Calyxo Web Application Framework

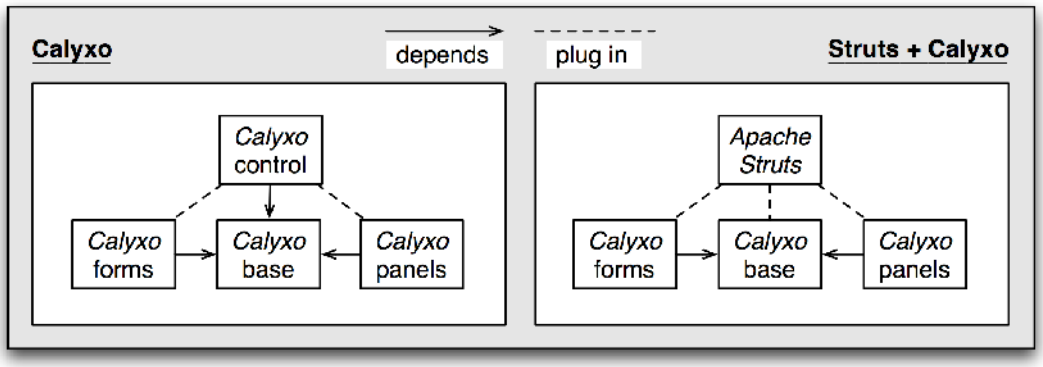
Das Java-Framework ***Calyxo*** ermöglicht die Entwicklung *MVC Model 2*-basierter, modularer und internationalisierter Web-Applikationen. ***Calyxo*** enthält leistungsstarke Komponenten zur Validierung und für das View-Management.

Überblick

Calyxo teilt sich in folgende Komponenten:

- ***Calyxo Base*** – bildet das Fundament von ***Calyxo*** und setzt Basis-Konzepte um
- ***Calyxo Control*** – stellt einen leistungsfähigen Controller zur Verfügung
- ***Calyxo Forms*** – bietet umfangreiche Möglichkeiten zur Form-Validierung
- ***Calyxo Panels*** – ermöglicht ein flexibles View-Management

Calyxo kann wahlweise als Standalone-Framework oder in Verbindung mit *Apache Struts* eingesetzt werden.



Im reinen ***Calyxo*** Szenario übernimmt die ***Calyxo Control*** Komponente die Steuerung.

Im ***Struts+Calyxo*** Szenario übernimmt Struts die Rolle des "führenden" Frameworks, erweitert durch ***Calyxo***. Dies ermöglicht den Einsatz von ***Calyxo*** auch dann, wenn Struts als Standard für die Entwicklung von Web-Applikationen vorgegeben ist. So profitieren Struts-Entwickler vollständig von den ***Calyxo Base***, ***Forms*** und ***Panels*** Komponenten, welche über Plugins in Struts verfügbar gemacht werden.

Projekt

Calyxo ist ein Open Source Projekt der Odysseus Software GmbH (www.odysseus.de) und ist unter der Apache License frei verfügbar. Odysseus bietet Schulungen, Support und Coaching-Dienstleistungen rund um ***Calyxo*** sowie die Unterstützung in Projektleitung, Design und Entwicklung an.

Calyxo erfordert Java 1.4 und die Servlet 2.4 / JSP 2.0 APIs. ***Calyxo***-Applikationen sind somit etwa unter Tomcat 5 oder in jedem J2EE 1.4-fähigen Application Server lauffähig. ***Calyxo*** ist unabhängig von der verwendeten Entwicklungsumgebung und lässt sich leicht in gängige IDEs wie zum Beispiel *Eclipse* integrieren.

Calyxo ist aktuell als Release-Candidate (RC) für die Version 0.9 verfügbar. Die RC-Versionen gelten als stabil und sind für den produktiven Einsatz geeignet. Die Version 0.9 wird voraussichtlich im Herbst 2006 verfügbar sein.

Die umfangreiche Dokumentation ist auf der ***Calyxo*** Site (www.calyxo.org) verfügbar. Dort besteht auch die Möglichkeit zum Download der aktuellen Distribution.

