12.–15.09.2010
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Fahrstuhl zum Web

Web-Anwendungen mit Lift

Markus Knittig
Universität Stuttgart

Mirko Stocker
Hochschule Rapperswil

Die Referenten

- Markus Knittig ist Softwaretechnik Student an der Universität Stuttgart
  - Scala Enthusiast
  - Mitinitiator der Scala Users Southern Germany
- Mirko Stocker ist Wissenschaftlicher Mitarbeiter am Institut für Software der Hochschule Rapperswil, CH.
  - Masterarbeit „Scala Refactoring“ Projekt
  - Mitarbeit an Eclipse Plug-Ins wie CDT, RDT, Scala IDE.

Themen

- Einführung in Lift
  - Schwerpunkt Templates und Binding
- Interaktives Web mit Comet
- Weitere interessante Lift Features
- Ausblick

Einführung in Lift

„Lift is an expressive and elegant framework for writing web applications„

Einführung in Lift

- Webframework basierend auf der Scala Programmiersprache
- gestartet von **David Pollak** in 2007
  - frustriert von Ruby on Rails, da zu ungewohnt für Entwickler und Performanceprobleme
- Schwerpunkte auf Sicherheit, Skalierbarkeit und Performance
- Integration von Ajax/Comet

Warum Scala?

- Scalas Typsystem hilft, Fehler früh zu finden
- knappe Syntax, ohne viel Ballast und Altlasten
- mächtige Möglichkeiten um Abstraktionen zu schreiben → DSLs
- kompiliert zu Java Bytecode → einfaches Deployment auf bestehender Java-Infrastruktur
- Scala hat XML-Literale,
  - Funktionale Programmierung eignet sich hervorragend um strukturierte Daten wie XML zu transformieren

Brauchen wir noch ein Framework?

- Übernimmt das Beste aus anderen Frameworks
  - Rails: minimale Konfiguration nötig um loszulegen, Generatoren für einfachen Start
  - Wicket: Templating, View-First
  - Django: mehr als nur CRUD, vorgefertigte Klassen
  - Seaside: Sicherheit
- trotzdem gut integrierbar in bestehende Java-Welt
  - „it's just another JAR“
- bestehende Libraries wiederverwenden

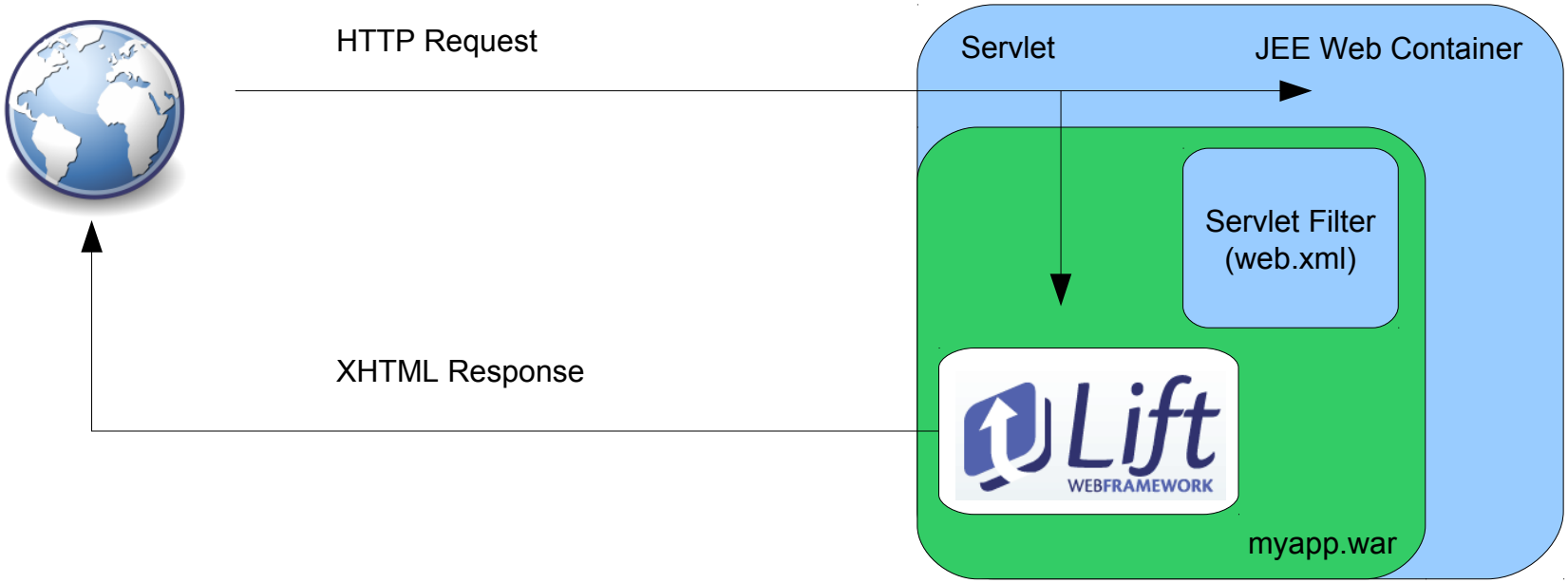
→ entscheiden Sie selbst!

