

# Java™magazin

Internet & Enterprise Technology

www.javamagazin.de



**Auf Magazin-CD:**  
→ Fujitsu Siemens BeanConnect  
→ NetBeans 4.1 IDE

→ DynamIT DB-Browser, DBOject Suite  
& EsprIT Server Suite  
→ enough Software J2ME Polish 1.2.4



# Security Architect

## Der Software-Architekt als Security-Experte

# Qualität im Griff

### Projekterfolg durch Agile Quality Management

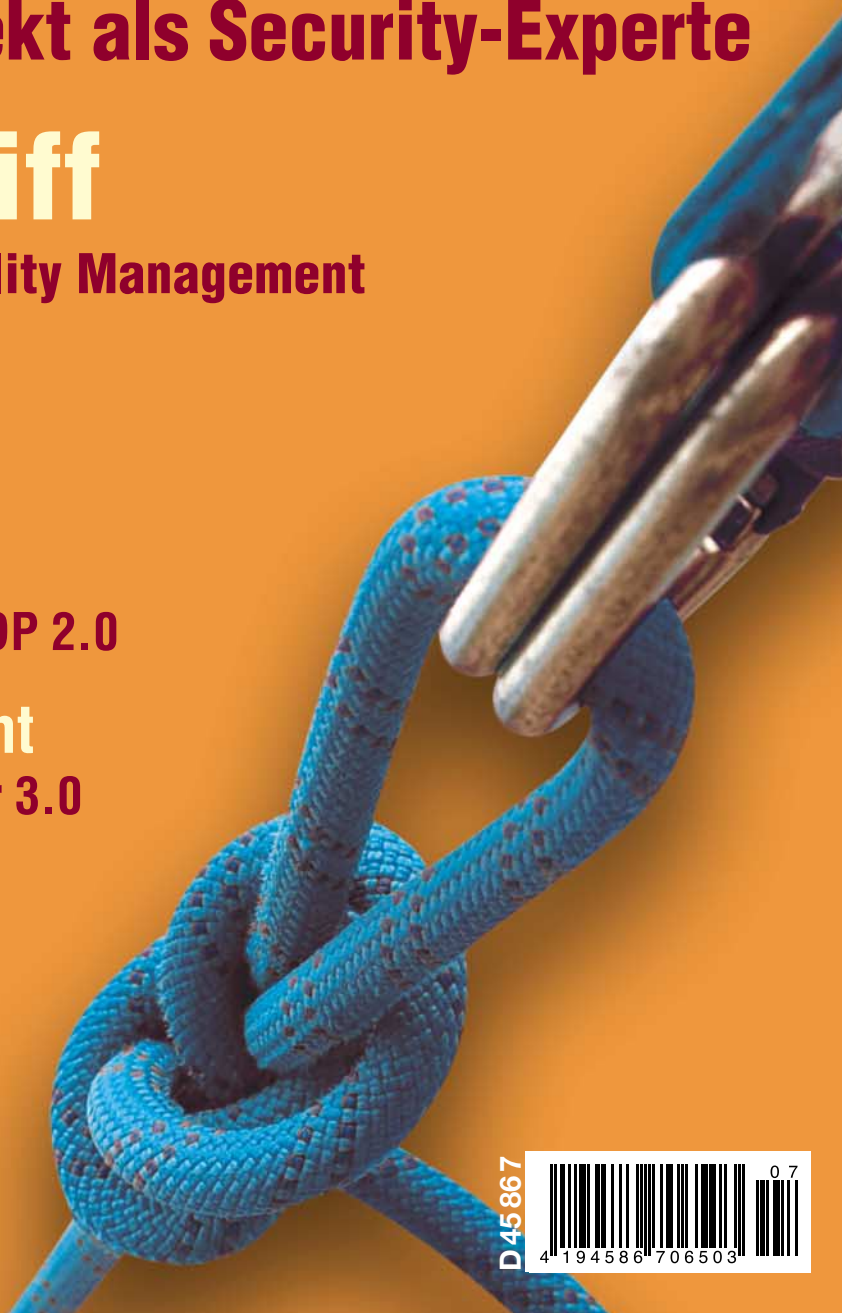
**Architektur**  
Großprojekt PRISMA

**Java Mobile**  
Games-Programmierung mit MIDP 2.0

**Model Driven Development**  
openArchitectureWare Generator 3.0

**Wikis**  
Flexibles Wissensmanagement  
mit SnipSnap

**Code Coverage**  
Ist Ihrem Unit-Test  
etwas entgangen?



## Eine Einführung in das komponentenbasierte Web-Framework wingS

# Beflügelte Weiten

■ VON BENJAMIN SCHMID

Mit Swing zeigte Sun, wie ein effizientes GUI-Komponentenmodell aussehen kann, verharnt aber im Webbereich mit den JavaServer Faces (JSF) doch im seitenorientierten HTML/Java-Mix. Das schon seit längerem existierende open source verfügbare Framework wingS will dagegen zeigen, wie sich Webapplikationen auch vollständig über reine Java-Komponenten entwickeln lassen, und dabei dem designierten Platzhirsch Paroli bieten.