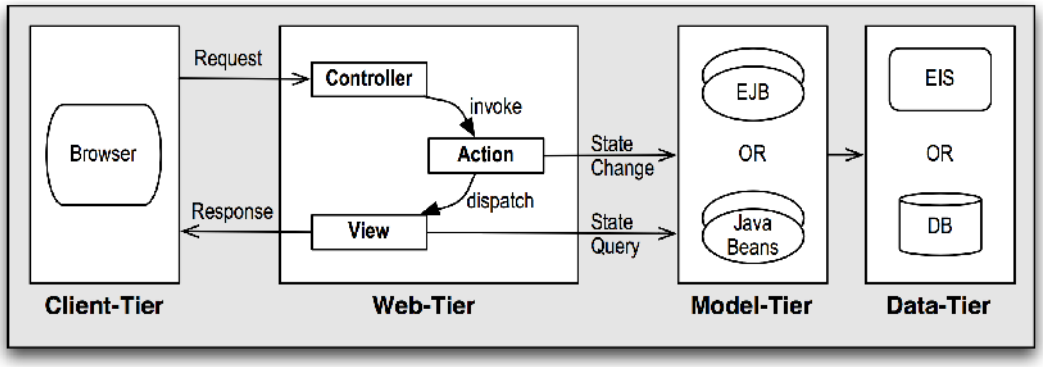
Die Entwickler von ***Calyxo*** sind dankbar für Anregungen oder Feedback (info@calyxo.org).

Einführung

Seit Ende der 90er Jahre wurden diverse auf Servlet-Technologie basierende Frameworks entwickelt. Der wohl bekannteste Vertreter dieser Klasse ist Struts. Durch Struts hat der MVC Model 2 Ansatz weite Verbreitung gefunden.

In einer *MVC Model 2* Anwendung werden sowohl Servlets als auch JSPs eingesetzt:

- Servlets werden vom *Controller* verwendet. Ein eingehender Request wird behandelt, indem ein Stück Application-Code, eine sogenannte Action, selektiert und ausgeführt wird. Actions interagieren lesend und schreibend mit dem *Model*.
- JSPs werden als *View*-Technologie verwendet. Nachdem eine Action ausgeführt wurde, wird ein View selektiert, was letztlich zur Weiterleitung des Requests an eine JSP führt. JSPs greifen nur lesend auf das *Model* zu.



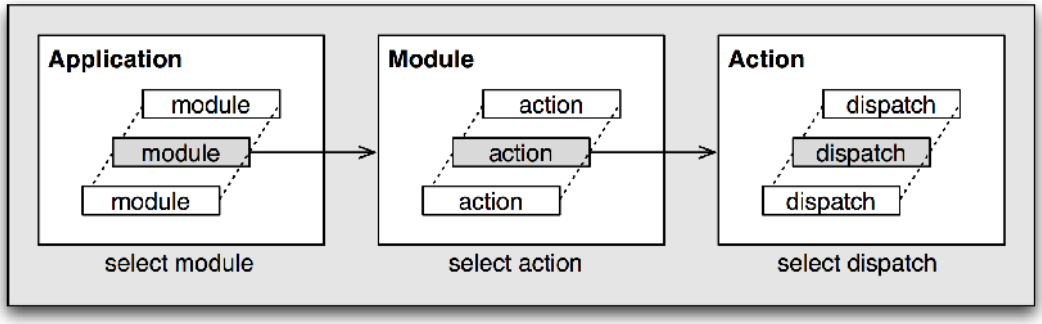
Der MVC Model 2 Ansatz verspricht eine erhöhte Wiederverwendbarkeit, Skalierbarkeit und Wartbarkeit bei gleichzeitig reduzierter Komplexität. In der Praxis ist es jedoch schwer, eine solche Anwendung aus dem Nichts heraus zu entwickeln. An dieser Stelle kommt die Notwendigkeit für den Einsatz eines Frameworks wie ***Calyxo*** ins Spiel.

Die Intention von ***Calyxo*** ist es, die Entwicklung von MVC Model 2 Anwendungen zu erleichtern. ***Calyxo*** übernimmt die Stärken von Struts, behebt einige Schwächen und fügt neue Leistungsmerkmale hinzu. Das Ergebnis ist ein Web-Application-Framework, das nicht nur zu einer beschleunigten und vereinfachten Entwicklung beiträgt, sondern auch zu einer gesteigerten Qualität der resultierenden Anwendungen führt.