Grundideen von Lift

- Abstraktion des HTTP Request–Responses Zyklus
  - lässt den Programmierer auf die Businesslogik konzentrieren
- „do the right/common thing by default“
  - Security out-of-the-box
- View first
  - dazu gleich mehr
- Jedem UI Element im Browser wird eine GUID zugewiesen, welche auf dem Server mit einer Funktion korrespondiert.
- Lift ist grundsätzlich Stateful.

Lift Architektur

- Lift Applikation als *.war Datei
- Lift hängt sich per Servletfilter ins HTTP Processing ein
- Auf allen JEE-Containern lauffähig

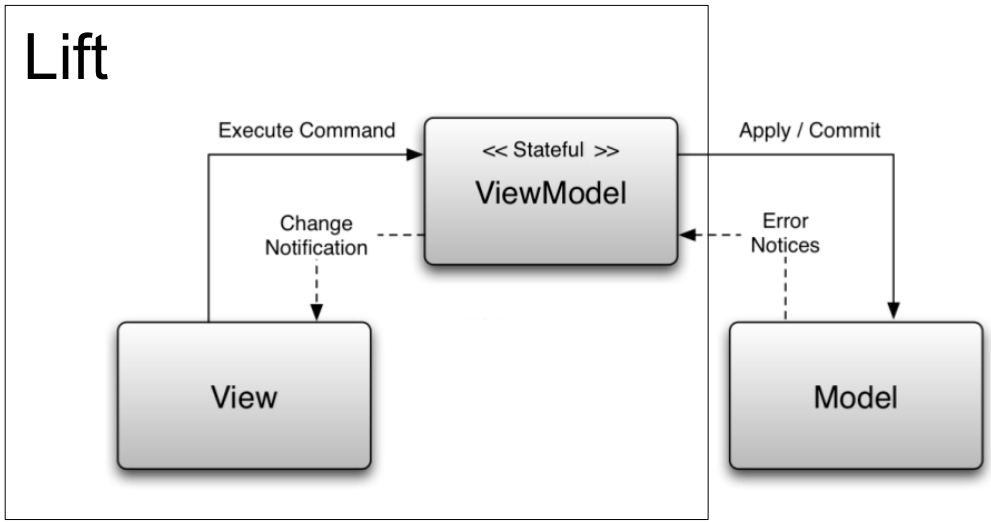


View First

- Primärziel: keine Programmlogik in der View
- Die View ist immer pures XML:
  - gut für Designer
  - gut um mit Tools wie Dreamweaver zu bearbeiten
  - also auch keine Kontrollstrukturen in der View (vgl. JSP Expression Language)
- View first entspricht *nicht* dem Model-View-Controller Pattern
  - MVC hat pro Seite einen dedizierten Controller.
  - Bei View-First ist die View die erste Anlaufstelle.
    - Mehrere dynamische Teile auf einer Seite sind kein Problem

