

# Java<sup>TM</sup>magazin

Internet & Enterprise Technology

**XML**  
extra  
included

## Eclipse vs. Netbeans

Open Source-Tools: Welche IDE für wen?

## Einführung in JUnit

Grundlagen für automatisiertes Testen

## Struts und Together

Webentwicklung mit UML

## J2EE-Architektur

Grundlagen und Prinzipien

## Groovy

Alternative mit Java-Integration

## UDDI 3.0

Verzeichnisdienst wird erwachsen

## Naked Objects

Objekte pur



mit CD!

D45867



# UML-Unterstützung des Entwicklungsprozesses von Struts-basierten Webanwendungen

von Andreas Lux und Christoph Schmidt-Casdorff

## All Together Now

Struts ist das derzeit wohl am meisten eingesetzte Framework zur Entwicklung von Webanwendungen. Der Großteil der Applikationslogik ist in Struts in der Datei *struts-config.xml* definiert. Diese fachliche Logik soll nun in einem UML-Modell beschrieben werden, um daraus die *struts-config.xml* generieren zu können. Wir stellen vor, mit welchen Sprachmitteln man die Oberflächensteuerung in Struts-basierten Webanwendungen modellieren kann und welche Rahmenbedingungen hierfür gegeben sein müssen. Das Vorgehen und die Beispiele stammen aus einem aktuellen Projekt der Deutsche Post AG Niederlassung Renten-Service.



### Motivation

In vielen Softwareprojekten ist es eine Herausforderung, die Welt der fachlichen Analyse mit der Welt der Softwareentwicklung zusammenzubringen. Dabei ist es sehr wichtig, dass eine „gemeinsame Sprache“ gesprochen wird. Aufgrund ihrer Sprachmittel und ihrer Verbreitung ist unserer Meinung nach die UML hierzu besonders geeignet.

In diesem Artikel wollen wir die Modellierung der Teilbereiche von Webanwendungen betrachten, die sich mit der Oberflächensteuerung beschäftigen. Wir sehen ein großes Problem in der fehlenden Abstraktionsschicht zwischen den fachlichen Anforderungen und den von den

Softwareentwicklern implementierten Benutzeroberflächen (GUI). Dieses gilt es zu beheben, jedoch ohne zusätzliche Redundanzen zu schaffen.

### Warum ein Modell?

Die Entwicklung einer Seitennavigation (Workflow) mittels Struts erstreckt sich im Wesentlichen auf die Editierung der *struts-config*. Dort wird der Workflow über die Beschreibung des Zusammenspiels von *actions* und *forwards* definiert. Derzeit finden sich ausgereifte Werkzeuge zur Modellierung von Struts-Anwendungen [1]. Nach unserer Einschätzung richten sich die Werkzeuge aber im Wesentlichen an Softwareentwickler.

Die Erfahrung zeigt, dass die Modellierung gerade von größeren Systemen in den Details der Konfigurationsdatei untergeht. Es fehlt der abstrakte Überblick auf das gesamte System. In den uns bekannten Projekten ist es in der Regel so, dass die fachlichen Anforderungen an das System von den Fachabteilungen kommen. Anschließend erarbeitet die Softwareentwicklung basierend auf diesen Anforderungen zusammen mit der Fachabteilung einen Realisierungsvorschlag bzgl. des GUI.

Aufgrund der fehlenden Abstraktionsschicht wird oft auf andere Hilfsmittel (z.B. eine prototypische Applikation) zurückgegriffen, um den Fachabteilungen die fachliche Logik zu veranschaulichen.