Die folgenden Abschnitte geben einen Überblick über die einzelnen ***Calyxo***-Komponenten. Dabei steht die Vermittlung der wesentlichen Ideen und Konzepte im Mittelpunkt. Für weitergehende Details sei auf die ***Calyxo*** Dokumentation verwiesen.

Calyxo Base

Calyxo wurde für den Einsatz mit Controllern entworfen, welche die *Command and Controller* Strategie implementieren. Diese Strategie baut auf die *Front Controller* und *Application Controller* Design Patterns auf. Der *Front Controller*, implementiert als Servlet, dient als zentraler Eingangspunkt für eingehende Requests. Er delegiert an den *Application Controller* (Modul), welcher für die Identifizierung und Ausführung von *Command Objects* (Actions) sowie die Weiterleitung an Views (Dispatching) verantwortlich ist.



Die ***Calyxo Base*** Komponente liefert – ausgehend von dieser Strategie – eine Basis an wiederverwendbaren Funktionalitäten für die übrigen ***Calyxo***-Komponenten, ohne jedoch eine konkrete Controller-Implementierung vorauszusetzen.

Als Konsequenz ist ***Calyxo*** offen für die Kooperation mit unterschiedlichen Controllern. Zur Zeit kann die eigene ***Calyxo Control*** Komponente oder Struts verwendet werden.

Konzepte

Es lohnt sich, einen Blick auf einige der fundamentalen Patterns und Services zu werfen, die in ***Calyxo Base*** gekapselt sind:

- ***Calyxo*** Anwendungen bestehen aus *Modulen*. Module bilden unabhängige Einheiten, etwa wie Sub-Anwendungen. Ein Modul ist ein Container für *Actions*. Modularisierung ist eine bewährte Strategie zur Verminderung der Komplexität und zur Vereinfachung der Entwicklung im Team.
- ***Calyxo*** Komponenten werden über XML-Dateien konfiguriert, die gemeinsame Funktionalitäten haben, wie das Importieren einer anderen XML-Datei, die Definition von Variablen, die Verwendung der JSP Expression Language (inkl. Funktionen) usw.
  Beispiele: `<import file="foo.xml"/>`, `<set var="foo" value="${…}"/>`
- ***Calyxo*** unterstützt Internationalisierung (I18n) von Grund auf. Als Basis steht ein Mechanismus zur Verfügung, um sprachabhängige Resourcen aufzulösen. Andere ***Calyxo*** Komponenten bieten weitere I18n-abhängige Services, wie etwa lokalisierte Views und Formulare.
- ***Calyxo*** propagiert die Verwendung der JSP Expression Language (EL) in Views. ***Calyxo*** unterstützt diesen Ansatz durch sogenannte *Accessors*, die eine Hierarchie von Java Beans und Maps für die Benutzung in JSP EL Ausdrücken zur Verfügung stellen.
  Beispiel: `${calyxo.base.i18n.bundle['strings'].resource['foo']}`

Fazit

Die ***Calyxo Base*** Komponente setzt grundlegende Konzepte um und stellt wesentliche Funktionalitäten bereit, die über die gesamte ***Calyxo*** Plattform wiederverwendet werden.

Hierzu gehören unter anderem die *Command and Controller* Strategie, Modularisierung, XML-Konfigurierung, I18n und die auf der JSP EL basierten *Accessors*.

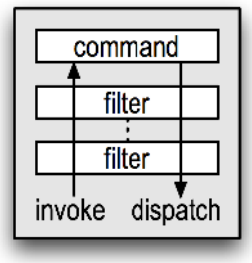
Calyxo Control

Der Controller ist das Herz einer MVC Model 2 Anwendung. Die ***Calyxo Control*** Komponente stellt eine saubere, modulfähige Controller-Implementierung bereit, die die *Front/Application Controller*, *Service To Worker* und *Intercepting Filter* Design Patterns realisiert.