Mit der Einführung der JavaServer Faces kam frischer Wind in den Markt der Web-Frameworks: Die Spezifikation verspricht vollmundig den Schritt zu wieder verwendbaren Komponenten, Events und eine klare Abtrennung der Präsentationsschicht von der Applikationslogik. Bei genauerer Betrachtung kann jedoch JSF seine Ursprünge im etablierten Struts nicht verbergen, verharnt es doch konzeptionell in der Orientierung auf Dialogseiten und Requests. Durch seine tiefe Verankerung mit JSP, XML und HTML impliziert es weiterhin eine fehleranfällige Mischung verschiedener Medien und erschwert damit auch die Umsetzung grundlegender OO-Techniken. Vererbung, Refactoring der Präsentationslogik und testgetriebene Entwicklungsansätze sind mit Hausmitteln erst einmal nicht zu erreichen und so zielte Sun von Anfang an für JSF auf lindernde Tool-Unterstützung.

Dieser Artikel stellt mit wingS ein alternatives Framework fernab des Mainstreams vor, das versucht, eine Brücke zwischen den verschiedenen Welten von Java und World Wide Web zu schlagen.

## wingS is net generation Swing

Wer schon mal mit Swing GUIs entwickelt hat, der wird sich beim Web-Framework wingS schnell heimisch fühlen: Das API orientiert sich in weiten Teilen am großen Bruder für den Desktop und so findet der

Entwickler mit Klassen wie *SButton*, *SPanel*, *STextField* etc. viele vertraute Namen und Methoden wieder. Wie auch in Swing lassen sich diese visuellen Komponenten beliebig erweitern und über Containerklassen zu kompletten Dialogen kombinieren. Für einen ersten Eindruck zeigt Listing 1 ein vollständiges Grundgerüst einer einfachen wingS-Webapplikation. Die aufgeführten Beispiele wurden mit wingS in der Version 1.0.5 realisiert und getestet.

Um nun das aufgeführte Beispiel zum Leben zu erwecken, bedarf es in der *web.xml* des Servlet-Containers (z.B. Tomcat) nur noch der einmaligen Deklaration des *org.wings.session.WingServlet* als zentrales Servlet und einiger Parameter wie dem Namen der Einstiegsklasse für unsere wingS-Applikation. Abbildung 1 zeigt das Ergebnis und belegt damit, wie sich Webapplikationen heutzutage auch ohne kryptische JSPs und XML-Definitionsdateien realisieren lassen. Natürlich möchten wir das Erscheinungsbild noch den eigenen Bedürfnissen anpassen, aber zunächst soll es darum gehen, etwas Logik ins Spiel zu bringen.

## Aktionismus

Zur Ankopplung der Logik verwendet wingS ein in weiten Teilen von Swing adaptiertes Event-Konzept. Die Oberflächenkomponenten bieten Methoden zum Registrieren von Listener-Objekten für be-

stimmte Ereignisse an. Tritt das Ereignis ein, so erzeugen die Komponenten ein entsprechendes Event-Objekt und übergeben dies an die registrierten Listener-Objekte zur Verarbeitung. In unserem Beispiel lässt sich unter anderem mit einem *ActionListener* auf das Klicken des SPEICHERN-Buttons reagieren. Die Implementierung des Listener kann dann die aktuellen Werte der Eingabekomponenten abfragen und zur Weiterverarbeitung an das Backend übergeben:

```
save.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String n = name.getText();
        String t = (String) typ.getSelectedItemAt();
        // Weiterverarbeitung ...
    }
});
```

Neben diesen Eingabeereignissen feuert wingS auch verschiedene Events für die verschiedenen Phasen eines Requests, z.B. bei erkannten ZURÜCK-Navigtionen über den Browser, oder aktiviert einen DEFAULT

## Steckbrief

**wingS 1.0.5****Lizenz:** LGPL License**Download URL:** [sourceforge.net/project/showfiles.php?group\\_id=66895](http://sourceforge.net/project/showfiles.php?group_id=66895)  
[j-wings.org](http://j-wings.org)

Button, falls der aktuelle Request nicht durch einen Mausklick, sondern durch Druck der Eingabetaste ausgelöst wurde. Doch werfen wir einen kurzen Blick hinter die Kulissen, um die zugrunde liegenden Mechanismen näher zu beleuchten.