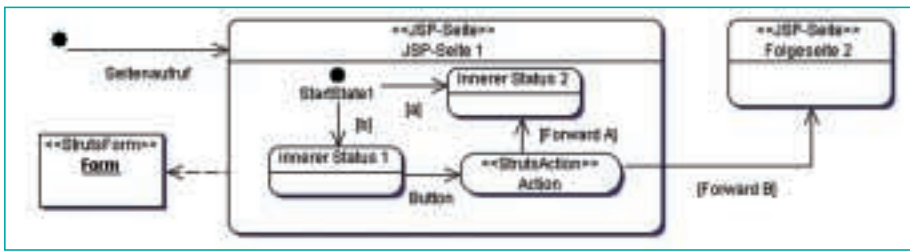


Abb. 1: Beispiel eines Struts-Diagramms

Dieser Prozess zieht oft aufwändige und fehleranfällige Änderungen an der Konfigurationsdatei nach sich. Das von uns erstellte Modell zur Visualisierung fachlicher Abläufe der Seitensteuerung basiert auf der UML. UML-Modelle sind als Grundlage für die Diskussion an der Schnittstelle zwischen fachlicher Analyse und Softwareentwicklung bestens geeignet, da die UML bereits in beiden Welten eingesetzt wird. Es ist wichtig für uns, den Workflow im UML-Modell vorliegen zu haben, da wir die Symbole des Workflows mit weiteren UML-Informationen (Sequenzdiagramme, Klassen ...) in Beziehung bringen wollen. Würde der Workflow außerhalb des Modells beschrieben werden, wäre dies nicht mehr möglich.

Die meisten Workflow-Modelle im HTTP-Umfeld lassen sich als ein endlicher Zustandsautomat abbilden. Es ist nur entscheidend, welche Aktion ausgeführt wird und wie der aktuelle Zustand des Systems ist. Es ist nicht von Bedeutung, auf welchem Weg der Anwender zu diesem aktuellen Systemzustand gelangt ist. Basierend auf dieser Annahme kann der Struts-Workflow auf Basis eines endlichen Automaten in UML State Chart-Diagrammen modelliert werden.

Über die reine Visualisierung des Struts-Workflows bietet ein solches Modell weitere Möglichkeiten: Es können Zustände der JSP in Abhängigkeit vom Datenmodell modelliert werden. Oft ist die Verarbeitungslo-

gik einer Seite stark durch den aktuellen Zustand des zugrunde liegenden Datenmodells beeinflusst. So kann beispielsweise die Auswahl eines Elements aus einer Trefferliste nur erfolgen, wenn die Trefferliste auch tatsächlich Elemente enthält. Ein Modell bietet die Möglichkeit, auf Basis der Modelldaten abhängige Ressourcen wie Konfigurationsdateien oder Code zu generieren. Beispielsweise kann die *struts.config* auf Basis eines Modells erzeugt werden. Nimmt man den Struts-Workflow als Modellbasis, können auf dieser Ebene weitere Informationen an einzelne Elemente dieses Modells gebunden werden. Zum Beispiel können innerhalb des Modells Hilfetexte oder Menüs an JSPs oder Rechte an *actions* geknüpft werden.

### Struts-Einstieg

Die Implementierung einer Benutzeroberfläche mittels Struts basiert auf dem MVC-Pattern. Sie besteht aus Elementen der Präsentationsschicht (View), die als JSP-Seiten umgesetzt werden. Die Anbindung an das Struts-Framework erfolgt in Form von *ActionHandler* und *ActionForms*. *ActionHandler* dienen als Mittler zwischen der Repräsentation der Daten des Geschäftsmodells (Modell) und der Darstellung der Daten in der Präsentationsschicht und stellen somit den Controller dar. Des Weiteren definieren *ActionHandler* die Logik der GUI-Steuerung. *ActionForms* hingegen dienen zur Darstellung der Daten des Geschäftsmodells in den JSP-Seiten. Die Verknüpfung zwischen den *ActionForms* und dem *ActionHandler* wird in der Datei *struts.config* beschrieben und stellt den eigentlichen Workflow dar. Allerdings ist diese Konfigurationsdatei nur für Entwickler lesbar und wird selbst in kleineren Projekten schnell sehr unübersichtlich. Zur Pflege dieser Datei existieren bereits eine Vielzahl guter Werkzeuge (Abb. 3), die aber zwei

entscheidende Nachteile beherbergen: Die Inhalte der *struts.config* sind sehr technisch und somit als Diskussionsgrundlage mit den Fachabteilungen nicht geeignet. Die Darstellung der Konfigurationsdaten in den Tools tragen oft wenig zum fachlichen Verständnis des Workflows bei. Es ist daher notwendig, eine Möglichkeit zu schaffen, die einerseits die Grundlage für die Diskussion der seitenübergreifenden Navigation bietet und andererseits die Komplexität der *struts.config* beherrschbar macht. Dies ist eine stark vereinfachte Darstellung des Einsatzes eines Struts-Frameworks. Es dient an dieser Stelle lediglich dazu, die notwendigen Begrifflichkeiten einzuführen. (Zur Beziehung zwischen dem Struts-Framework und dem MVC-Pattern siehe auch [2].)