Calyxo Control wurde als ein "Verwandter" zu Struts entworfen, behebt jedoch einige Design-Schwächen und fügt neue Leistungsmerkmale hinzu. Wesentliche Unterschiede zu Struts sind:

- ***Calyxo Control*** unterstützt *Action Filter*. Filter können einer Action in der Konfiguration zugeordnet werden. Action Filter realisieren das *Intercepting Filter* Design Pattern.
- ***Calyxo Control*** vermeidet die Beimischung von Aspekten der Validierung in der Action-Konfigurierung. Stattdessen wird die Validierung durch einen von der ***Calyxo Forms*** Komponente bereitgestellten Filter integriert.
- ***Calyxo Control*** Actions implementieren ein Action-Interface, anstatt von einer Action-Klasse abgeleitet sein zu müssen.
- ***Calyxo Control*** instanziiert ein Action-Command pro konfigurierter Action, nicht als Singleton. So können diese ihre Konfiguration bereits während der Initialisierung analysieren und müssen dies nicht bei jedem Request neu tun.
- ***Calyxo Control*** unterstützt das *Dispatcher* Konzept. Ein Dispatcher ist verantwortlich dafür, an eine andere Action oder einen View zu delegieren. Zum Beispiel wird die ***Calyxo Panels*** Komponente über einen Dispatcher integriert.
- ***Calyxo Control*** verwendet ein eigenes Servlet für jedes Modul. Auf diese Weise wird die Selektion des Moduls dem Servlet-Container überlassen.

Jedes Modul verfügt über einen eigenen Controller, der die Actions des Moduls verwaltet. Während der Initialisierung werden für jede Action die Filter und das Action-Command instanziiert und zu einer ausführbaren Kette arrangiert. Für einen eingehenden Request selektiert der Controller eine Action und führt sie aus. Das Resultat einer Action ist eine Dispatch-Konfiguration, die dann vom Controller an einen Dispatcher weitergereicht wird.



Beispiel

Die folgende Action-Konfiguration benutzt den Filter `forms` (zur Validierung) und definiert zwei Dispatch-Konfigurationen:

```
<action path="/login" class="org.foo.bar.LoginAction">
  <filter name="forms">
    <param name="form" value="login"/>
    <!-- dispatch to redisplay login page -->
    <dispatch path="/WEB-INF/jsp/login.jsp"/>
  </filter>
  <!-- dispatch to action "/welcome" in module "inside" -->
  <dispatch name="success" module="inside" action="/welcome"/>
</action>
```

Fazit

Die ***Calyxo Control*** Komponente weist gegenüber Struts einige Verbesserungen auf, wobei die Unterstützung von Filtern und Dispatchern besonders hervorzuheben ist. Auf Grund der vielen Analogien zu Struts sollten sich Entwickler mit Struts-Erfahrung praktisch ohne Aufwand mit ***Calyxo Control*** zurechtfinden können.

Calyxo Forms

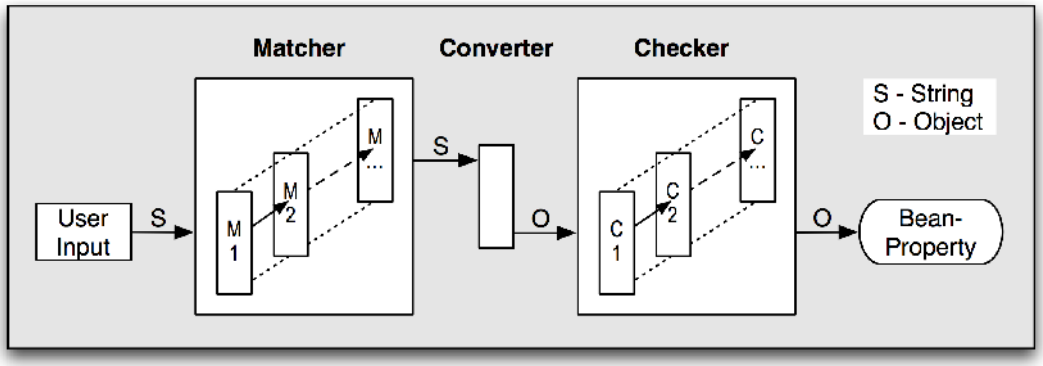
Die Validierung und Präsentation von Benutzereingaben ist essentiell für ein gelungenes Web-Interface. Diese Aufgaben erledigt die ***Calyxo Forms*** Komponente flexibel und zuverlässig. Sie stellt die Bausteine zur Umwandlung von Eingabedaten in geeignete Datentypen (und zurück) zur Verfügung, ergänzt durch vielfältige Validierungsmöglichkeiten. Im einzelnen bietet sie:

