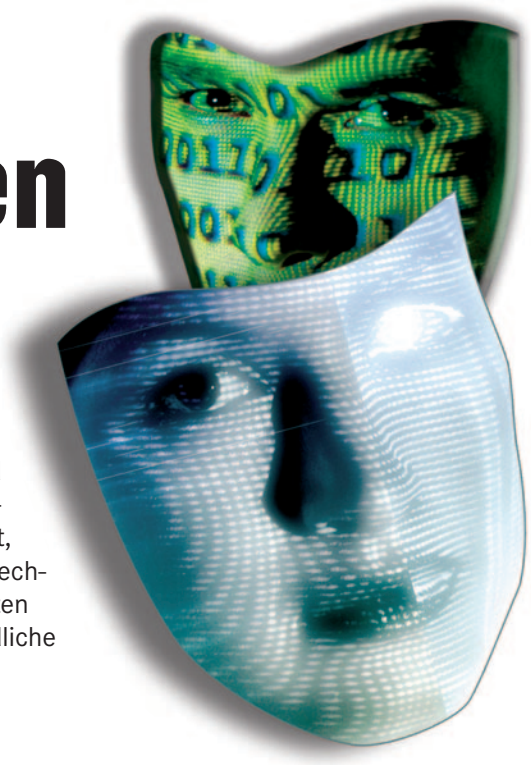
Aus dem Struts-Lager deutlich zu hören: Weiter geht es mit JavaServer Faces!

Von Sackgassen und Neubaustraßen



VON PAPICK GARCIA TABOADA

JavaServer Faces (JSF) ist eine relativ junge Spezifikation für die Entwicklung von grafischen Oberflächen für serverseitige Anwendungen. Als einfaches User Interface Framework geboren, wurde JSF von der Java-Community gleich zum Web Application Framework (WAF) geschlagen. Im gleichen Atemzug wird JSF heute mit Struts genannt, mal als Ergänzung, mal als überlegene Technologie und meistens als konkurrierendes Framework. Es gibt Stimmen im Internet, die einen direkten Vergleich beider Technologien anstreben [1] [2]. Derartige Technologievergleiche sind nicht ausreichend, da weder Struts noch JSF alle Facetten der Webentwicklung abdecken. Beide Frameworks lösen teilweise unterschiedliche Probleme, interessant wird es erst dann, wenn man das Umfeld betrachtet.

Wie oft wird man im Rahmen eines Projektes gefragt, warum das Erstellen einer einfachen Webanwendung so aufwendig ist? Die Problembereiche Multi-User, Multi-Threading, Internationalisierung, Testbarkeit, Erweiterbarkeit und Skalierbarkeit sind doch dank Java EE durch moderne Frameworks und die dazu passenden Application Server längst mehr oder weniger elegant gelöst. HTML, Cascading Style Sheets und JavaScript sind bekannte Technologien, die Browser nicht mehr ganz so verschieden in der Anzeige und Interpretation dynamischen Inhaltes, die User inzwischen mit dem Umgang von Webanwendungen vertraut – wo bleiben dann die Risiken, der Aufwand, die Rechtfertigung für die immense Komplexität?

Die Antwort zu dieser einfachen Frage ist aus Sicht der Entscheider alles andere als einfach zu verstehen, genauso wenig, wie eine Webanwendung für das Entwicklerteam eine triviale Problemstellung ist. Erst die Entscheidung für eine bestimmte Architektur und die technologische Infrastruktur ermöglicht eine Aussage über das richtige Framework bzw. den technologischen Ansatz. Bereits die Wahl eines Werks aus der Literatur zum Thema „Webanwendungen" gestaltet sich kontextabhängig: Je nach Art des Buches wird das Thema von völlig verschiedenen Blickwinkeln durchleuchtet. Es werden oft Grundlagen (Model 1- und Model 2-Webanwendungen [3]) und die dahinter stehenden Prinzipien vorgestellt. Einige Werke befassen sich mit dem Thema aus Sicht der Anwendung des Frameworks, wie z.B. Multi-Channeling-Webanwendungen mit XML und XSLT durch das Apache Cocoon-Projekt. Als Webentwickler und als Softwarearchitekt muss man heute ein sehr breites Spektrum an Wissen abdecken – sowohl die Architektur und die dazu gehörenden abstrakten Hintergründe als auch die jeweilige Umsetzung und die verwendeten Technologien müssen verstanden werden.