### Modellierungsvoraussetzungen in Struts

Um den Struts-Workflow zu modellieren, interessieren wir uns für die Oberflächenelemente, in der Regel also die JSPs, die einzelnen Bearbeitungsschritte der Geschäftslogik (*Struts-Actions*) und die Übergänge zwischen diesen Elementen.

Die Darstellung von Applikationsoberflächen (JSPs, Dialoge ...) kann man als einen Status sehen, in dem sich die Applikation befindet. Die ausgehenden Transitionen einer JSP sind die URLs, die entweder per Link oder *submit* gesendet werden können. Zur Definition der *Struts-Actions* und ihrer ausgehenden Transitionen können in Struts deklarativ einzelne Schritte des Workflows beschrieben werden. Die Konfiguration dieser Beschreibung ist Bestandteil der Struts-Konfiguration (Listing 1). Diese einzelnen Beschreibungen heißen *actions* und werden durch einen eindeutigen Bezeichner identifiziert. *actions* beschreiben, welche Geschäftslogik in einem Arbeitsschritt zu bearbeiten ist. Diese wird durch eine so genannte *action*-Klasse repräsentiert. Die möglichen nächsten Schritte im Workflow nach der Abarbeitung der *action* wird in der Struts-Terminologie als *forward* bezeichnet. Diese Schritte beinhalten einen Namen und das Ziel des nächsten Schrittes. In der Regel verweisen die *forwards* auf eine JSP. Als Ergebnis liefert *action* einen Identifikator, unter dem ein *forward* ermittelt

#### Listing 1

##### Action Mapping der *struts.config.xml*

```
<action path="/banksuchen"
type="web.handler.BankSuchenAction"
name="banksuchenform"
scope="request"
input="/banksuchen.jsp">
<forward name="success" path="/banksuchen.jsp"/>
</action>
```

werden kann. Für die in Listing 1 beschriebenen *actions* werden die Ansteuerung und der Bearbeitungsablauf skizziert. Mithilfe der Setzung des Servlet-Mappings in der *web.xml* und des *ActionServlets* von Struts wird nun einem Request eine *action* zugeordnet. Dieser Vorgang wird als Action Mapping bezeichnet.

Anhand der Servlet Mappings wird erkannt, welcher Request durch das Struts-Framework bearbeitet wird. Dieses extrahiert aus dem Request Path den Bezeichner der *action* und sorgt für deren Ausführung. Die URL *http://localhost/myweapp/banksperren.do* wird dem Struts-Framework übergeben. Dem Request Path */banksperren.do* wird die *action /banksuchen* zugeordnet. Es wird eine Instanz der Klasse *web.handler.BankSuchenAction* ausgeführt. Auf Basis des Ergebnisses aus Schritt 3 wird der *forward* bestimmt. Wird zum Beispiel *success* geliefert, so wird */banksuchen.jsp* angesteuert (Listing 1).

Dabei sind verschiedene Spielarten möglich wie global definierte *forwards*, *actions* ohne Anbindung an die Geschäftslogik oder *actions* mit *forwards*, die wiederum auf andere *actions* verweisen.