- flexible Validierungsmechanismen (für Felder und feldübergreifende Bedingungen)
- prägnante Fehlermeldungen, optisches Hervorheben der betreffenden Eingabefelder
- Umwandlung von Eingaben in Objekte, die zur Weiterverarbeitung zur Verfügung stehen
- Speicherung validierter Eingabewerte zur späteren Wiederanzeige
- Formatierung vorgegebener bzw. gespeicherter Formulardaten durch JSP Tags
- durchgängige I18n-Unterstützung (bezüglich aller zuvor genannter Punkte)
- umfangreiche Konfigurierungsmöglichkeiten über XML-Dateien
- Abdeckung vieler Standardvalidierungen, vielfältige Erweiterungsmechanismen
- Verwendung in Verbindung mit dem ***Calyxo***-Controller oder als Struts-Plugin

Konzepte

Das Kernstück der ***Calyxo Forms*** Komponente bildet die Validierung der Eingabedaten. Diese geschieht in zwei Phasen: Einzelfeld-Validierung und feldübergreifende Validierung.

Die Validierung eines einzelnen Eingabefeldes erfolgt durch eine Folge von *Validatoren*. Jeder Validator erhält den aktuellen Eingabewert und weist ihn entweder zurück (d. h. die Validierung des Feldes ist gescheitert) oder reicht ihn (unverändert oder nach Weiterverarbeitung) weiter. Die Validierung ist in die Abschnitte *Matchen*, *Konvertieren* und *Checken* unterteilt. Jeder Abschnitt enthält Validatoren eines spezifischen Typs.



1. *Matchen*: Der erste Abschnitt der Validatoren-Folge besteht aus einer Folge von Matchern. Ein Matcher prüft seinen Eingabe-String und akzeptiert oder verwirft ihn. Typische Beispiele sind der Trim-Matcher und der RegExp-Matcher.
2. *Konvertieren*: Der zweite Abschnitt besteht aus einem einzelnen Converter. Ein Converter versucht, seinen Eingabe-String zu parsen und in den von ihm unterstützten Datentyp umzuwandeln. Typische Beispiele sind der Date-Converter und der Long-Converter.
3. *Checken*: Der dritte Abschnitt besteht aus einer Folge von Checkern. Ein Checker prüft sein Eingabe-Objekt und akzeptiert oder verwirft es. Typische Beispiele sind der Length-Checker und der Range-Checker.

Neben den zahlreichen Standard-Validatoren können auch selbst definierte Matcher, Converter und Checker verwendet werden.

Wenn alle Eingabedaten die Einzelfeld-Validierung erfolgreich passiert haben und für jedes Eingabefeld ein Ergebniswert vorliegt, können die Eingaben mit Hilfe von *Assertions* weiteren Prüfungen unterzogen werden. Eine Assertion ist eine Bedingung, die als JSP-EL Expression beschrieben wird. Sie kann über Variablen auf alle Eingabedaten (in Rohform oder in Form konvertierter Ergebniswerte) zugreifen. Neben allen gängigen Operatoren und Funktionen besteht die Möglichkeit, selbst definierte Funktionen einzubinden, so dass beliebig komplexe Validierungs-Bedingungen formuliert werden können.

Beispiel

Das folgende Konfigurierungs-Beispiel für ein Formular mit zwei Eingaben illustriert die Verwendung des Long-Converters, des Greater-Checkers sowie einer Assertion, die gewährleistet, dass der erste Wert kleiner ist als der zweite (`lt` steht für "less than").

```
<form name="ltForm">

  <field property="value1">
    <convert name="long">
      <message><arg name="field" value="von"/></message>
    </convert>
    <check name="greater">
      <property name="min" value="0"/>
      <message><arg name="field" value="von"/></message>
    </check>
  </field>

  <field property="value2">
    <convert name="long">
      <message><arg name="field" value="bis"/></message>
    </convert>
  </field>

  <assert test="property.value1 lt property.value2">
    <message bundle="messages" key="error.lt">
      <arg value="von"/>
      <arg value="bis"/>
    </message>
  </assert>

</form>
```