Was an dieser Stelle deutlich werden soll: Es ist weniger die Frage nach dem Vergleich zwischen Struts und JavaServer Faces, die entscheidungsrelevante Antworten liefern wird, sondern es ist die Frage nach dem Umfeld und der zu füllenden Lücken, die hier gestellt werden sollte. Im Prinzip dürfen wir Struts und JSF nicht vergleichen, indem wir die einzelnen technischen Features konkret gegeneinander halten, sondern indem wir sie einem abstrakten Konzept eines möglichst vollständigen Web Application Framework (WAF-42 [4]) gegenüberstellen. Ich wähle an dieser Stelle beispielhaft einige Punkte aus einem zu definierenden abstrakten WAF aus:

- logische vs. physikalische Navigation
- Programmiermodell und Infrastruktur
- Komponentenmodell und die Wirtschaftlichkeit
- Grenzkonflikte HTML und Objektorientierung
- Investitionsschutz vs. Zukunftssicherheit

Logische vs. physikalische Navigation

Eine Webanwendung basiert grundlegend auf Seiten – in der EDV-Welt auch gerne Masken oder User Interface genannt –, die meist in HTML beschrieben und serverseitig abgelegt bzw. dynamisch generiert werden. Diese Masken werden dann

übertragen und im Browser angezeigt. In dieser Seiten-Masken-UI-Beschreibungssprache hat man die Möglichkeit, über Formulare und Links Seiten miteinander zu verknüpfen. Dadurch entsteht eine Art Navigation, Links oder Knöpfe führen den Benutzer von Maske zu Maske. Das ist eine sehr reduzierte Beschreibung der zugrunde liegenden Technologie, viel mehr wird jedoch von dem Gespann HTTP und HTML nicht angeboten. Wenn wir Java-Server Pages nach diesem Vorbild entwickeln, indem wir von der einen JSP auf die nächste verweisen, dann sprechen wir von einer physikalischen Navigation. Mit dieser Technik werden wir unsere ersten „Hello World"-Beispielanwendungen entwickeln. Diese Vorgehensweise bekommen wir in JSP- und Servlet-Einführungen gelehrt und so werden die ersten Gehversuche in Projekten gemacht.

Mit dieser durchaus trivialen Technik bilden Entwickler ganze Prozesse ab, von einem Login-Vorgang bis hin zu einer Buchung und Bezahlung des nächsten All-inclusive-Urlaubs. Eine komplette Webanwendung besteht natürlich nicht nur aus einem Login und einem Anwendungsfall. Die Anwendung wächst üblicherweise sehr schnell und in der gleichen Geschwindigkeit verlieren wir den Überblick und sind nicht mehr in der Lage, diese zu warten, neue Navigationspfade anzulegen oder, noch schlimmer, diese zu verändern.

In einer Webanwendung werden wir verschiedene Arten von Navigationspfaden vorfinden. Einmal haben wir eine prozessbedingte Navigation bedingt durch die Abarbeitung eines Anwendungsfalls. Des Weiteren haben wir die globale Anwendungsnavigation, bei der Anwender durch Menüs zu dem entsprechendem Einstiegspunkt eines Anwendungsfalls gelangen. In der zweiten Fassung der Java EE Core Pattern [5] wurde der ursprüngliche Front-Controller-Gedanke in einen Navigations- und einen Business Controller erweitert.

Nehmen wir zum Beispiel ein Praxisverwaltungssystem. Hier muss sich der Therapeut in seiner Praxisverwaltungssoftware bis zum Patienten oder bis zum Terminplan durchklicken, bis er einen neuen Termin vergeben kann. Dies gestaltet sich dann unterschiedlich, je nachdem aus welchem Navigationspfad der Therapeut gekommen ist: Mal ist der Patient bereits ausgewählt, mal der gewünschte Termin. Der Navigations-Controller ist für die Navigation in die Terminplanansicht bzw. Patientenverwaltungsansicht zuständig. Der Business Controller ist für die Navigation innerhalb des Terminvergabeprozesses zuständig und muss entsprechend der Vorgaben die richtigen Masken anzeigen.

