

## syngenio in der Fachpresse

Medium: Javamagazin

**Thema: Automatische Auswertung von Zurück- und Abbruch-Buttons**

Autor: Erich Becker

Ausgabe : 3.2005

### **Die Navigationszentrale**

Bei Webanwendungen mit komplexen Geschäftsprozessen gibt es einerseits häufig Verzweigungen in der Navigation in Abhängigkeit von Benutzereingaben, andererseits oft auch die Anforderung, dass der Benutzer über einen Zurück-Button wieder auf die vorhergehende Maske gelangen soll. Normalerweise führen diese Anforderungen zu ausprogrammierten Abfragen in *Action*-Klassen, die schlecht wartbar sind und für die Martin Fowlers Ausdruck „Bad Smell in Code“ zutrifft [2]. Dieser Artikel beschreibt eine in einem Kundenprojekt entwickelte Alternative, die die Anforderungen vereint, ohne diese Nachteile mitzubringen, sich als robust erwiesen hat und ständig weiter gepflegt wird.



### **Kontakt & weitere Informationen:**

syngenio AG  
Ivonne Machnacz  
Andreas-Hermes-Straße 3  
D-53175 Bonn  
Fon +49 (0)2 28-6 20 95-100  
Fax +49 (0)2 28-6 20 95-150  
ivonne.machnacz@syngenio.de  
www.syngenio.de

## Automatische Auswertung von Zurück- und Abbruch-Buttons

# Die Navigationszentrale

■ VON ERICH BECKER

Bei Webanwendungen mit komplexen Geschäftsprozessen gibt es einerseits häufig Verzweigungen in der Navigation in Abhängigkeit von Benutzereingaben, andererseits oft auch die Anforderung, dass der Benutzer über einen Zurück-Button wieder auf die vorhergehende Maske gelangen soll. Normalerweise führen diese Anforderungen zu ausprogrammierten Abfragen in *Action*-Klassen, die schlecht wartbar sind und für die Martin Fowlers Ausdruck „Bad Smell in Code“ zutrifft [2]. Dieser Artikel beschreibt eine in einem Kundenprojekt entwickelte Alternative, die die Anforderungen vereint, ohne diese Nachteile mitzubringen, sich als robust erwiesen hat und ständig weiter gepflegt wird.



Eine Anforderung in vielen Projekten ist, dass die verwendeten Frameworks unverändert bleiben. Die Vorteile liegen auf der Hand: Bei einem Versionswechsel kann

leicht eine Migration durchgeführt werden und die am Projekt beteiligten Entwickler müssen nur ihre Anwendungslogik und die veröffentlichten Schnittstellen des Frameworks kennen, nicht aber dessen Interna. Zudem kann man Fehler besser isolieren, wenn Framework- und Anwendungs-Code streng getrennt werden. Also werden Erweiterungen des Frameworks, in diesem Fall Struts, in neuen Klassen eingebaut. Hier stellt sich die Frage, wie die Erweiterungen der Anwendungslogik zugänglich gemacht werden sollen. Am bequemsten für einen Entwickler ist die Definition einer Basisklasse, von der die Applikationsklassen erben. Dies kann jedoch auch Nachteile haben und ist bei bereits bestehenden Anwendungen meist unpraktikabel. Alternativ schreibt man Hilfsklassen, an die die Anwendungslogik Routineaufgaben delegiert. Bei dem Projekt, aus dem heraus dieser Artikel entstand, haben wir uns dafür entschieden, technische Hilfsfunktionen, die in der gesamten Anwen-

dung gebraucht werden, in eine Basisklasse auszulagern, dagegen fachliche Hilfsfunktionen, die oft nur einzelne Geschäftsprozesse betreffen, in Hilfsklassen zur Verfügung zu stellen.