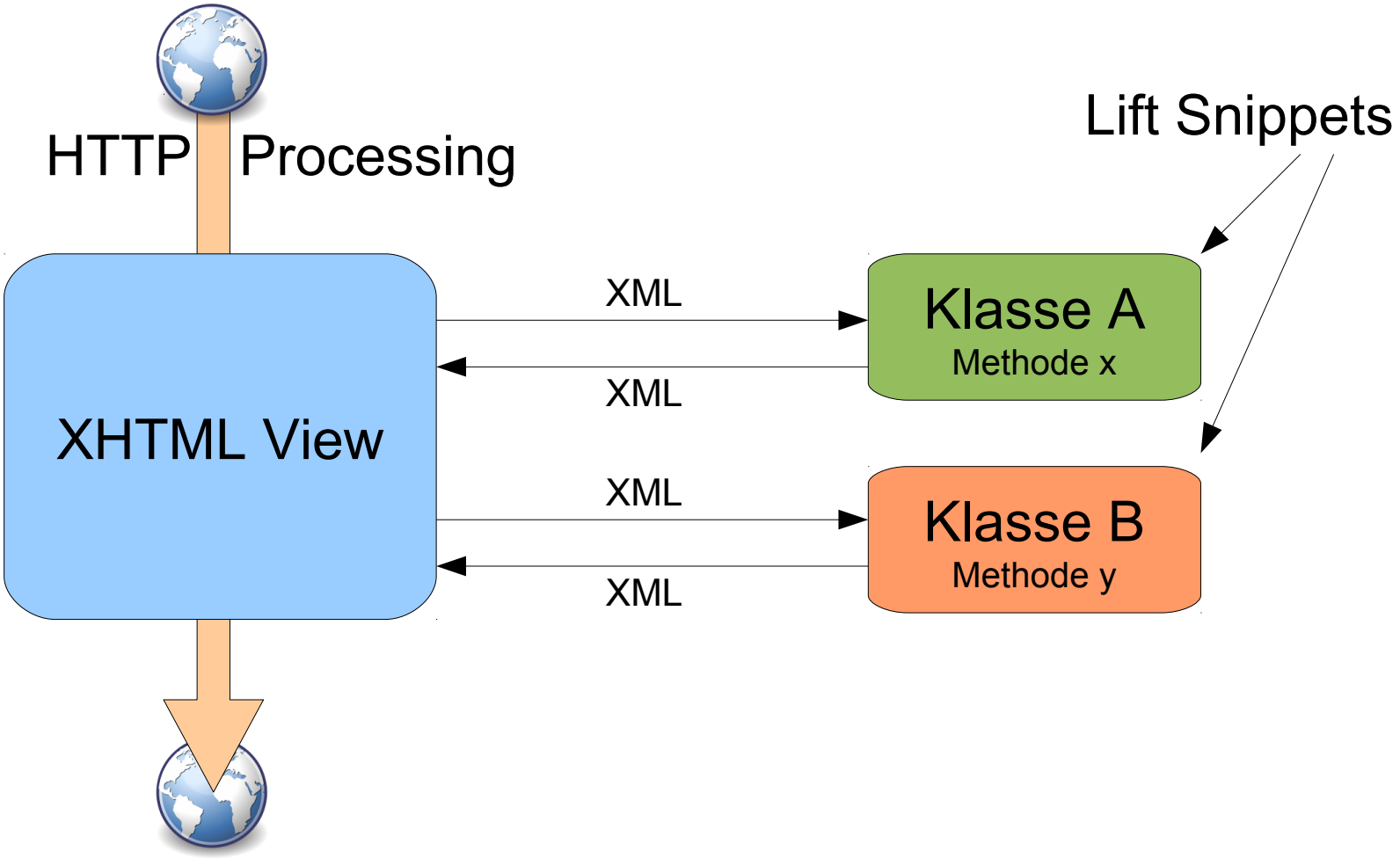
Model-View-ViewModel

- View: XHTML Template
- ViewModel: Scala Klasse die Daten aufbereitet
- Model: Scala oder Java