Bei einer in Teilbereichen gegliederten Webseite ist es sinnvoll, den Navigations-Controller entscheiden zu lassen, welches Ausgabefragment in einen bestimmten Teilbereich der Gesamtausgabe geschickt wird. So gesehen übernimmt der Navigations-Controller auch die Rolle des Layout-Managers. Das entspricht in etwa der serverseitigen Realisierung von Framesets, wie wir sie aus HTML kennen.

Moderne Webanwendungen haben ein gegliedertes Layout (besonders bekannt bei Portalen), in dem Teilbereiche definiert sind. So wird sich ein Anwender sehr schnell zurechtfinden, wenn in der linken Spalte ein Navigationsmenü zu finden ist.

Damit der Anwender nicht in die Irre geführt wird, sollten nicht gültige Menüeinträge ausgeblendet werden (alternativ nicht auswählbar gemacht werden) oder sogar das gesamte Menü verschwinden. Was in Rich Clients trivial ist, muss in einer Webanwendung dann mühsam nachgebaut werden.

Eine der wichtigsten Eigenschaften einer Model 2-Webanwendung [3] ist, einen zentralen Einstiegspunkt zu definieren. Nutzt man diesen Ansatz, so kann man in dieser Einstiegsseite (JSP oder Servlet) die gewünschte Ausgabe aus Teilbereichen zusammenstellen. Das Zusammenstellen wird dann statisch definiert oder zur Lauf-

Anzeige

zeit dynamisch durch eine Komponente – einen Navigations-Controller – gesteuert. Dieser Ansatz macht eine logische Navigation erst möglich. Der angesprochene Navigations-Controller kann sowohl über HTTP-Parameter als auch über den in der Sitzung gespeicherten Zustand der Anwendung gesteuert werden. So kann ein Navigations-Controller z.B. durch die URL *app.home?navigate=moduleB* dazu gebracht werden, Modul B anzuzeigen und den Wechsel in der Sitzung festzuhalten.

Durch den Einsatz eines zentralen Einstiegspunktes (Central Point of Entry, Central Point of Failure) ist die physikalische Navigation nicht mehr sinnvoll. Die einzelnen JSP-Dateien sind Fragmente, die nur in Verbindung mit dem Rahmen der Webanwendung und den darin enthaltenen Navigations- und Business-Controllern Gültigkeit haben.

Weder Struts noch JSF bieten Lösungsansätze, mit denen man die oben beschriebenen Navigationsprobleme vollständig lösen kann. Je nach Technologie sind aber Ansätze vorhanden und teilweise bereits in der Praxis umgesetzt (siehe Kasten „Logische Navigation“).

Programmiermodell und Infrastruktur

Sehr viele der uns heute bekannten Frameworks im Java-Umfeld sind invasiv: Klassen werden Interfaces oder Vererbungshierarchien aufgezwungen. Die Bindung zwischen Logik und Framework ist meistens hoch, der Traum der Modularisierung vernichtet. Moderne, elegantere Ansätze finden wir in Frameworks wie Spring [8] oder PicoContainer [9]. Die dort eingesetzte Vorgehensweise ist zwar nicht neu, aber erst mit und dank AOP so elegant zu lösen.

Es ist erstrebenswert, die eigene Anwendungslogik von der verwendeten Infrastruktur zu entkoppeln. Wie man die Abhängigkeiten zur Infrastruktur entkoppelt, wird sowohl in der Literatur als auch in IOC-Containern [10] vorgeführt. Tatsache ist, dass Entwickler diese Entkopplung nur dann konsequent einhalten werden, wenn dadurch Vorteile entstehen. Aus der Sicht des Entwicklers liegen die Vorteile darin, dass Probleme (insb. Abhängigkeiten) mit weniger Kodieraufwand gelöst werden. Muss ich mir diese Entkopplung durch eigene Adapterklassen erst selbst entwickeln, wird es wohl halbherzig durchgesetzt werden – erst kurz vor Deadlines.