### Woher kommst du?

Die Navigationslogik einer Struts-Anwendung liegt in der Controller-Schicht, also den *Action*-Klassen. Folglich wird die hier beschriebene Problemlösung in einer Klasse *BaseAction* implementiert. Mit einer Ausnahme, welche unten beschrieben wird, lässt sich die Funktionalität übrigens auch als Hilfsklasse nutzen. Die Navigationsobjekte in Struts sind *ActionForwards*. Um auf die Navigationshistorie zugreifen zu können, benötigt man also einen Speicher für *ActionForwards*. Dieser sollte im Session-Kontext liegen, weil die Navigation an die aktuelle Benutzersitzung gebunden ist. Damit ergibt sich im ersten Schritt der Quelltext aus Listing 1.

### Wohin willst du?

Mit dem bisherigen Code lässt sich bereits die Navigationshistorie befüllen und auswerten. Eine Action muss nur nach der Bestimmung der nächsten Maske *pushNavigationForward()* aufrufen und wenn sie feststellt, dass der Benutzer auf ZURÜCK

geklickt hat, die letzte Maske mit Aufrufen von *getNavigationHistory()* und *getLastForward()* bestimmen. Dies ist in bereits vorhandenen Actions schon hilfreich, erfordert aber immer noch eine Menge mehrfachen Codes. Es wäre doch viel praktischer, wenn eine einzelne Methode feststellen würde, welcher Button gedrückt wurde, und sich automatisch um die Ermittlung der nächsten Maske und die Pflege und Auswertung der Navigationshistorie kümmern würde. Woher aber soll eine generische Klasse wissen, welche Buttons es in einer Anwendung gibt? Eine Antwort ist: Wir werten nur Standard-Buttons wie ABBRECHEN, ZURÜCK und WEITER aus. Die andere ist, der Klasse mitzuteilen, welches die Standard-Buttons sind. In *BaseAction* sind beide Möglichkeiten kombiniert (Listing 2).

Mit diesen Standard-Buttons können wir uns eine Methode bauen, die automatisch die Buttons auswertet und die Navigationshistorie pflegt. Hierbei ist allerdings noch etwas zu beachten: Es gibt Masken, bei denen auf unterschiedliche Buttons sehr unterschiedlich reagiert werden muss, in den meisten Fällen genügt die Information, ob die Aktion erfolgreich war, für eine Entscheidung aus. Durch die zusätzlichen Parameter *failed* und *generic* werden diese

Fälle mit abgedeckt. Die Bedeutung von *failed* ist klar, wenn *generic=true* sowie *failed=false* ist, wird einfach das *success-Forward* ermittelt, ist *generic=false*, wird ein Button-spezifisches Forward gesucht (Listing 3).

## Und wie koche ich das?

Damit haben wir alle Zutaten zusammen und das Kochrezept ist dann ganz einfach.

Eine *execute*-Methode einer Action sieht jetzt aus wie im nächsten Listing. Tatsächlich enthält die Klasse *BaseAction* bereits diese Implementierung, die wiederum eine weitere *execute*-Methode mit erweiterter Signatur und reiner Fachlogik aufruft. Hier ist die einzige Stelle, an der das Delegation Pattern nicht greift; Actions, die nicht von *BaseAction* erben, müssen den Code kopieren oder explizit *BaseAction.execute()* auf-

rufen, abgeleitete Klassen können sich auf die Implementierung der erweiterten *execute()*-Methode beschränken (Listing 4).

Eine Action-Klasse kann in ihrer *execute*-Methode durchaus ein *ActionForward* zurückgeben, der von der Logik in *getActionForwardFromButton()* abweicht. Dieses wird dann anstelle des automatisch gefundenen in die Navigationshistorie eingefügt.