## Beschreibung des Modells

Ein UML-Modell muss also in der Lage sein, diese beschriebenen Mechanismen anschaulich und vor allem vollständig abzubilden. So ergibt sich folgende Beschreibung der Abbildung eines Struts-Work-

flows in einem Zustandsautomaten: Der Struts-Workflow wird als endlicher Automat mithilfe von State Charts modelliert. Einer JSP wird ein Zustand zugeordnet. Genauer gesagt, wird der Zustand betrachtet, den eine JSP in dem Moment einnimmt, in dem sie bereit ist, Benutzereingaben zu akzeptieren. Die Events, die als Benutzungsaktion ausgelöst werden können, werden durch die aufgerufene URL definiert. Diese URL verweist auf eine *action* oder eine andere JSP. Durch das oben beschriebene Action Mapping von Struts kann einer Benutzungsaktion direkt eine *action* zugeordnet werden. Wir modellieren also die *actions*, die von einer JSP aus-

gelöst werden können. Es können dabei auch Subzustände der JSP als Reaktion auf unterschiedliche Zustände des Datenmodells modelliert werden. Die Ausführung der *action* ist eine Aktivität. Die „do activity“ dieses Zustandes besteht aus der Ausführung der *execute*-Methode der zugrunde liegenden *action*-Instanz. Diese liefert einen Identifikator, unter dem im *forward-mapping* die Konfiguration des nächsten Schritts zu finden ist. Die Transitionen einer JSP zu einer *action* werden über eine Benutzeraktion (*Event*) ausgelöst. Die Transition von einer *action* zum nächsten Schritt wird über das Resultat der Bearbeitung durch die *action*-Instanz

### Listing2

#### Auszug des Action Mapping-Bereichs der zugehörigen *struts-config.xml*

```
<action path="/start"
type="web.handler.InitBankSuchenAction"
name="banksuchenform"
scope="request"
input="/banksuchen.jsp">
<forward name="success" path="/banksuchen.jsp"/>
</action>
<action path="/banksuchen"
type="web.handler.BankSuchenAction"
name="banksuchenform"
scope="request"
input="/banksuchen.jsp">
<forward name="success" path="/banksuchen.jsp"/>
</action>
<action path="/initbankinfo"
type="web.handler.InitBankInfoAction"
name="bankinfoform"
scope="request"
input="/banksuchen.jsp">
<forward name="success" path=
"/sperrkennzeichenveraendern.jsp"/>
</action>
<action path="/sperrkennzeichenveraendern"
type="web.handler.SperrkennzeichenVeraendernAction"
name="bankinfoform"
scope="request"
input="/sperrkennzeichenveraendern.jsp">
<forward name="success" path="/banksuchen.jsp"/>
</action>
```

## Anzeige

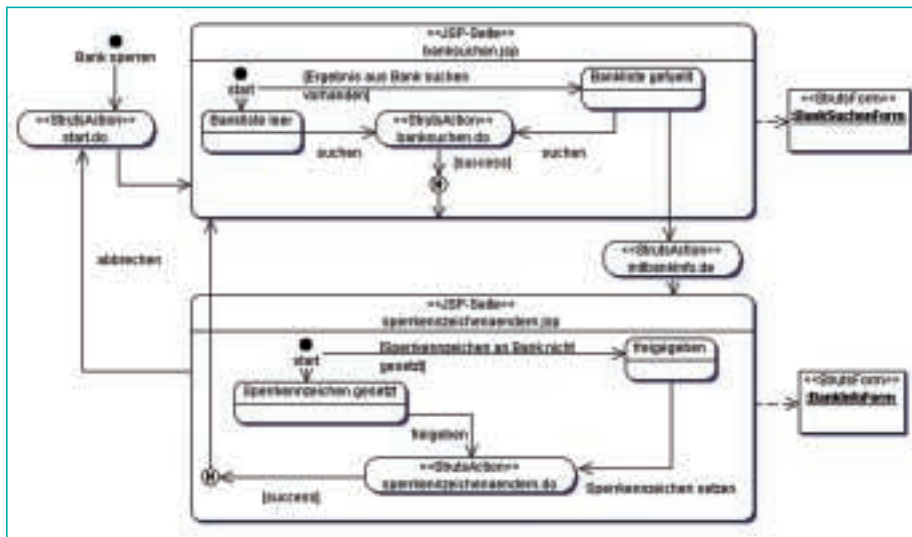


Abb. 2: Modellierung des Workflows mittels State Charts

bestimmt. Die Beendigung dieser Methode führt dazu, dass zum nächsten Schritt übergegangen wird. Diese Transitionen werden daher als anonyme Transition (ohne Benennung des Events) mit dem Identifikator für das *forward-mapping* als *Condition* modelliert. Einzige Ausnahme von diesen Übergangsregeln ist das Exception Handling von Struts, das eine andere Beschreibung des Workflows als über das *action/forward*-Prinzip zulässt (siehe dazu Plausibilitätsregeln/Best Practices).