Struts bietet an dieser Stelle keine Hilfe. Die Action-Klassen müssen von einer bestimmten Klasse erben, das Framework ist invasiv. Wenn der Entwickler nicht sehr behutsam vorgeht und die gesamte Logik

Logische Navigation

… mit Struts und Tiles

Schon seit einiger Zeit findet man in der Struts-Distribution die Templating Engine Tiles. Mit Tiles wollte man ursprünglich das Composite View Pattern für Webanwendungen serverseitig realisieren. Ein sehr interessantes Feature ist, das man bei der Verwendung einer in Tiles definierten Ansicht zusätzlich eine Struts-Action-Klasse als Navigations-Controller angeben kann. Diese kann dann die in der Tiles-Konfigurationsdatei definierten Teilbereiche über ihren Tiles-Kontext verändern. Das Verhalten des Navigations-Controllers darf dann ausprogrammiert werden: Wie und was als Entscheidungsgrundlage genommen wird, wird dem Entwickler überlassen bzw. muss durch ein Rahmenwerk bereitgestellt werden. Die Zahl der Klassen, die bei der Anzeige einer fertigen HTML-Seite beteiligt sind, lässt sich üblicherweise nicht mehr an einer Hand abzählen.

… mit JSF

Die Navigation ist in JSF sehr elegant, aber leider nur teilweise gelöst. Hier kennt man keine physikalische Navigation: Wir arbeiten nur noch mit Events und haben kein Wissen über URLs. Seiten sind „Views“ und werden von dem internen Navigations-Controller angesteuert. Dem eingebauten Navigations-Controller wird man durch „Outcomes“ signalisieren, wohin die Reise geht. Dieser Navigations-Controller kann aber leider nur ganze Views anspringen, da Templates nicht unterstützt werden. An dieser Stelle werden wir uns noch nach Lösungsansätzen umschauen müssen. Die aktuelle Spezifikation umfasst weder Templates noch die passenden Navigations-Controller. Es gibt verschiedene Ansätze, in denen versucht wird, Tiles aus der Struts-Distribution zu entkoppeln, um es als eigenständige Komponente mit JSF verwenden zu können. Wobei die Schwierigkeit im zweiten Teil liegt. Da die Komponenten in JSF nicht dem trivialen Request-Response-Zyklus unterliegen, ist Tiles in der aktuellen Realisierung nicht so einfach zu integrieren.

Unter dem Apache Struts-Projekt entsteht ein neues Framework namens Struts Shale [6]. Mit Struts hat man nur den Namen gemeinsam, es handelt sich hier um einen vollkommen neuen Ansatz. Ein gestecktes Ziel von Shale ist, es den Entwicklern zu erleichtern, JSF einzusetzen. Dabei meint man nicht die Portierung bestehender Struts-Anwendungen, sondern die Verwendung im Allgemeinen. So werden wir bei Shale einen Dialog-Manager, Remoting (auch unter dem Namen AJAX [7] bekannt) und ein Plug-in, mit dem man es leichter hat, Komponentenbäume wieder zu verwenden (Clay), finden.

Bis die Java Community diese Ansätze annimmt und sie sich bewährt haben, bleibt dem JSF-Lager erst einmal nur der Blick in Richtung Portale oder einer eigenen Entwicklung. Es ist auch nicht unbedingt als Fehler der Spezifikation zu sehen. Für die hier beschriebenen Probleme würde man eher die Lösung bei der Portalspezifikation (JSR 168) suchen – leider vergeblich. Jedenfalls sind es Portallösungen mit JSF-Integration, die uns begehbare Wege zeigen. Zwar findet man in der JSF-Spezifikation 17-mal das Wort „Portlet“, aber in der Portlet-Spezifikation findet man JSR 127 (der JSR für JSF) bzw. JSF nur einmal – bei den Danksagungen.

Eventuell müssen wir hier noch eine Runde bei den Spezifikations-Releases warten: JSP und JSF müssen sich annähern, die Expression Language muss schnellstens vereinheitlicht werden. Portlets sollten meiner Meinung nach mehr von JSF wissen, vielleicht sogar Kompatibilität vorgeschrieben bekommen.