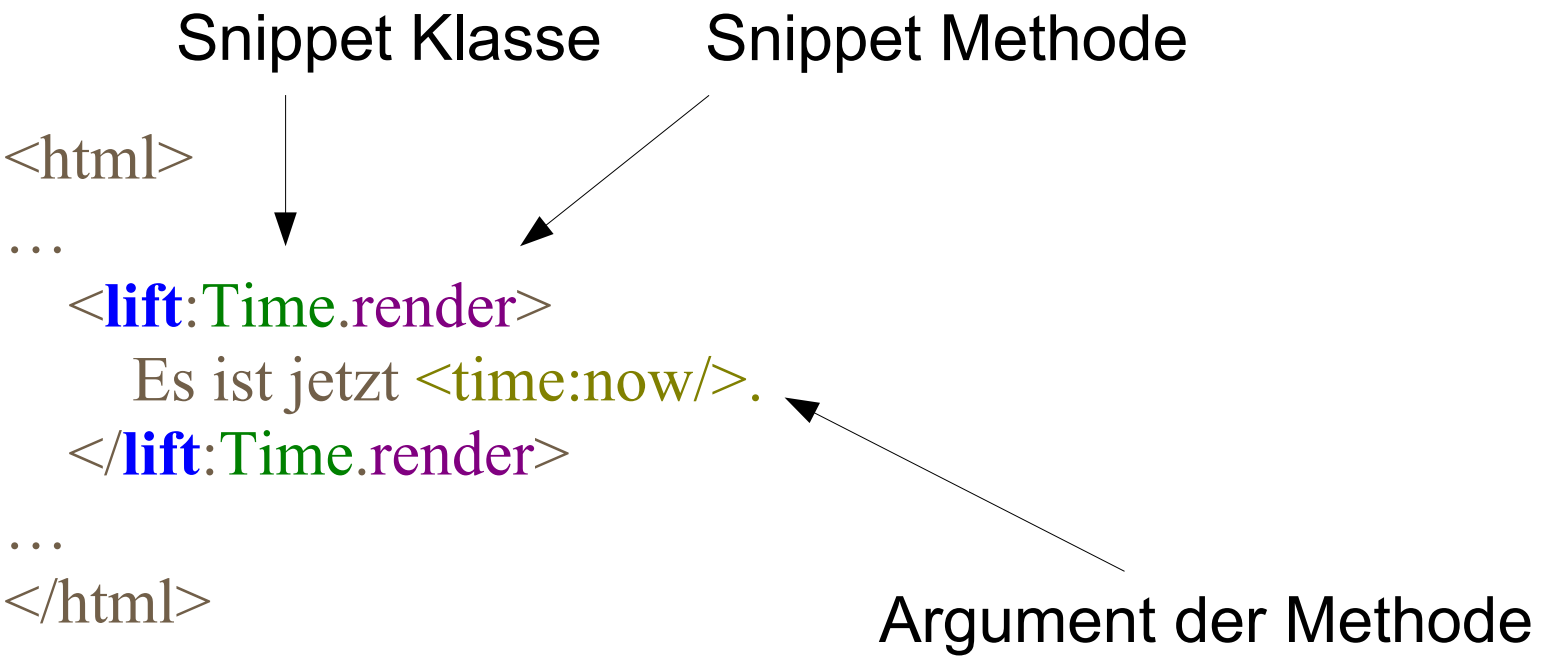


Basierend auf „Lift in Action“ – Timothy Perret

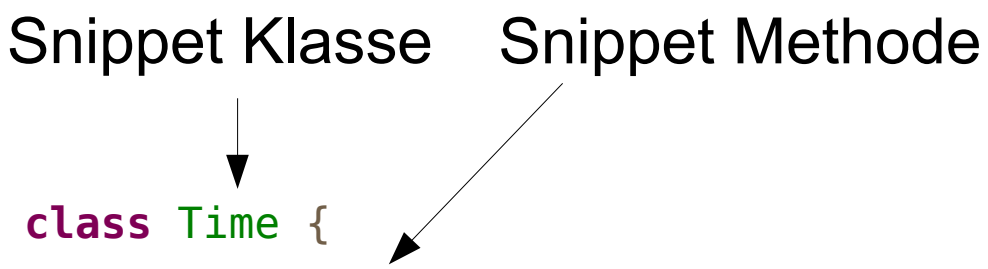
View First


HTTP
Processing
Lift Snippets
XHTML View
XML
XML
XML
XML
Klasse A
Methode x
Klasse B
Methode y

Hello World View



Hello World Snippet




```
class Time {
    def render(xml: NodeSeq): NodeSeq = {
        val currentTime = TimeHelpers.now.toString
         bind("time", xml, "now" -> currentTime)
    }
}
```

Hello World Ablauf

```
<html>
…
  <lift:Time.render>
    Es ist jetzt <time:now/>.
  </lift:Time.render>
…
</html>
```

```
class Time {
    def render(xml: NodeSeq) : NodeSeq = {
        val currentTime = TimeHelpers.now.toString
        bind("time", xml, "now" -> currentTime)
    }
}
```

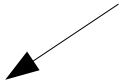
```
<html>
…
    Es ist jetzt Thu Sep 09 14:00:00 PDT 2010.
…
</html>
```

Formulare mit Callbacks

```
<html>
…
   <lift:HelloWorld.howdy form="GET">
     <person:name /> <person:submit />
   </lift:HelloWorld.howdy>
…
</html>
```