## Architektur

Oberflächen werden in wingS durch das Erstellen eines hierarchischen Objektnetzes erzeugt, wofür ein erweiterbarer Satz an visuellen Komponenten bereitsteht. Neben einfachen Eingabeelementen, Labels und Containern gehören dazu auch komplexere Komponenten wie Tabellen, Bäume und ToolTips. Wie beim Vorbild Swing erben sie alle von einer gemeinsamen Überklasse *SComponent* und stellen so genannte MVC Delegates dar. Controller und View bilden hierbei eine eng gekoppelte Einheit, während der Zustand einer Komponente als Model gehalten wird.

Für die Darstellung der Komponenten sind spezifische Renderer-Klassen verantwortlich, die als Sammlung austauschbare Pluggable Look & Feels (PLAF) bilden. Diese Renderer arbeiten die gesetzten visuellen und funktionalen Eigenschaften der Komponenten in ihren generierten HTML- und CSS-Code ein. Für aktive Komponenten wie klickbare Elemente, Eingabefelder und Auswahlfelder enthält dieser generierte Code neben der eindeutigen ID zur Identifikation der zugehörigen Komponente auch noch einen Epochen-

Abb. 1: Das EditDemo in Aktion

wert, auf den später noch eingegangen wird. Die Gesamtdarstellung der aktuellen Sicht entsteht somit also über das Durchlaufen des Komponentenbaums der aktuellen Session ausgehend vom obersten Element.

Interaktionen mit dieser generierten Sicht führen zu Requests des Browsers, welche die IDs der manipulierbaren Komponenten zusammen mit jeweiligen Zustandswerten (z.B. der aktuelle Text eines Textfeldes) enthalten. Diese Wertepaare werden vom zentralen wingS Servlet als Low-Level-Events aufgefasst und an die entsprechenden Komponenten der zugehörigen Session weitergereicht. Dort führen diese Low-Level-Events gegebenenfalls zu Zustandsänderungen in deren Modell und zur Auslösung entsprechender Events.

## Layouting

Wenn es um die visuelle Anordnung der Komponenten innerhalb einer Container-

# Anzeige

### Listing 1

```
import org.wings.*;

public class EditDemo {
    public EditDemo() {
        SFrame frame = new SFrame();
        SGridLayout layout = new SGridLayout(2);
        layout.setCellPadding(5);
        SForm panel = new SForm(layout);

        panel.add(new SLabel("Name:"));
        STextField name = new STextField("Max Meister");
        panel.add(name);

        panel.add(new SLabel("Kundentyp:"));
        SComboBox typ = new SComboBox(
            new Object[]{"Stammkunde", "Neukunde"});
        panel.add(typ);

        panel.add(new SLabel("Kommentar:"));

        STextArea kom = new STextArea(
            "Offen gegenüber Vorschlägen");
        panel.add(kom);

        panel.add(new SLabel("Kundengruppe:"));
        SList gruppenList = new SList(new Object[]{
            "Club-Mitglieder", "Premium-Kunden",
            "Professionals", "Tester"});
        gruppenList.setVisibleRowCount(3);
        panel.add(gruppenList);

        SButton save = new SButton("Speichern");
        panel.add(save);

        frame.getContentPane().add(panel);
        frame.show();
    }
}
```

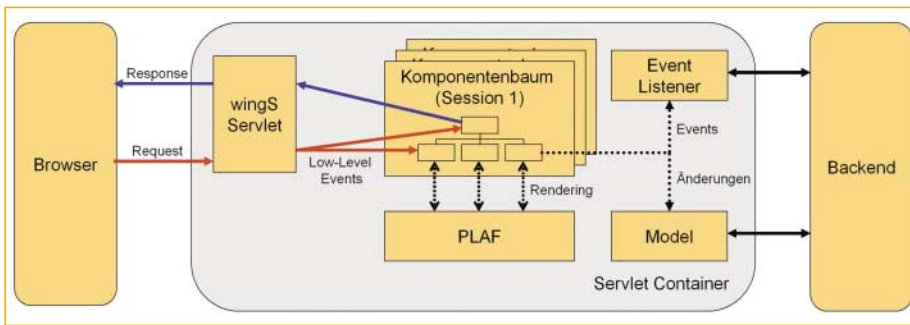


Abb. 2: Grobe Struktur einer wingS-Applikation

Komponente geht, dann kommen die *LayoutManager* zum Zug. Die Palette der auch aus der Swing-Welt bekannten dynamischen *LayoutManager* reicht vom dargestellten, tabellarischen *SGridLayout* über das nautisch angehauchte *SBorderLayout* bis hin zum *SGridBagLayout*, das auch komplexere, programmatische Arrangements über die bekannten *GridBagConstraints* erlaubt. Einige dieser *LayoutManager* benötigen also nähere Informationen über die gewünschte Position der einzelnen Komponenten und erwarten daher, dass diese beim Hinzufügen der Komponenten zu den Containerkomponenten als *Layout Constraints* mit übergeben werden.