JSF wird oft als das fehlende Puzzlestein im Java EE-Umfeld beschrieben. Das ist in meinen Augen eine sehr gute Metapher, die ich noch erweitern möchte: Man hat das fehlende Puzzleteil nicht gefunden, sondern eines aus einem viel neueren Puzzle gestohlen – es passt einfach nicht so richtig, ist aber ein sehr schönes Puzzleteil. Die Seite, die zu JSP angrenzt, passt noch nicht hundertprozentig: So sind die verwendeten Expression Languages noch sehr unterschiedlich und verkanten manchmal. Dieser schlechte Übergang soll mit JSP 2.5 passend gemacht werden. Die Integration in die Portallösungen ist herstellerspezifisch, da schweigt sich die Spezifikation komplett aus. Als Entwickler darf man sich schon wundern, schließlich stehen alle Technologien unter der Schirmherrschaft des einen Hauses – und dennoch passt es nicht wirklich zusammen.

in eigene Objekte kapselt, wird man die Anwendungslogik verteilt auf viele kleine Struts-Klassen wiederfinden. Entkopplung wird nur geschaffen, wenn eine eigene Indirektionsschicht eingebaut wird: Die Action-Klasse ruft eine andere Klasse auf, die nicht von Struts-Klassen erben muss – das ist ein teurer Preis für die Entkopplung. Dieser macht sich aber spätestens bei der Testbarkeit der Logik wieder bezahlt. Bedingt durch seinen monolithischen Aufbau lassen sich einzelne Komponenten aus Struts nicht entkoppeln und sind schwer zu ersetzen. Auch sind Erweiterungen nicht unbedingt zueinander kompatibel.

JSF hat es durch die Umsetzung des Request-Response-Zyklus in Event-Benachrichtigungen um einiges leichter. Hier hat man sogar auf Interfaces verzichtet und das Anmelden eines Methodenaufrufs als Reaktion auf einen Event ermöglicht. Im Optimalfall gibt es keine Abhängigkeit zwischen der aufgerufenen Bean und dem JSF-Framework. Die Bindung zwischen den Beans und den Masken wird über eine neue Syntax abgebildet, sodass die Maskenbeschreibung direkt die Bindung zu den Methoden einer Bean definiert. Dadurch ist die Logik der Anwendung technologieneutral in POJOs [11] aufgehoben und kann wieder verwendet werden. Es ist schließlich möglich, diese Klassen von der JSF-Anwendung, einem Testsystem oder sogar von einem RCP Rich Client aus zu verwenden. Eine stärkere Entkopplung bekommt man z.B. durch den Einsatz von Spring, da hier die Integration zu JSF besonders elegant gelungen ist.

Komponentenmodelle und die Wirtschaftlichkeit

Aus der Entwicklung von Rich Clients sind grafische Komponentenmodelle bekannt und weit verbreitet. Allein in der Java-Welt stehen dem Entwickler AWT, Swing, SWT und JFace zur Verfügung. Kein Rich-Client-Entwickler würde auf die Idee kommen, sich eine Baumansicht seiner Datenstruktur von Hand zu malen. Umso größer ist dann der Kulturschock, wenn man in die Fraktion der Webentwickler wechseln darf. Grafische Komponenten sind hier keine Regel, sondern die Ausnahme. Je nach Projekt wird die eine oder andere Bibliothek dazugekauft oder selbst entwickelt. Sehr oft reicht das Wissen oder die Zeit nicht dazu aus, diese Komponenten so konfigurierbar zu gestalten, dass diese wieder verwendet werden können, oder die Lizenzen lassen es nicht zu, dass die kürzlich eingesetzte Bibliothek im nächsten Projekt wieder eingesetzt wird.

Durch das Fehlen eines einheitlichen grafischen Komponentemodells für Java EE-Webanwendungen zeichnete sich eine mittelgroße Katastrophe ab, die erst jetzt durch JSF abgewendet werden kann. Nicht nur, dass Entwickler sich nicht immer wieder von Projekt zu Projekt in neue Komponentenmodelle einarbeiten müssen, nun kann ein Markt entstehen, in dem verschiedene Anbieter kompatible grafische Komponenten anbieten können.