Die JSPs selbst können ebenfalls noch interne Status besitzen, die Einfluss auf den Workflow haben. Abgehende Transitionen der inneren Status können nur ausgeführt werden, wenn sich die Seite (Applikation) in diesem Status befindet.

In unserem kleinen Beispiel (Listing 2) können Bankinstitute mit einem Sperrkennzeichen versehen werden. In einem ersten Schritt wird über ein Suchmuster eine Menge an passenden Bankinstituten ausgewählt. Falls über die Suche mindestens ein Institut ermittelt werden kann, so kann eines der gefundenen Institute ausgewählt und in einem nächsten Schritt sein Sperrkennzeichen gesetzt/zurückgesetzt werden. Das Diagramm in Abbildung 2 spiegelt exakt den dargestellten Sachverhalt wider.

### Plausibilitätsregeln/Best Practices

Jede abgehende Transition einer *action* entspricht einem *forward*. Die Bedingung der Transition ist der Name des *forwards*.

Es können *forwards* definiert werden, die für alle *actions* gelten und immer dasselbe Verhalten besitzen. Solche globalen *forwards* müssen nicht modelliert werden. Die tatsächlich auszuführende Aktion sollte nicht erst durch einen nachgeordneten Request-Parameter ermittelbar, sondern vollständig durch die zugeordnete *action* bestimmt sein. Beispielsweise ist es sinnvoller, jeweils eine *action* für Einfügen, Löschen und Verändern eines Satzes zu modellieren als eine *action*, bei der anhand eines Request-Parameters erst entschieden wird, was tatsächlich zu tun ist. Diese Vorgehensweise erhöht die Verständlichkeit des Modells und erleichtert die Modellierung.

In unserem Vorgehen ist die Modellierung des Struts-eigenen Exception Handlings nicht möglich, da sie sich dem Prinzip der *actionforwards* entzieht. Daher ist es nicht sinnvoll, Struts-eigenes Exception Handling speziell für einzelne *actions* zu definieren. In unseren Projekten setzen wir daher ein globales, für alle *actions* gleiches Exception-Handling ein. Soll eine *action* ein davon abweichendes Exception-Handling implementieren, so definieren wir ein spezielles *forward* mit Namen *error* für diese *action*. Dieses kann dann wieder innerhalb des Modells beschrieben werden.

### Generierung

Nachdem nun das Verhalten der einzelnen Seiten in den Struts-Diagrammen modelliert wurde, sollten die Informationen wie-

der aus dem Modell extrahiert werden und daraus die *struts-config* generiert werden. Für eine erfolgreiche Generierung muss es möglich sein, das für die Beschreibung verwendete UML-Diagramm und die verwendeten UML-Symbole mit den Eigenschaften zu versehen, die später im Generat erscheinen sollen. Weiterhin muss das verwendete UML-Werkzeug über eine Programmierschnittstelle oder eine Exportfunktion verfügen, um die im Modell hinterlegten Eigenschaften auch wieder auslesen zu können. Beide Anforderungen können im Together Control Center sehr komfortabel abgedeckt werden.

Neben der Oberflächenlogik (definiert in *action mapping*) und den Seiteninhalten (definiert in *forms*) enthält die *struts-config* weitere Inhalte wie die Definition der Data Sources, Message Resources und Controller.

Diese Definitionen beinhalten keine Beschreibung der GUI-Navigation, sondern technische Zusatzinformationen für das Struts-Framework. Da unser Modell als Diskussionsgrundlage zwischen fachlicher Analyse und Softwareentwicklung dient, sollten diese Informationen nicht im UML-Modell hinterlegt werden. Wir haben uns deshalb dafür entschieden, diese allgemeinen Teile über ein Ant Script mit den aus dem Modell generierten Teilen zu verknüpfen. Im Folgenden wollen wir uns nur noch mit der Generierung der Action Mappings und der Form-Beans beschäftigen.