Eine Sonderrolle nimmt hierbei das in der Swing-Welt unbekanntere *STemplateLayout* ein. Mit diesem Layouttyp lassen sich statische HTML-Fragmente als Layoutvorlage für einen Container verwenden. Dies bietet sich insbesondere für die grobe Seitenstruktur einer Applikation an, um innerhalb dieser dann beispielsweise den Arbeitsbereich, die Menüleiste und weitere komplexe Komponenten zu positionieren. So lässt sich das Layout einer wingS-Applikation schnell an das gefor-

dierte Corporate Design anpassen, während man für die tiefer geschachtelten Komponenten weiterhin auf die Vorteile der dynamischen Layouts zurückgreifen kann.

Um nun eine Vorlage für das *STemplateLayout* vorzubereiten, müssen in ihr Platzhalter für die einzubindenden Komponenten definiert werden. Dies geschieht über das reservierte `<object>` Tag:

```
<table>
<tr><td colspan="2">
  <object name="headline"></object>
</td></tr><tr>
  <td><object name="menu"></object></td>
  <td><object name="workarea"></object></td>
</tr>
</table>
```

Der Parameter *name* deklariert einen Namen für die entsprechende Stelle in der Vorlage. Um diese Vorlage nun zu verwenden, wird sie im Konstruktor des *STemplateLayout* als *File Handle* oder *URL* übergeben. Innerhalb des Containers platziert man dann die Komponenten, indem man als *Layout Constraints* die definierten Platzhalternamen übergibt:

```
STemplateLayout layout
= new STemplateLayout("main.html");
SPanel panel = new SPanel(layout);
panel.add(headline, "headline");
panel.add(menu, "menu");
panel.add(workArea, "workarea");
```

Erwähnt sei noch die Fähigkeit des *STemplateLayout*, innerhalb der Vorlagen über zusätzliche Tag-Attribute *JavaBean Properties* der platzierten Komponenten setzen zu können. Ein Feature, das sich speziell für größere Projekte anbietet, in denen das komplette Layout und Styling von spezia-

lisierten Webdesignern übernommen wird. Somit haben diese auch noch die Möglichkeit, umfangreichere Änderungen am visuellen Erscheinungsbild der Applikation durchzuführen, ohne Änderungen am Code vornehmen zu müssen (z.B. Zeichenlänge von Eingabefeldern).

## Styling

Prinzipiell besitzen alle Komponenten einen gemeinsamen Satz an visuellen Eigenschaften, den sie von *SComponent* erben. Hierzu gehören unter anderem Vorder- und Hintergrundfarbe, Rahmen und Abstände zu den umgebenden Komponenten, die horizontale und vertikale Ausrichtung, die bevorzugte Größe der Komponenten und *CSS-Attribute*. Darüber hinausgehend haben einige Komponenten noch weitere spezifische Eigenschaften, die ihre Darstellung beeinflussen. Listing 2 zeigt exemplarisch die Verwendung einiger dieser Formatierungen.

Allerdings ist diese Art des Stylings nicht die einzige Möglichkeit, die wingS bietet: Wie bereits erwähnt, übernehmen die *PLAFs* das Rendering der Komponenten. Sie werten die gesetzten visuellen Eigenschaften aus und erzeugen im Fall des *HTML PLAF* sowohl eine *HTML-Seite* als auch ein dynamisch generiertes *Cascading Stylesheet (CSS)*. Die Darstellung dieser Seite innerhalb des Browsers entsteht durch die kaskadierende Anwendung der Definitionen eines globalen Stylesheets mit denen der dynamisch erzeugten Variante. Möchte man globale Anpassungen vornehmen, so empfiehlt es sich, diese im mitgelieferten wingS Stylesheet vorzunehmen. Das *Web Developer Extension Plug-in [2]* für den *Firefox-Browser* leistet hierbei wertvolle Dienste, erlaubt es schließlich das *Live-Editieren* der *CSS-Definitionen* einer Webapplikation.