Bei der Entwicklung von Struts-Anwendungen gibt es verschiedene Bibliotheken. Einige dieser Komponenten sind lediglich als Taglibs implementiert. Wie die Daten angezeigt werden und welche Voraussetzungen erfüllt sein müssen, wird von Komponente zu Komponente unterschiedlich sein. Welche Requests von der in HTML gerenderten Komponente ausgelöst werden und wie Entwickler dann reagieren können, ist auch sehr unterschiedlich gelöst. So wird bei der nächsten Komponentenbibliothek Struts so erweitert, dass verschiedene Callback-Methoden je nach Anwendungsfall serverseitig aufgerufen werden. Ob und wie sich eine zugekaufte Komponente in die eigene Anwendung einbinden lässt, kann nicht pauschal gesagt werden und verlangt eine genaue Untersuchung.

Das bei JSF definierte Komponentenmodell ist im Umfang nicht so bunt geraten, wie es die Praxis heute verlangt, bildet aber die notwendige Basis, um Kompatibilität zwischen Komponenten zweier Anbieter zu gewährleisten. So lassen sich zum Beispiel die Komponenten von MyFaces [12] auch unter der Referenzimplementierung von Sun einsetzen (siehe Beitrag ab Seite 40 in dieser Ausgabe).

Weder die Komponenten von Sun noch die von MyFaces sehen auf den ersten Blick besonders attraktiv aus, lassen sich aber hervorragend über Stylesheets anpassen. Vielen Teams fehlt an dieser Stelle das nötige Know-how, da nicht jeder Entwickler in die Designerrolle springen kann oder

Anzeige

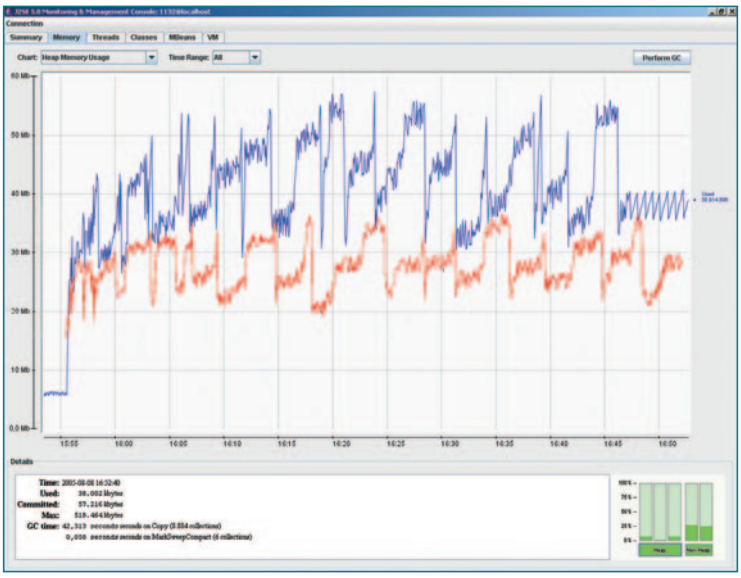


Abb. 1: Speicherverbrauch über die Zeit

will. Bei Struts-Anwendungen wurde diese Lücke oft durch den Einsatz von kommerziellen Lösungen geschlossen – bei JSF wird es mit großer Wahrscheinlichkeit nicht anders sein. Allerdings müssen wir bei JSF nicht jedes Mal das Haus abreißen, wenn wir eine neue Farbe an den Wänden haben wollen – das Austauschen des Render Kit dürfte meistens ausreichen. Auch die Investition in ein auf die eigene Corporate Identity angepasstes Render Kit würde sich lohnen, da mehrere Projekte gleichzeitig davon profitieren können.

Grenzkonflikte HTML und Objektorientierung

Seit den 40ern entwickeln Wissenschaftler an immer schnelleren Rechnern, wir haben die unterschiedlichsten Programmiersprachen und Paradigmen kommen und gehen sehen – heute werden sogar die Bremsen unserer Autos von Computern gesteuert, mein Navigationssystem kennt alle wichtigsten Straßen Europas und spricht mich in meiner Muttersprache Portugiesisch an –, aber unsere Browser sind nicht in der Lage, zwischen Text und Zahl zu unterscheiden.