Formulare mit Callbacks

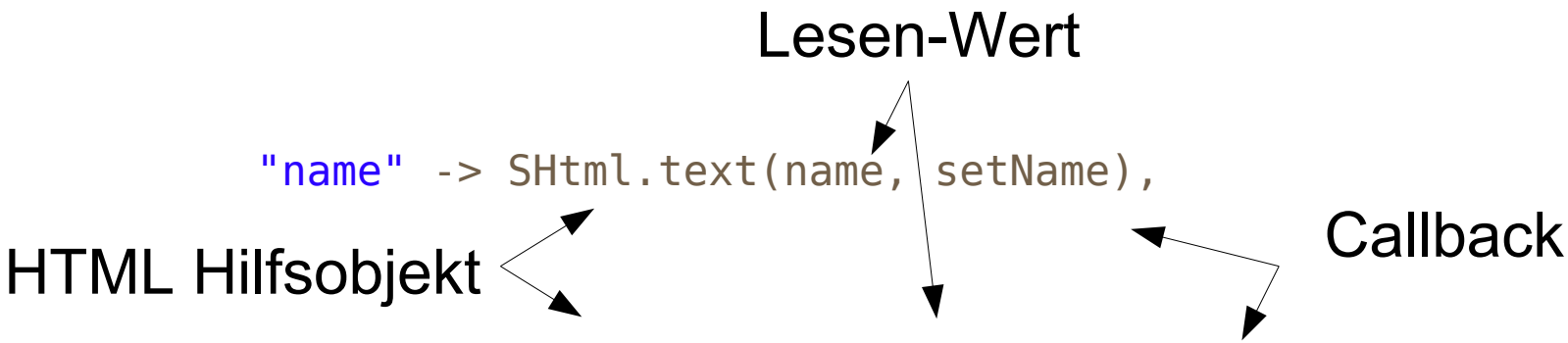
gewöhnliche Scala Methoden



```
class HelloWorld {

  var name = "Ihr Name"

  def setName(name: String) = this.name = name

  def logName() = Log.info(name)

  def howdy(xml: NodeSeq): NodeSeq =

     bind("person", xml,
      "name" -> SHtml.text(name, setName),
      "submit" ->  SHtml.submit("Senden", logName))
  }
}
```

Formulare mit Callbacks

```
def howdy(xml: NodeSeq): NodeSeq =
   bind("person", xml,
        "name" -> SHtml.text(name, setName),
        "submit" ->  SHtml.submit("Senden", logName))
}
```



Formulare mit Callbacks 2.0

```
<html>
…
   <lift:HelloWorld.howdy form="GET">
     <person:name /> <person:submit />
   </lift:HelloWorld.howdy>
…
</html>
```

