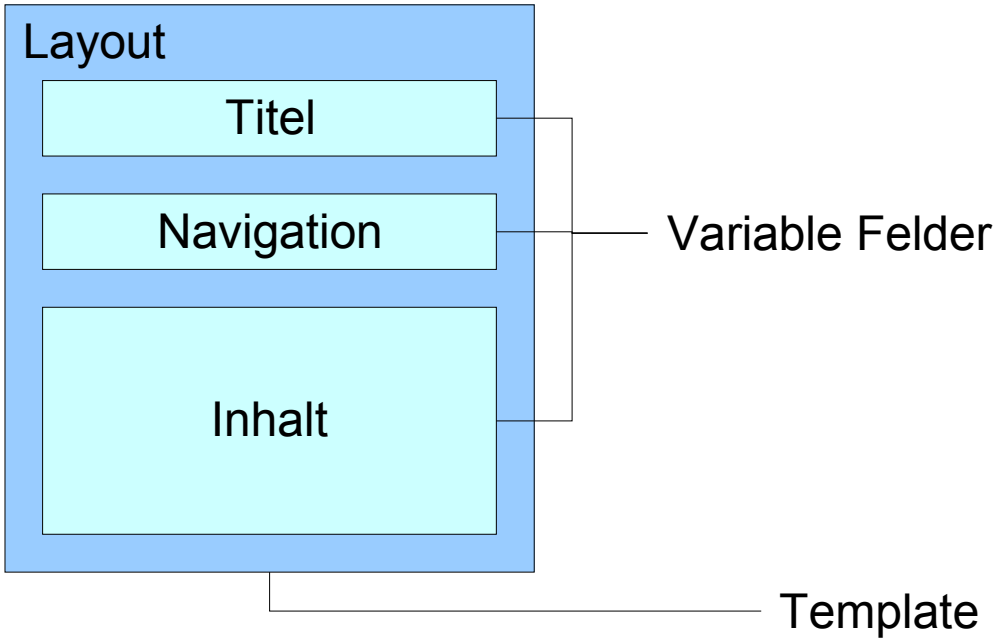
Hauptprobleme, die ich bei der Web-Entwicklung lösen wollte:

- Trennung von Darstellung, Code und Daten (MVC-like)
  - Ermöglicht leichtere Wiederverwendbarkeit der drei genannten Komponenten in anderen Projekten
  - Leichtere Wartbarkeit durch übersichtlichere Form (z.B. keine Brüche zwischen den Komponenten innerhalb einer Quelldatei)
  - Bessere Arbeitsaufteilung möglich: Ein Programmierer muss sich so nicht mit dem Layout/der Darstellung beschäftigen
- Wiederverwendbarkeit von Code- und Darstellungskomponenten
  - Schnellere Entwicklung durch Vermeidung doppelter Arbeit
  - Redundanzvermeidung

Meine Lösung: iBlock – ein Designframework

- Zentrale (ursprüngliche) Komponente: Templating
  - Eine Website besteht in der Regel aus Komponenten, die an vielen Stellen verwendet werden (Layout, Navigation, Rahmen, Tabellen etc.). Diese Komponenten sollten also auch eigenständig bestehen und nicht per Copy & Paste (Redundanz) an den benötigten Stellen eingesetzt werden.



Templating: Felder

- Definition einer Template

layout.iblock

```
<html>
    <head>
        <title>
            <+$ titel $+>Kein Titel<+$ /titel $+>
        </title>
    </head>
    <body>
        <table>
            <tr><td>
                Navigation: <+$ nav $+>Keine<+$ / $+>
            </tr></td>
            <tr><td>
                Inhalt: <+$ inhalt $+>Keiner<+$ / $+>
            </tr></td>
        </table>
    </body>
</html>
```

-> Ergebnis

```
<html>
 <head>
  <title>
   Meine Seite
  </title>
 </head>
 <body>
  <table>
   <tr><td>
    Navigation: [...]
   </tr></td>
   <tr><td>
    Inhalt: Viel toller Text
   </tr></td>
  </table>
 </body>
</html>
```

- Nutzung einer Template

index.html

```
<& iblock src="layout.iblock" &>
    <$ titel $>Meine Seite<$ / $>
    <$ nav $>
        <& iblock src="nav.iblock" / &>
    <$ / $>
    <$ inhalt $>Viel toller Text<$ / $>
<& / &>
```

Templating: Listen

- Definition

liste.iblock

```
<table>
    <tr><th><+$ kopf1 / $+></th><th><+$ kopf2 / $+></th></tr>
    <+@ zeilen @+>
        <tr><td><+$ spalte1 / $+></td><td><+$ spalte2 / $+></td></tr>
    <+@ / @+>
</table>
```