In der Evolution der Browser- und der HTML-Spezifikation haben interessanterweise nicht die Besseren überlebt. Im Krieg der Browser wurden wir mit innovativen und sinnvollen Features wie z.B. blinkenden Texten überhäuft, aber nach einem Datumsfeld für Formulare sucht man vergeblich. Hätte man den Browser bzw. HTML gleich zu Beginn um ein solides Komponentenmodell erweitert, blieben uns heute einige technologische Klimmzüge erspart. Unsere Lösungen sind alles andere als elegant und irgendwann, hoffentlich, werden wir uns mit Grauen an diese Zeiten erinnern.

Solange wir aber nicht das Tree Tag in HTML haben und wir uns daher serverseitig überraschen lassen müssen, ob Zahlen oder Texte abgeschickt worden sind, müssen wir weiterhin Klimmzüge machen. An diesem Grenz- übergang von untypisierten Daten aus dem HTML/Browser-Gespann in die serverseitige Anwendung findet ein Paradigmenwechsel statt. In einer Struts-Anwendung haben die Form Beans, die Validierungsmethoden und die Action-Klassen die Aufgabe, die Daten aus der einen in die andere Welt zu übertragen. Die Form Beans repräsentieren den Zustand der Webformulare, werden ausgelesen und die Daten dann weiterverarbeitet.

Bei JSF ist die Laufzeitumgebung für diese Übersetzung zuständig. In der Maskenbeschreibung werden die Konvertierungs- und Validierungsregeln definiert, Entwickler müssen keine Form Beans definieren und können mit den eigenen Klassen arbeiten [13]. Durch den Einsatz von Validierern und Konvertierern werden Entwickler mit neuen Problemen konfrontiert: Wenn die mitgelieferten Komponenten nicht ausreichen, müssen eigene entwickelt werden. Das ist ein nicht zu unterschätzendes Problem: Wir müssen lernen, komponentenbasiert zu entwickeln, das Komponentenmodell zu verstehen, und auch noch wissen, wie man eigene Komponenten entwickelt oder bestehende erweitern kann. JSF verschiebt diesen Grenzübergang so weit von der Business-Logik weg, wie es nur geht: in die Maskendefinition. Die Probleme werden dort gelöst, wo sie auftreten: Konvertierung, Validierung und Anwendungslogik werden sauber getrennt entwickelt. Langsam nähern wir uns zu fairen Kosten dem OO-Versprechen Wiederverwendbarkeit.

Investitionsschutz vs. Zukunftssicherheit

Noch immer gehen Struts-Projekte an den Start, obwohl seit einiger Zeit mit JSF ein Standard als Konkurrent zur Verfügung steht. Einer der mir am häufigsten genannten Gründe ist der Investitionsschutz. Schließlich habe man die Entwickler geschult, ein eigenes Navigations-Framework gebaut, eventuell eigene Taglibs bzw. Bibliotheken erworben, viele Tools getestet und sich für einige entschieden. Das alles kostet Zeit. Viel Zeit. Kein Wunder also, dass man diese Prozedur nicht noch einmal durchleben möchte. Keiner möchte der Erste sein, keiner möchte die Kinderkrankheiten einer recht jungen Technologie ausbaden müssen.

In letzter Zeit nehmen wir bei Kunden zunehmend den Trend zu einer Portallösung wahr. Der Hintergrund ist meistens ähnlich: Man hat in den letzten Jahren eine bunte Landschaft von zueinander inkompatiblen Anwendungen entwickelt und dabei gute und schlechte Erfahrungen gemacht. Jetzt ist es Zeit zu konsolidieren, die wichtigsten Anwendungen zu identifizieren und sie unter das Dach eines Portals zu bringen. Single Sign-on und eine gemeinsame Infrastruktur sind wichtige Themen geworden. Hier werden bestehende Struts-Anwendungen es besonders schwer haben, in eine Portallösung hineinportiert werden zu können: Je größer die Anwendung, umso wahrscheinlicher hat man eigene Navigationskonzepte umgesetzt und auf die eine oder andere Weise ein mehrspaltiges Layout realisiert. Diese Aufgaben soll jetzt die Portallösung übernehmen. Auch wenn Hersteller von Portallösungen mit Struts-Integration wer-