immer noch exakt gleich

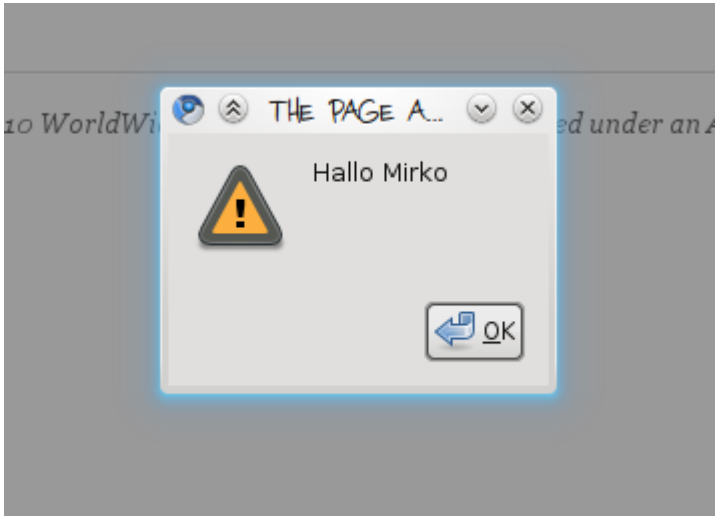
Formulare mit Callbacks 2.0

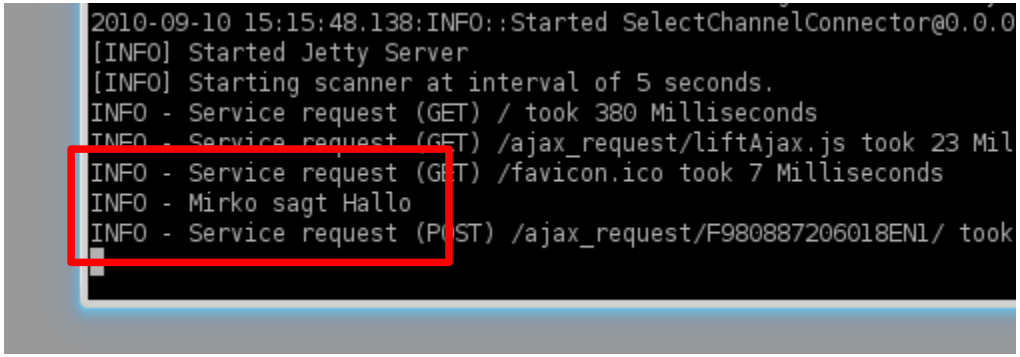
```
class HelloWorld {

  var name = "Ihr Name"

  def setName(name: String) = this.name = name

  def logName() = {
    Log.info(name)
    Alert("name"+ name)
  }

  def howdy(xml: NodeSeq): NodeSeq =

     bind("person", SHtml.ajaxForm(xml),
      "name" -> SHtml.text(name, setName),
      "submit" ->  SHtml.ajaxSubmit("Senden", logName))
  }
}
```

JavaScript Alert

Ajax Helfer

Formulare mit Callbacks 2.0





Comet

Demo

SiteMap

- Routing und Zugriffskontrolle in Lift.
- SiteMap bietet eine zentrale Übersicht über alle verfügbaren Seiten meiner Applikation.
- Anhand der SiteMap wird auch die Navigation auf der Webseite generiert.
- Menüeinträge können sich dynamisch verhalten.

```
val sitemap = SiteMap(
  Menu("Home") / "index",
  Menu("Search") / "search",
  Menu("History") / "history",
  Menu("Admin") / "admin" >> If(
    () => isUserLoggedIn(),
    () => RedirectResponse("/login")))

 LiftRules.setSiteMap(sitemap)
```

Persistenz in Lift

- Lift hat zwei eigene ORM:
  - Mapper (der Alte)
  - Record (der Nachfolger)
- Vergleichbar mit Rubys Active Record
- Backends können z.B. MySQL, Postgres, Oracle sein.


```
class Post extends LongKeyedMapper[Post] with IdPk {
  object title extends MappedString(this, 140)
  object contents extends MappedText(this)
  object published extends MappedBoolean(this)
  …
}
```

- Alternativ kann auch JPA einfach eingesetzt werden.

Typsichere SQL-Abfragen

Anfragen an Mapper sollte nie direkt über SQL, sondern über die bereitgestellten Funktionen geschehen:

```
val publishedPosts: List[Post] =
  Post.findAll(By(Post.published, true))
```

Dies entspricht folgender Abfrage:

```
SELECT * FROM post WHERE post.published = true;
```

