Task-Server

MVC

Model	View	Controller
Datenbank	Templates	Steuerung
Rose	(CSS)	Interaktion

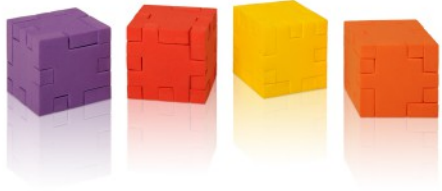


Bisher: Frontend, Backend, Views

Frontend-Funktionen in /bin/mozilla/xy.pl

Backend-Routinen in SL/XY.pm

Views in templates/webpages/xy



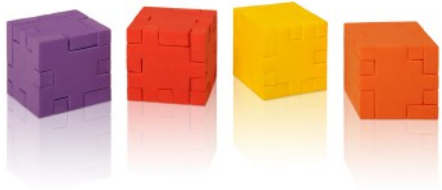
Nachteile

Verwendung pseudo-globaler Variablen

Enge Verzahnung zwischen Frontend- und Backend. Geringe Wiederverwendbarkeit.

Uralt-Code mit Verzahnung aus Layoutcode und Programmcode (HTML in Perl)

Backendcode bereitet Layoutcode vor



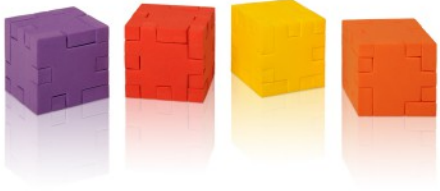
Mehr Nachteile

**Auswirkung kleiner lokaler Änderungen
nicht abschätzbar**

**Designprobleme wie Feststellung, welche
Funktion durch Buttondruck aufgerufen wird**

**Daten werden an verschiedensten Stellen
formatiert und geparst (Zahlenformate)**

Kaum Helfer-Routinen; Wiederholung



Ziel: MVC, DRY und andere

MVC: Model, View, Controller

DRY: Don't repeat yourself

Convention over configuration



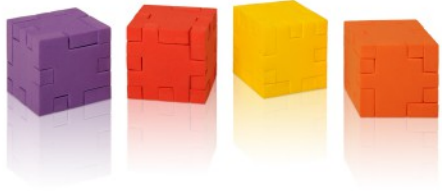
Models

Abstraktion der Datenbankzugriffe durch objektrelationalen Mapper und Abbildung in Perl-Objekthierarchie

Wir verwenden: Rose, Rose::DB und Rose::DB::Object (kurz: RDBO)

Hohe Wiederverwendbarkeit

Viele kleine Funktionen, dadurch DRY



Views

Layouting-Code für HTML und andere (z.B. Text-E-Mails)

Weiterhin durch Perl-Modul Template realisiert

Benutzung von RDBO-Objekten in Template analog zur Benutzung einfacher Hashes

Layout-Helfermodul „L“



Controller

**Schnittstelle zwischen Models und Views:
Aufbereitung von Daten, Durchführen von
Aktionen**

**Der Code, der durch Benutzerinteraktion
aufgerufen wird**

Convention over configuration

Eigenentwicklung



Vorteile

Sehr schnelle Codeentwicklung

Ermöglicht Schreiben automatischer Tests

Klare Trennung, wo Daten geparkt und formatiert werden müssen



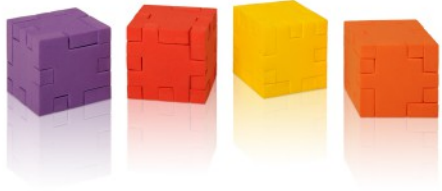
Nachteile

Durch viele Perl-Modulabhängigkeiten hohe Startup-Zeiten pro Request (bis 4 Sekunden)

-> Durch Einsatz von FastCGI kein Problem mehr

Höhere Installationsanforderungen

-> Benötigte Module mitliefern



Neue Struktur

SL/DB
SL/Controller
templates/webpages

Models
Controller
Views (wie bisher)

Helfermodule:
SL/DB/Helper
SL/Controller/Helper
SL/Template/Plugin

Models
Controller
Views



Rose

(Rose::**DB::Object** oder RDBO)

Relationale Datenbank Objekt – Mapper

id	name	customernumber
1001	Meier	3011
1002	Müller	3012

Tabelle: customer



Objekt: customer

```
...  
$customer->load;  
my $oldname = $customer->name;  
$customer->name('Maurer');  
$customer->save;  
...
```

Console zum Testen

```
./scripts/console -l bernd -C Bernd_Current
```


Hands-On History-Controller

Erstmal nur anzeigen ...

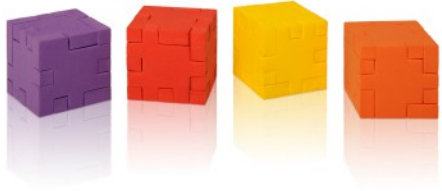
- `SL/Controller/History.pm`
- `templates/webpages/history/list.html`



Hands-On History-Controller

Dann filtern ...