- Nutzung

liste.html

```
<& iblock src="liste.iblock" kopf1="Name" kopf2="Telefonnummer" &>
    <@ zeilen @>
        <$ spalte1 $>foo<$ / $><$ spalte2 $>555-123456<$ / $>
        <$ spalte1 $>bar<$ / $><$ spalte2 $>555-654321<$ / $>
    <@ / @>
<& / &>
```

- Ergebnis

-> Ergebnis

```
<table>
    <tr><th>Name</th><th>Telefonnummer</th></tr>
    <tr><td>foo</td><td>555-123456</td></tr>
    <tr><td>bar</td><td>555-654321</td></tr>
</table>
```

Plugins

- Das Templating ist nur ein Plugin namens „iblock“:
  `<& iblock src=“...“ / &>`
- Alle Tags der Form `<& ... &>` starten im Hintergrund ein Plugin, das bestimmt wodurch das Tag und sein Inhalt ersetzt werden.
- Weitere Plugins:
  - Upcase: Text in Großbuchstaben umwandeln
  - Embperl: Eingebetteten Perl-Code ausführen

    ```
    <& embperl &>
       print 'hi';
    <& / &>
    ```

  - Guestbook
  - Blog
  - Forum
  - ...
  - (Beispielseiten)

SQL-Plugin

- Das SQL-Plugin generiert iBlock-Felder.
  Aus

  ```
  <& sql query="SELECT name, nummer FROM telefonnummern" / &>
  ```

  wird z.B.:

  ```
  <$ name $>foo<$ / $><$ nummer $>555-123456<$ / $>
  <$ name $>bar<$ / $><$ nummer $>555-654321<$ / $>
  ```

- Wenn man also ein SQL-Tag in eine Liste in einem iBlock-Tag setzt, wird z.B. automatisch eine formatierte Tabelle erzeugt.
- Durch die Möglichkeit der Verschachtelung der Tags ergeben sich ausdrucksstarke und mächtige Konstrukte.

- Warum Perl und nicht $sprache?
  - Perl erlaubt sehr kurze Entwicklungszeiten (gerade bei kleinen bis mittleren Projekten)
    - Der Code beschränkt sich fast nur auf die Semantik, die ganze Schmutzarbeit wird einem abgenommen.
    - Es gibt fast nichts, was es im CPAN nicht gibt.
  - Performance/Integration
    - Mein Framework läuft unter Apache/mod_perl als Apache-Modul, das in Perl geschrieben wurde.
    - So wird der Code nur einmal pro httpd-Prozess kompiliert und kann bei jedem Request ausgeführt werden.
    - Durch mod_perl hat man vollen Zugriff auf den Ablauf des Requests und kann so ohne Umwege über /cgi-bin/ oder eingebettete Tags die Ausgabe kontrollieren.
  - Usw.: Plattformunabhängigkeit, hoher Reifegrad (Perl ist 18 Jahre alt...), OpenSource-Lizenz, Universalität, Schnittstellen, ...

Aber ein großes Projekt in Perl? Das ist doch nicht wartbar!

- Projektgröße (in Zeilen):

```
streawkceur@server:~/iblock$ cat `find -type f` | wc -l
18059
```

- Jedoch aufgeteilt in 67 Module und organisiert durch einen vollständig objektorientierten Ansatz (keine einzige globale Variable).
- Einfaches Refactoring durch klare Schnittstellen und Typenfreiheit.
- Einfache Dokumentation mittels Perldoc
- Bei diszipliniertem Programmieren auch leichte Lesbarkeit des Codes (und z.B. keine Ablenkung durch für die Semantik unwichtige technische Details)

Gibt's sowas denn nicht schon?

- Es gibt viele ähnliche Ansätze (sicherlich habe ich nicht alle gesehen), die jedoch in vielen Aspekten anders funktionieren (und mir nicht 100%ig gefallen)
  - HTML::Embperl (Einbettung von Code in HTML-Seiten. Keine direkte Plugin-Architektur, kein explizites Templating, keine strikte Trennung von Code/Darstellung).
  - HTML::Mason (Ähnlich zu Embperl, jedoch mehr komponentenorientiert. Immernoch keine 100%ige Trennung von Code und Darstellung, kompliziertere Nutzung durch reine Perl-Befehle anstatt verschachtelbaren Tags).
  - AxKit (XML/XSL-basiert mit den verbundenen Vor- und Nachteilen. Nachteile: z.B. dass die Formulierung von XML-Code deutlich aufwendiger zu schreiben und zu parsen ist als mit einer Spezial-Syntax (Namespaces etc), XSLT ist um ein Vielfaches aufwendiger zu formulieren als die einfache iBlock-Template-Sprache, es liegt nicht immer valider XML-Input vor).

www.zentrifuge.biz

z e n t r u m
fuer informatik
und geistige
experimente