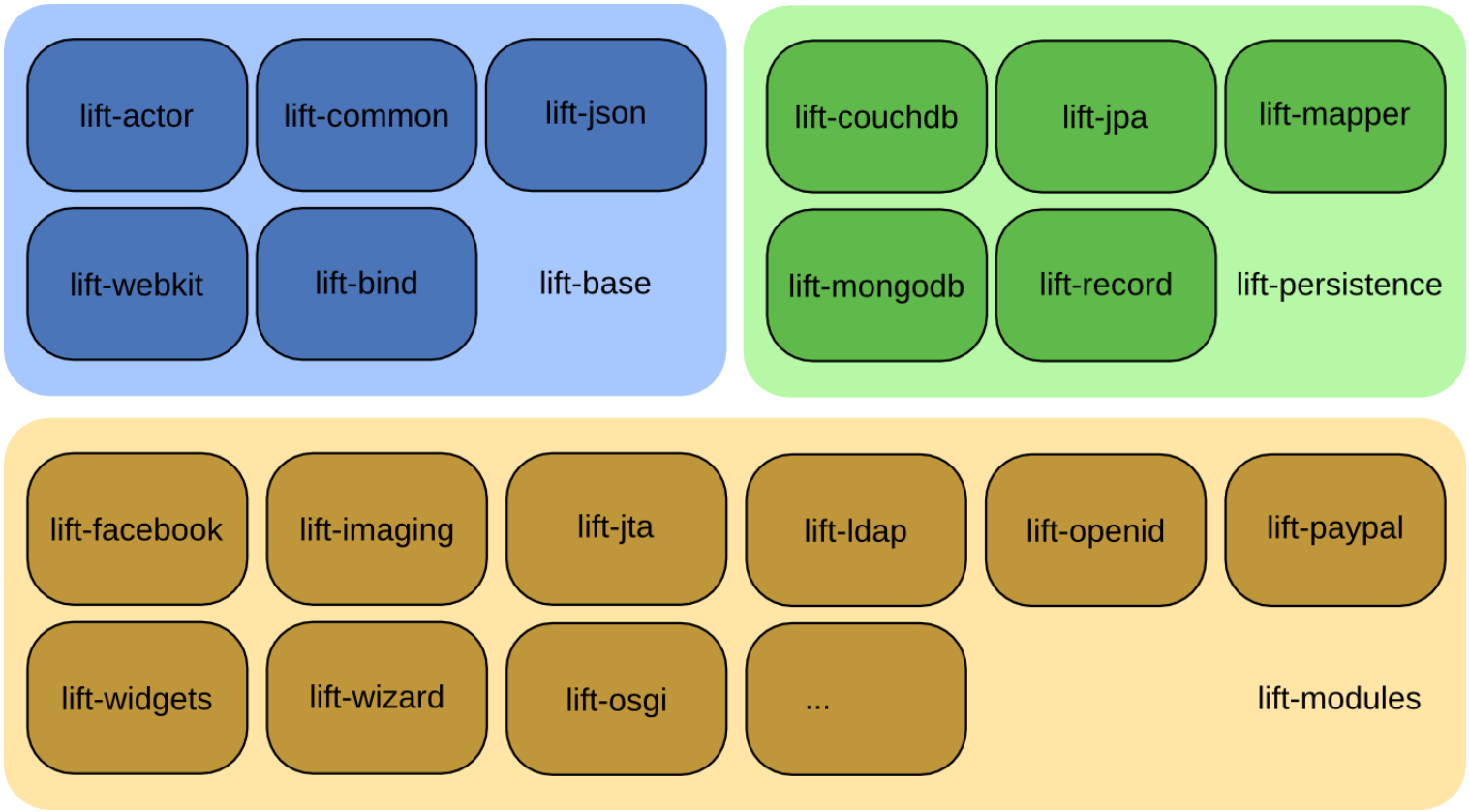
mit dem Vorteil, dass die Abfrage vom Compiler geprüft wird.

Typsichere SQL-Abfragen

- Lift bietet verschiedenste Möglichkeiten, um solche Abfragen zusammenzustellen.
- Falls nötig, kann man immer noch auf SQL zurückgreifen.
- Auch das erstellen und verändern von Datensätzen ist sehr komfortabel gelöst:

```
val thePost: Post = …

val newComment =
  Comment.create.author("Markus").comment("This is great!")

post.comments += newComment

post.save
```

Weitere Lift-Bausteine

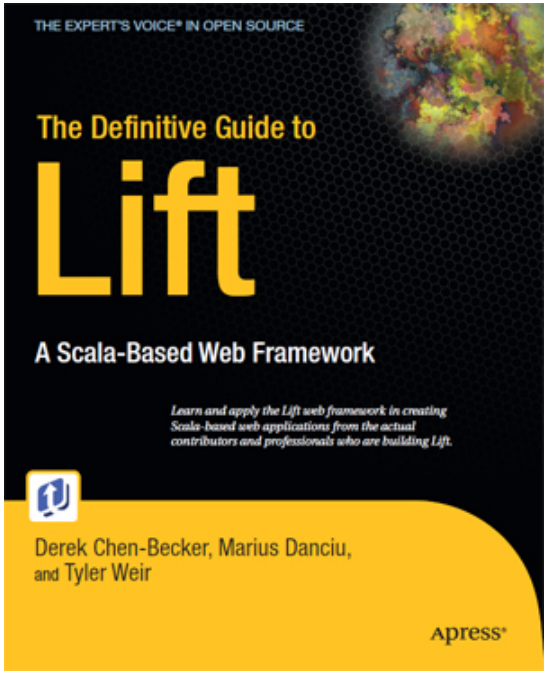


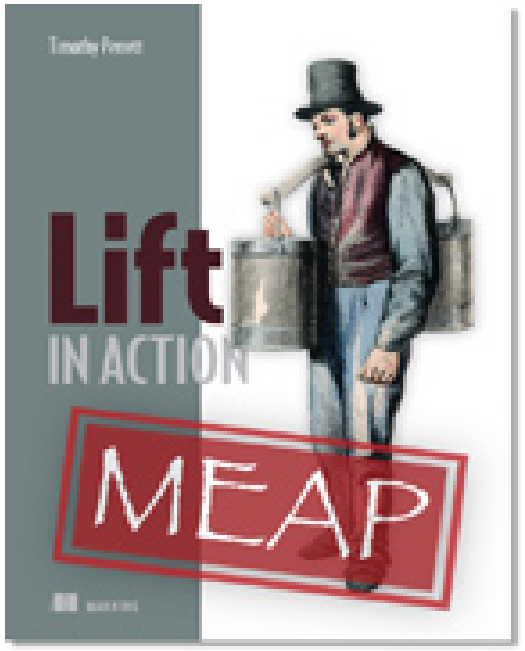
Wer setzt Lift ein?