Wie Abbildung 2 zeigt, entsprechen die im Struts-Diagramm verwendeten Aktivitäten jeweils einem Element `<action/>` in der Sektion `<action-mappings/>` in der *struts-config*-Datei. Alle Attribute des Elements `<action/>` werden im UML-Modell als eigenes Property bei der Aktivität hinterlegt. Die einer *action* im Attribut *type* zugewiesene Klasse, der Action Handler, definiert nach ihrer Abarbeitung einen *forward*. Dieser *forward* muss, außer es handelt sich um einen *global forward*, bei der jeweiligen *action* als Element *forward* definiert sein.

In unserem Struts-Diagramm werden alle abgehenden Transitionen einer Aktivität als *forward*-Deklarationen interpretiert. Die Transitionen erhalten somit zusätzliche Eigenschaften. Nun steht einer Generierung nichts mehr im Weg. Wie be-

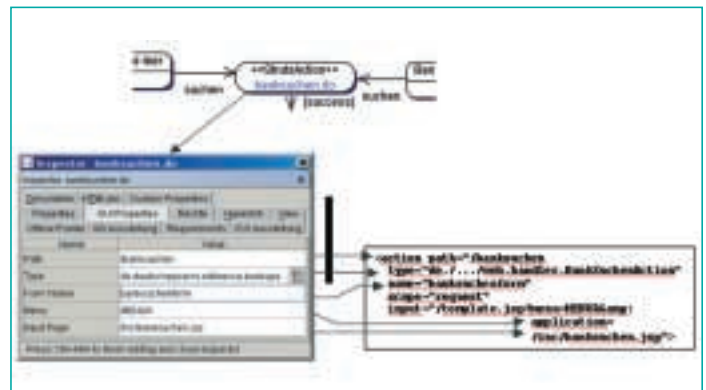
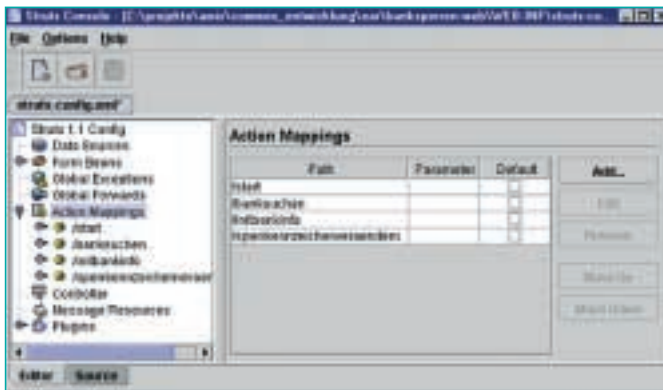
Abb. 3: Editierung der *struts.config* in Struts Console [7]

Abb. 4: Interpretation Struts-Diagramm

reits erwähnt, verwenden wir für die Modellierung Together Control Center. Dieses verfügt über ein sehr mächtiges API, das es erlaubt, über in Java programmierte Erweiterungen (so genannte Modules oder Together-Skripte) direkt auf die UML-Modellinformationen zuzugreifen. Somit ist es nun auch ein Leichtes, die aus dem Modell gewonnenen Informationen wieder in einer XML-Datei auszugeben.

Da es Together erlaubt, die Skripte nicht nur über das Together GUI, sondern auch im Batch auszuführen, lässt sich der Generierungsprozess sehr leicht in einen Weekly Build integrieren. Dies führt dazu, dass die fachliche Modellierung des Workflows jederzeit mit der implementierten Variante übereinstimmt, da niemand mehr die Konfigurationen von Hand erstellen wird.

### Weitere Generierungsmöglichkeiten

An dieser Stelle möchten wir noch einen kleinen Ausblick auf weitere Generierungsmöglichkeiten geben, die potenziell existieren, wenn die Navigation der Oberfläche in Modellform vorliegt: In den Struts-Diagrammen sind alle JSPs, in denen die Geschäftsprozesse abgewickelt werden, vorhanden. Da wir in unserem Projekt eine seitenbasierte Online-Hilfe integrieren werden, hinterlegen wir die Texte für die Hilfe als Properties direkt bei den Symbolen der JSP-Seiten im Modell. Es muss somit für die Beschreibung der Hilfe kein neues Vorgehen definiert werden, sondern es genügt, das bestehende Modell anzureichern.