Die Validierung der Eingaben kann nun an drei Stellen scheitern:

1. Wenn im Eingabefeld `value1` keine positive ganze Zahl eingegeben wird, erscheint die Standard-Fehlermeldung des Long-Converters bzw. Greater-Checkers mit der Feldbeschreibung `"von"` (z. B.: "Das Feld 'von' muss eine ganze Zahl enthalten." oder "Das Feld 'von' muss größer als 0 sein.").
2. Wenn im Eingabefeld `value2` keine ganze Zahl eingegeben wird, erscheint die Standard-Fehlermeldung des Long-Converters mit der Feldbeschreibung `"bis"` (z. B.: "Das Feld 'bis' muss eine ganze Zahl enthalten.").
3. Wenn die in `value1` eingegebene Zahl nicht kleiner ist als die in `value2`, erscheint die unter dem Schlüssel `error.lt` eingetragene Fehlermeldung (z. B.: "Die Eingabe 'von' muss kleiner sein als die Eingabe 'bis'.").

Fazit

Die ***Calyxo Forms*** Komponente ermöglicht einen komfortablen Umgang mit formularbasierten Eingaben. Sie erledigt das Validieren, Konvertieren und Formatieren von Eingabedaten. Auf diese Weise befreit ***Calyxo Forms*** die Entwickler auch von zeitaufwändigen Routine-Aufgaben.

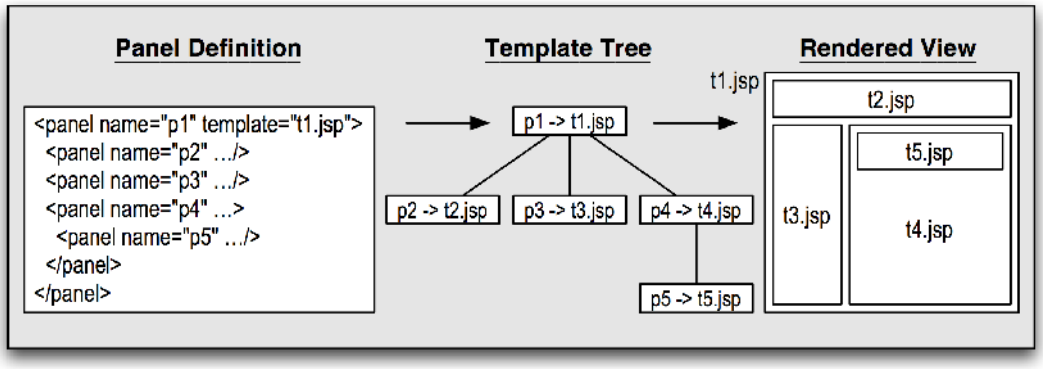
Calyxo Panels

Die Entwicklung und Wartung dynamischer Views ist eine Herausforderung, da oftmals große Gemeinsamkeiten in Inhalten und Layout verschiedener Seiten bestehen. Das einfache Duplizieren von Code macht Anwendungen jedoch schwer wartbar und erweiterbar.

Die ***Calyxo Panels*** Komponente ermöglicht die Komposition von Views nach dem Container/Component-Prinzip. Die Seiten werden dynamisch aus einer Hierarchie einzelner Fragmente zusammengesetzt. Die Definitionen der Views werden in XML beschrieben.

Konzepte

Viele Aspekte der Präsentationsschicht einer Web-Anwendung werden von allen oder zumindest mehreren Views geteilt. Zum Beispiel können Seiten in Bereiche für den Seitenkopf, das Menü und den Inhalt unterteilt sein. Für ein hohes Maß an Wartbarkeit und Konsistenz ist also die Wiederverwendung von View-Komponenten ein zentraler Punkt.



Wir zerteilen die Views in wiederverwendbare Fragmente, sogenannte *Templates*, die zur Laufzeit zu Seiten zusammengesetzt werden. Ein Template kann andere Templates einfügen (und von anderen Templates eingefügt werden).

Um Wiederverwendbarkeit zu gewährleisten, referenziert ein Template die einzufügenden Templates über symbolische Namen. Die jeweiligen aktuellen Pfade der Templates werden anhand der hinterlegten View-Definitionen aufgelöst. In ***Calyxo***-Terminologie heißen diese Definitionen *Panels*.