- Foursquare
  - ortsbezogenes soziales Netzwerk
  - 3 Millionen Benutzer
- Novell Pulse
  - Google Wave für Unternehmen
- Apache ESME
  - Enterprise Social Messaging Experiment
  - entwickelt von Siemens und SAP

Weitere Informationen zu Lift

- http://liftweb.net
- http://groups.google.com/group/liftweb





Vielen Dank!

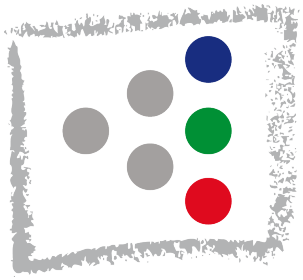
Markus Knittig
Universität Stuttgart

Mirko Stocker
Hochschule Rapperswil

Scala Users Southern Germany

- Community: http://scala-southerngermany.mixxt.de/
- Ziel: Austausch sowie Verbreitung von Scala fördern
- Stand
  - Ein Treffen in München im Sommer
  - Diesen Donnerstag treffen in Stuttgart
- Beitreten und Mitmachen!

Hochschule Rapperswil



INSTITUTE
FOR
SOFTWARE

- Refactoring Tools für Scala, C++, Ruby, Groovy
- Code-Analyse und Verbesserung für C++

ifs.hsr.ch

Advanced Binding Beispiel

```
<lift:Person.family>
        <p><person:name />'s children:</p>
        <ul>
                <person:children>
                <li><child:cname /></li>
                </person:children>
        </ul>
</lift:Person.family>
```

Advanced Binding Beispiel

```
def family(xhtml: NodeSeq): NodeSeq = {

  persons flatMap { person =>

    val children = person.children flatMap { child =>


      val childrenTemplate =
        chooseTemplate("person", "children", xhtml)

      bind("child", childrenTemplate,
           "cname" -> Text(child.name))
    }

    bind("person", xhtml,
         "name"     -> Text(person.name),
         "children" -> children)
  }
}
```

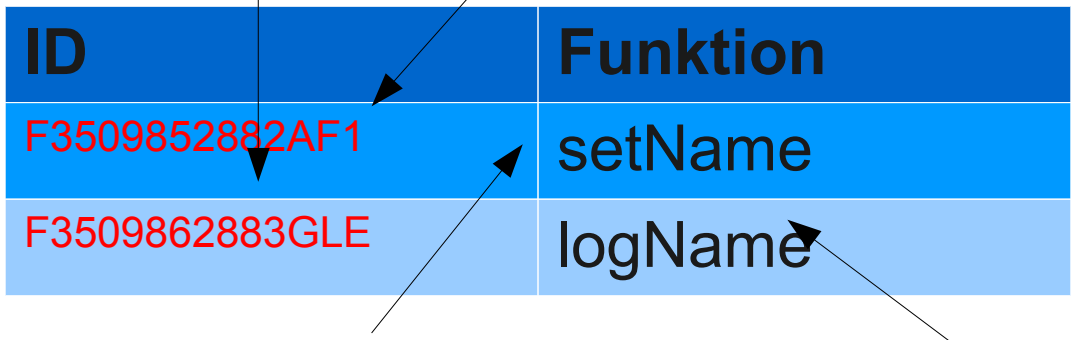
<child:cname />

Advanced Binding Beispiel

```
<p>Paul's children:</p>
<ul>
        <li>Thomas</li>
        <li>Lukas</li>
</ul>

<p>Peter's children:</p>
<ul>
        <li>Max</li>
        <li>Moritz</li>
</ul>
```

Magische Eingabefelder?

```
<form action="/index" method="post">
   <input value="Ihr Name" type="text" name="F3509852882AF1" />
   <input name="F3509862883GLE" type="submit" value="Senden" />
</form>
```




```
SHtml.text(name, setName)
```

```
SHtml.submit("Senden", logName)
```

Magische Eingabefelder?