Im oben beschriebenen Generierungsprozess erzeugen wir nun komplett alle Online-Hilfen als eigene JSPs. Dies hat den Charme, dass bei einer Änderung der Lo-

gik die Hilfen sofort mit angepasst werden können. Des Weiteren kann die Qualitätssicherung (QS), die ohnehin das Modell prüfen muss, die Hilfetexte sofort prüfen, ohne das Werkzeug und das Vorgehen wechseln zu müssen.

### Rechte: Definition/Menüstruktur

Da in Webapplikationen geprüft werden muss, ob ein bestimmter Request vom angemeldeten Benutzer ausgeführt werden darf oder nicht, hinterlegen wir bei den *action*-Definitionen auch gleich das benötigte Recht, um diese Aktion ausführen zu können. So verfahren wir auch mit den Rechten auf Applikationsmenüs, deren applikationsabhängige Struktur in Aktivitätsdiagrammen definiert werden. Hierbei entsprechen die Zustände den aufklappbaren Menüs und die Aktivitäten den ausführbaren Menüpunkten. Zu jedem Menüpunkt definieren wir das benötigte Recht. So lässt sich zusätzlich zu den Rechtedefinitionen die gesamte Definition der Menüstruktur generieren.

### Fazit

Missverständnisse zwischen den Fach- und den IT-Abteilungen beinhalten Risiken und kosten Kraft und Geld. Durch die beschriebene Vorgehensweise ist es uns gelungen, eine Grundlage zur Diskussion der Ablaufsteuerung in Webanwendungen zu finden. Voraussetzung hierfür ist, dass beide Seiten, Fachabteilung und SW-Entwicklung, die UML für ihr Einsatzgebiet beherrschen. Der Einsatz dieses Modells reicht von der reinen Beschreibungssprache bis zur Modellierung des Struts-Workflows und weiterer workflowabhängiger Infor-

mationen im Modell. Durch die genannten Verfahren ist die Akzeptanz der Navigation der Webanwendung, die Qualität der Modelle und die Stabilität der Konfiguration erheblich gestiegen. Zusätzlich wird durch den massiven Einsatz der Generatoren der Nachweis erbracht, dass die fachlichen Definitionen mit der implementierten Anwendung übereinstimmen. ■

*Andreas Lux (a.lux@excellent.de), Projektleiter und Gesellschafter der excellent solutions GmbH. Er befasst sich seit vielen Jahren mit der Erstellung von Individualsoftware für Businessapplikationen im Umfeld Java und Internettechnologien. Christoph Schmidt-Casdorff (c.schmidt-casdorff@iks-gmbh.com) ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme mbH tätig. Er beschäftigt sich seit mehreren Jahren mit der Konzeption und Umsetzung von Systemarchitekturen auf Basis von Java und J2EE.*

### Links & Literatur

- [1] Sven Haiges: Tools zum Verstärken. Kostenlose und kommerzielle Tools im Überblick, in *Java Magazin* 10.2003
- [2] Ted Husted, Craig R. McClanahan, David Winterfeldt: *Struts in Action: A Practical Guide to the Leading Java Web Framework*, Independent Publishers Group 10/2002
- [3] Chuck Cavaness: *Programming Jakarta Struts*, O'Reilly Press 11/2002
- [4] Sue Spielman: *The Struts Framework*, Morgan Kaufmann Publishers 10/2002
- [5] TogetherSoft: [www.togethersoft.de/](http://www.togethersoft.de/)
- [6] Struts-Homepage: [jakarta.apache.org/struts/](http://jakarta.apache.org/struts/)
- [7] UML: [www.omg.org/](http://www.omg.org/)
- [8] Struts Console: [www.jamesholmes.com/struts/console/](http://www.jamesholmes.com/struts/console/)