- `SL/Controller/History.pm`
- `templates/webpages/history/list.html`



Methodenaufruf bei Controllern

Nur Subs von außen aufrufbar, deren Name mit „action_“ beginnt

Browser muss „controller.pl“ aufrufen

Parameter „action“ hat die Form „Controllername/actionname“

Beispiel: „action=Customer/list“ ruft „sub action_list“ in „SL/Controller/Customer.pm“



Hands-On History-Controller

Und auch bearbeiten ...

- `SL/Controller/History.pm`
- `templates/webpages/history/list.html`
- `templates/webpages/history/edit.html`

Hands-On History-Controller

was fehlt noch?

- Menü-Eintrag
- Übersetzungen



Neue Texte übersetzen lassen

1. Text in Quelltext/View verwenden
2. `./scripts/locales.pl de`
3. In locale/de/missing übersetzen
4. `./scripts/locales.pl de`



Übersetzungen korrigieren

1. `locale/de/all` bearbeiten
2. `./scripts/locales.pl de`



Interpolation

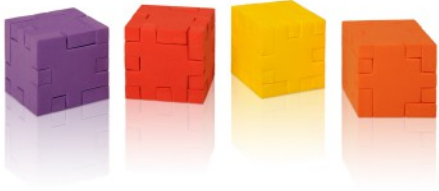
Problem: Übersetzungen haben andere Satzstellung als Originaltext; dynamische Werte sollen eingesetzt werden

Lösung: Platzhalter #1, #2, #3 etc und Ersetzen durch Argumente zu \$locale->text()

Beispiel:

Original: „Customer #1 phone #2“

Übersetzung: „Telefon #2 für Kunde #1“



Übersetzen im Quelltext

```
$::locale->text(„Phone number for #1 is #2.“,  
               $customer->name,  
               $customer->phone)
```

**Funktioniert in Frontendcode, Backendcode,
Models, Controllern, externen Scripten**



Übersetzen in Views

Erste Möglichkeit:

```
[% USE LxERP %]  
[% LxERP.t8('Customer #1, phone #2',  
customer.name, customer.phone) %]
```

Zweite Möglichkeit (keine Interpolation):

```
[% USE T8 %]  
[% 'List open invoices' | $T8 %]
```



Dokumentation

→ Rose-DB-Object:

→ <http://search.cpan.org/~jsiracusa/Rose-DB-Object-0.811/lib/Rose/DB/Object/Tutorial.pod>

→ `man Rose::DB::Object::Tutorial`

→ `perldoc SL/DB/Object.pm`

→ Controller:

→ `perldoc SL/Controller/Base.pm`

→ Templates:

→ `perldoc SL/Template/Plugin/L.pm`

→ `man Template::Manual`

Hands-On History-Controller

Erweitern der Tabelle (um „mtime“)

- Datenbank-Upgrade
- Rose-Meta-Setup neu erzeugen
- Controller anpassen

History-Controller

Datenbank-Upgrade

- Nicht direkt die DB ändern, sondern Upgrade-Skript schreiben
- Vorhandenes Skript als Template kopieren
- Oder `./scripts/dbupgrade2_tool.pl` nehmen

History-Controller

Datenbank-Upgrade

```
./scripts/dbupgrade2_tool.pl \  
--create history_erp_add_mtime \  
--type sql \  
--description "Erweitern der history_erp um mtime" \  
--depends release_3_1_0
```


History-Controller

Datenbank-Upgrade

- (erneut) an kivitendo anmelden
- Rose-Meta-Setup neu erzeugen
`./scripts/rose_auto_create_model.pl \
--client Bernd_Current history_erp`

History-Controller

Controller anpassen

Als Übung

(Hinweis: Das Modell ist geändert – muss wirklich der Controller angepasst werden?)



Neue Tabelle

- Gleiche Schritt wie bei einer neuen Spalte
- Namens-Konventionen beachten
SQL-Tabelle im plural, Objekt im Singular
customers vs. `SL::DB::Customer`
- Eintrag in `SL/DB/Helper/ALL.pm`
- Eintrag in `SL/DB/Helper/Mappings.pm`
hier können „unkonventionelle“ Namen angepasst werden

Dokumentation

- Offizielle Dokumentation: „SQL-Upgradedateien“
- `./scripts/dbupgrade2_tool.pl --help`
- `./scripts/rose_auto_create_model.pl --help`

Task-Server

- Führt zu bestimmten Zeiten Aufgaben aus
(→ Hintergrund-Jobs)
- Wird schon für wiederkehrende Rechnungen
und CSV-Import verwendet
- Kann für Selbsttests verwendet werden

Hands-On Hintergrund-Job

automatisches Anpassen des
Nummern-Kreises für Rechnungen

→ `SL/BackgroundJob/NumberRangeUpdate.pm`

Hands-On Hintergrund-Job

Rückgabe-Wert und Fehler

Dokumentation

- In der offiziellen Doku: „Der Task-Server“
- `perldoc SL/BackgroundJob/Base.pm`

Was gibt es noch

- Debug-Meldungen
- Tests
- getModels



Richardson
&
Büren

ihr partner für  kivitendo