### Listing 1

```
package com.syngenio.struts.actions;
...
public class BaseAction extends Action implements
    com.syngenio.struts.actions.
        IButtonHandler {

    public final static String BACK_FORWARD_KEY
        = "backForward";

    final static int MAX_STACK_SIZE = 10;
    final static String actionEnd = ".do";

    /**
     * Diese Methode legt ein ActionForward in der
     * Navigationshistorie ab. Dabei wird sichergestellt,
     * dass jedes ActionForward nur einmal abgelegt wird,
     * und dass keine Actions abgelegt werden. @param
     * request der Request-Kontext, in dem wir uns befinden.
     * @param forward das ActionForward, das gespeichert
     * werden soll.
     */
    public void pushNavigationForward( HttpServletRequest
        request, ActionForward forward ) {
        ActionForward last = null;
        Stack history = getNavigationHistory( request );
        if ( history.size() > 0 ) {
            last = (ActionForward) history.peek();
        }
        if ( forward != null ) {
            if ( !forward.getPath().endsWith( actionEnd ) ) { // don't
                push Actions!!!
            }
            if ( last == null || !last.getPath().equals( forward.
                getPath() ) ) {
                history.push( forward );
                while ( history.size() > MAX_STACK_SIZE ) {
                    history.remove( 0 );
                }
            }
        }
    }

    /**
     * Diese Methode liefert die aktuelle Navigationshistorie
     * zurück. @param request der Request-Kontext, in dem
     * wir uns befinden. @return Die Navigationshistorie als
     * Stack von ActionForwards. Wenn noch keine
     * angelegt wurde, wird null zurückgegeben.
     */
    protected Stack getNavigationHistory( HttpServletRequest
        request ) {
        Stack retVal = (Stack) request.getSession().
            getAttribute( BACK_FORWARD_KEY );
        if ( retVal == null ) {
            retVal = new Stack();
            putIntoSessionContext( request, retVal, BACK_
                FORWARD_KEY );
        }
        return retVal;
    }

    /**
     * Diese Methode wertet den Navigations-Stack aus und
     * liefert die letzte Maske zurück. Diese <em>letzte</em>
     * Maske ist das zweitoberste Objekt im Stack, das
     * oberste Objekt ist die <em>aktuelle</em> Maske.
     * Dieses oberste Objekt wird ebenfalls entfernt.
     * @param forwardStack ein Stack von ActionForwards
     * @return ein ActionForward, falls eines gefunden wurde,
     * sonst null.
     */
    protected ActionForward getLastForward( Stack
        forwardStack ) {
        ActionForward retVal = null;
        if ( !forwardStack.isEmpty() ) {
            forwardStack.pop();
            if ( !forwardStack.isEmpty() ) {
                retVal = (ActionForward) forwardStack.pop();
            }
        }
        return retVal;
    }
...
}
```

### Listing 2

```
private String[] defaultStandardButtons = {
    BUTTON_CONTINUE, BUTTON_CANCEL, BUTTON_ADD,
        BUTTON_CHANGE,
    BUTTON_CHANGE, BUTTON_CLOSE, BUTTON_CONTINUE,
    BUTTON_DELETE,
    BUTTON_SAVE, BUTTON_SEND, BUTTON_STATUS
};

public static final String PARAM_STD_BUTTONS
    = "standardButtons";

// tatsächlich genutzte standard buttons
private String[] standardButtons = null;

/**
 * Diese Methode lädt die Standard-Buttons der
 * Anwendung über den Request-Kontext.
 * Die Buttons werden in web.xml als Parameter
 * standardButtons dem ActionServlet übergeben.
 * @param request HttpServletRequest
 */
private void loadStandardButtons( HttpServletRequest
    request ) {

    if ( standardButtons == null ) {
        ServletContext context = request.getSession().
            getServletContext();

        String commaSeparatedListOfButtons =
            context.getInitParameter( PARAM_STD_BUTTONS );
        if ( commaSeparatedListOfButtons == null ) {
            standardButtons = defaultStandardButtons;
        }
        else {
            StringTokenizer token = new StringTokenizer
                ( commaSeparatedListOfButtons, ", " );
            Vector buttons = new Vector();
            while ( token.hasMoreTokens() ) {
                String button = token.nextToken();
                buttons.add( button );
            }
            standardButtons = new String[ buttons.size() ];
            buttons.copyInto( standardButtons );
        }
    }
}
```

Noch einen Hinweis zum Schluss: Da-  
mit eine JSP dargestellt werden kann, muss  
in der Regel nicht nur das zugehörige *ActionForward* bekannt sein, sondern auch  
die notwendigen *ActionForms*. Diese soll-  
ten im Session-Kontext liegen, wenn die  
Anwendung auf *BaseAction* aufbaut. Dies  
ist jedoch bei vielen Anwendungen der Fall  
und auch bei Struts als Voreinstellung ge-  
geben.