- Panels können beliebig tief geschachtelt werden. Die Kinder eines Panels heißen *Subpanels*. Die auf oberster Ebene definierten Panels heißen *Toplevel Panels*.
- Ein Panel hat einen Namen. Da dies der symbolische Name ist, der von einem Template beim Einfügen verwendet wird, muss der Name eines Panels eindeutig unter seinen Geschwistern sein.
- Ein Panel kann von einem Toplevel Panel abgeleitet sein. Das heißt, Panel-Definitionen sind selbst wiederverwendbar durch Vererbung.
- Ein Panel kann Parameter an sein assoziiertes Template übergeben. Die Parameter sind im assoziierten Template über einen ***Calyxo*** Accessor verfügbar.
- Fortgeschritte Möglichkeiten: Panels können lokalisiert sein (I18n); Panels können Listen enthalten; Templates können Panel-Parameter beim Einfügen überschreiben.

Für die Praxis sind Toplevel Panels von besonderer Bedeutung. In einer ***Calyxo Panels*** Umgebung können deren Namen als Resource-Pfade verwendet werden, genau wie Pfade zu "echten" JSP-Seiten. ***Calyxo Panels*** baut automatisch den kompletten View aus den assoziierten Templates auf.

Der Einsatz von Panels hat somit keinerlei Auswirkungen auf andere Teile der Anwendung.

Das Hinzufügen eines neuen Views erfordert lediglich die Definition eines Panels und die Erstellung eines Templates, das die für diesen View einmaligen Inhalte enthält. Durch die Verwendung von Vererbung bleibt die Konfigurierung übersichtlich und wartbar.

Beispiel

Das folgende Beispiel illustriert das Zusammenspiel von Panels, Templates und Actions.

```
<panel name="/base.page" template="/WEB-INF/layout/page.jsp">
  <param name="title"/>
  <panel name="header" template="/WEB-INF/layout/header.jsp"/>
  <panel name="menu" template="/WEB-INF/layout/menu.jsp"/>
  <panel name="content"/>
</panel>

<panel name="/foo.page" super="/base.page">
  <param name="title" value="Foo Page"/>
  <panel name="content" template="/WEB-INF/content/foo.jsp"/>
</panel>
```

- Der Panel `/base.page` definiert die Subpanel `header`, `menu` und `content`. Die `template` Attribute bezeichnen die jeweils assoziierten Templates.
- Der `content` Subpanel von `/base.page` definiert kein Template, ist also *abstrakt*. Er fungiert als Basis-Panel, der von anderen Panels erweitert werden kann. Weiterhin fehlt dem Parameter `title` das `value` Attribut.
- Der Panel `/foo.page` ist von `/base.page` durch Angabe des `super` Attributs abgeleitet. Er assoziiert ein Template mit dem `content` Subpanel und gibt einen Wert für den `title` Parameter an. Damit ist er *konkret*.

Das Template `/Web-INF/layout/page.jsp` fügt nun Templates mit dem `<panel>` Tag ein und greift mit dem `panels.param` Accessor auf Parameter zu:

```
<jsp:root version="2.0" xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:panels="http://calyxo.odysseus.de/jsp/panels">
  ... ${calyxo.panels.param['title']} ...
  <panels:panel name="header"/>
  ...
  <panels:panel name="menu"/>
  ...
  <panels:panel name="content"/>
  ...
</jsp:root>
```

Wird nun etwa der Struts-Controller verwendet, so kann die Seite "foo" einfach via

```
<action path="/foo" class="...">
  <forward name="foo" path="/foo.page"/>
</action>
```

angezeigt werden.

Fazit

Als wesentliches Plus bringt ***Calyxo Panels*** die Vorzüge der objekt-orientierten Programmierung in die Präsentationsschicht. Da Templates wiederverwendbare Einheiten bilden, lässt sich das Duplizieren von Code vermeiden, was zu einem hohen Grad an Konsistenz führt. Die Erhaltung eines einheitlichen Layouts wird erheblich vereinfacht.

Obwohl äußerst flexibel, ist die ***Calyxo Panels*** Komponente beindruckend klein. Das komprimierte Design trägt dazu bei, die Komponente einfach verständlich und leicht verwendbar zu machen.