## Epochen und andere Features

Einen Knopf, den viele Webentwickler am liebsten gerne komplett abschalten würden, ist die *ZURÜCK-Funktion* der *Browser (History Back)*. Jeder kennt die Situation: Schnell die *E-Mail* aus der Liste mit einem Klick auf den *Mülleimer* daneben gelöscht und dank gestikulierender *Mausnavigation* rasch zurück von der *Bestätigungsseite* auf die Liste, um die nächste *Spam-Nach-*

### Listing 2

```
SPanel panel = new SPanel(new SGridLayout(1));
panel.setBackground(Color.blue);
panel.setPreferredSize(new SDimension
    ("350px", "50px"));
panel.setBorder(new SLineBorder(Color.green));

SLabel label = new SLabel("Ich bin rot und kursiv!");
label.setHorizontalAlignment(SConstants.CENTER);
label.setBackground(Color.yellow);
label.setForeground(Color.red);
label.setFont(new SFont("Verdana, Arial",
    SFont.ITALIC, 12));
panel.add(label);
```

richt virtuell zu entsorgen. Aber halt! Für zustandsorientierte Frameworks wie Struts und JSF kann dieser Fall schnell problematisch werden, denn durch das vom Server unbemerkte Zurücknavigieren können Aktionen ausgelöst werden, die unter Umständen gar nicht mehr möglich oder sinnvoll sind. Hinterlegt man am MÜLLEIMER-Button lediglich eine Indexnummer, so löscht der zweite Klick schnell eine ganz andere E-Mail. Auch Effekte wie mehrfach abgeschickte Bestellungen lassen sich durch den Rückwärtssprung in JSF & Co. leicht provozieren.

Mit dem bereits erwähnten Epochenzähler bietet wingS hier Abhilfe. Die Epoche einer Komponente erhöht sich bei jeder Zustandsänderung einer Komponente. Dadurch lassen sich ungültige Low-Level-Events aus alten Sichten eindeutig identifizieren und einfach ignorieren. Um dies für rein lesende Operationen doch zu erlauben, kann diese Prüfung für einzelne Komponenten deaktiviert und damit die Rückwärtsnavigation selektiv erlaubt werden. Alternativ lässt sich dieser Mechanismus

auch derart konfigurieren, dass Rückwärtsnavigationen innerhalb des Browsers empirisch erkannt und als Event für die programmatische Behandlung innerhalb der Applikation zur Verfügung stehen.

### Ausblick

Im produktiven Einsatz hat wingS wiederholt die Stärken seines Konzepts gezeigt. So erweist sich das vertraute API und die damit einhergehenden Kapselung vieler Eigenarten des Webs hinter einem klaren Objektmodell als Vorteil. Besonders der vollwertig objektorientierte Ansatz stellt bei der Entwicklung von Webapplikationen einen Vorteil gegenüber JSP-basierten Frameworks dar. Dies eröffnet auch die Gelegenheit, eigene Frameworks, wie z.B. in [3] vorgestellt, auf wingS aufzubauen. Da wingS unter der Lesser GNU Public License (LGPL) verfügbar ist, steht auch einem Einsatz in kommerziellen Anwendungen nichts im Wege.

Das nächste verfügbare Release von wingS wird diesen Weg konsequent fortsetzen. Neben der Anpassung der PLAFs an

die erweiterten Möglichkeiten moderner Browser wurde auch erhöhte Aufmerksamkeit auf erweiterte Styling-Möglichkeiten der verschiedenen Komponenten mittels CSS gelegt. Neue Features wie Key Bindings, Komponenten-Pop-up-Menüs und die Option, mittels DWR [4] dynamische Seiten nach dem Vorbild von Gmail zu entwickeln, sind Voraussetzung für eine innovative Zukunft mit wingS. ■

*Benjamin Schmid ist Softwarearchitekt und Coach bei eXXcellent solutions in Ulm. Seine Schwerpunkte liegen im Bereich der objektorientierten Verfahren und Methodiken. Seit 2004 beteiligt er sich aktiv an der Weiterentwicklung von wingS. Kontakt: [B.Schmid@excellent.de](mailto:B.Schmid@excellent.de).*

### ■ Links & Literatur

- [1] wingS is net generation Swing: [www.j-wings.org](http://www.j-wings.org)
- [2] Web Developer Extension: [chrispederick.com/work/firefox/webdeveloper/](http://chrispederick.com/work/firefox/webdeveloper/)
- [3] Tobias Vollmer: Ausleihe per pleXX, in *Java Magazin* 3.2005
- [4] Direct Web Remoting: [dwr.dev.java.net](http://dwr.dev.java.net)
- [5] W4Toolkit: [w4toolkit.de](http://w4toolkit.de)

## Anzeige