*Dr. Erich Becker ist Senior Consultant Tech-  
nology bei syngenio, einem IT-Unterneh-  
men, das sich u.a. auf die Beratung und Re-  
alisierung von transaktionalen J2EE-Lö-  
sungen spezialisiert hat. Der Autor beschäf-  
tigt sich seit 1997 mit Webanwendungen,  
seit 2000 mit solchen auf JSP-Basis.*

### ■ Links & Literatur

- [1] Martin Fowler: Refactoring, Addison-Wesley, 2000  
(dt. Ausgabe)
- [2] [struts.apache.org](http://struts.apache.org)
- [3] Sven Haiges (Hrsg.): Struts, Software & Support Verlag,  
2003

### Listing 3

```
public final static String FORWARD_SUCCESS = "success";
public final static String FORWARD_FAILURE = "failure";
public ActionForward getActionForwardFromButton
    ( ActionMapping mapping, HttpServletRequest
      request, boolean failed, boolean generic ) {
    loadStandardButtons();
    ActionForward retVal = null;
    if ( request.getParameter( BUTTON_BACK ) != null ) {
        retVal = mapping.findForward( BUTTON_BACK );
        if ( retVal == null ) {
            retVal = getLastForward( getNavigationHistory
                ( request ) );
        }
        if ( retVal == null ) {
            retVal = mapping.getInputForward();
        }
    }
    else if ( failed ) {
        retVal = mapping.findForward( FORWARD_FAILURE );
        if ( retVal == null ) { // no failure specified
            retVal = ( ActionForward ) getNavigationHistory
                ( request ).peek();
        }
    }
    else {
        for ( int i = 0; i < standardButtons.length; i++ ) {
            String button = standardButtons[i];
            if ( request.getParameter( button ) != null ) {
                retVal = mapping.findForward( button );
                break;
            }
        }
    }

    if ( retVal == null && generic ) {
        retVal = mapping.findForward( FORWARD_SUCCESS );
    }
    pushNavigationForward( request, retVal ); // for use by
        next back button action

    return retVal;
}
```

### Listing 4

```
public final static int FLAG_FAILED = 0;
public final static int FLAG_GENERIC = 1;
public ActionForward execute
    ( ActionMapping mapping, ActionForm actionForm,
      HttpServletRequest request, HttpServletResponse
        response ) throws java.lang.Exception {
    ActionForward retVal = null;
    ActionErrors errors = new ActionErrors();
    boolean failed = true;
    boolean generic = true;
    if ( request.getParameter( com.syngenio.struts.
        actions.IButtonHandler.BUTTON_CANCEL ) != null ||
        request.getParameter( com.syngenio.struts.
            actions.IButtonHandler.BUTTON_BACK ) != null ) {
        failed = false;
    }
    else {
        boolean flags[] = new boolean[] { failed, generic };
        retVal = execute( mapping, actionForm, request,
            response, errors, flags );

        failed = flags[ FLAG_FAILED ];
        generic = flags[ FLAG_GENERIC ];
    }
    if ( !errors.isEmpty() ) {
        resolveFieldKeys( errors, request );
        saveErrors ( request, errors );
    }
    if ( retVal == null ) {
        retVal = getActionForwardFromButton( mapping,
            request, failed, genericForward );
    }
    else {
        pushNavigationForward( request, retVal );
    }
    return retVal;
}
```

Anzeige