ben, so werden die meisten Entwickler und Softwarearchitekten, in einer ernsthaften Minute gefragt, zugeben: Unsere Struts-Anwendung – da sind wir sicher – bekommt keiner so einfach integriert. Der Punkt ist: Jedes Projekt löst die Navigationsprobleme unterschiedlich. Das Mapping der Struts-Konfiguration ist bei weitem nicht ausreichend, die meisten setzen auf Tiles oder andere Lösungen. Diese Anwendungen kann man nicht einfach nahtlos in ein Portal einbinden.

Auch wenn die Vorteile des JSF-Konzeptes deutlich sind, werden wir in Zukunft noch sehr viele Struts-Projekte sehen. Schließlich ist Struts eine in der Praxis erprobte Technologie, die sich auch in größere Projekten bewiesen hat. Der Investitionsschutz spricht also deutlich für Struts. Für JSF spricht die Zukunftssicherheit: Es handelt sich hier um einen Standard unter den Fittichen von Sun, der von der Industrie angenommen wurde, es entstehen verschiedene Implementierungen, Tools, Komponenten und Render Kits.

Während das Struts Framework als performant gilt [14], gibt es kaum Aussagen zu JSF. Gerade bei JSF sind Bedenken gerechtfertigt, schließlich führt JSF eine Laufzeitumgebung ein und baut serverseitig Objektbäume auf, um die in den Seiten verwendeten Komponenten abzubilden.

Würde ich auch größere Projekte mit JSF realisieren? Prinzipiell spricht nichts dagegen. Eine einfache Versuchsreihe mit den bei MyFaces gelieferten Beispielen ergab, dass die Laufzeitumgebung 30 bis 50 gleichzeitige Nutzer verträgt. Sicherlich keine Aussage über die Skalierbarkeit oder das Lastverhalten der Technologie, aber zumindest ist sie nicht kollabiert (Abb. 1).

Fazit

Aus dem Blickwinkel der Navigationssteuerung lassen beide Ansätze Fragen unbeantwortet, die man im Struts-Lager teilweise mit Tiles und bei JSF unter Umständen mit Portalen beantworten kann. Vollständig im Sinne eines WAF-42 ist keiner der Ansätze. Das Programmiermodell von Struts ist deutlich invasiver als das von JSF. JSF definiert ein Komponentenmodell für grafische Elemente – in diesem Bereich macht Struts überhaupt keine Aussagen. Auch im Grenzkonflikt zwischen HTML und Objektorientierung hat JSF die elegantere Lösung aufgetischt.

Die deutlichste Anwort kommt direkt aus dem Struts-Lager: JSF ist die Zukunft [6]. Man sollte sich nicht von Struts Shale in die Irre führen lassen. Struts Shale hat mit Struts wenig bis gar nichts gemeinsam. Es wurde von den Entwicklern nicht als Struts 2.0 angenommen (weil es kein Struts ist) und man wollte den aufwendigen Prozess für das Anlegen eines neuen Apache-Projektes vermeiden.

Wir haben es in der Vergangenheit schon oft erlebt. Probleme werden identifiziert und gelöst und gerade im Java-Umfeld wurde das Problem Web Application Framework viel zu oft gelöst: Es gibt so viele Web-Frameworks, sodass man es schwer hat, sich zu entscheiden. Das Erstellen einer Webanwendung bleibt eine anspruchsvolle Aufgabe, die einfache Webanwendung gibt es schlichtweg nicht. Mit der Spezifikationskeule JSF hat Sun endlich eine Richtung vorgegeben, wir warten gespannt auf die nächsten Schritte.

Papick Garcia Taboada ist freiberuflicher Softwarearchitekt. Sein Fokus liegt auf der Architektur von Java EE-Anwendungen, agiler Softwareentwicklung und Softwareentwicklung mit Open-Source-Technologie.

Links & Literatur

[1] Roland Barcia: JavaServer Faces (JSF) vs. Struts: websphere.sys-con.com/read/46516.htm

[2] Matt Raible: Comparing Web Frameworks: equinox.dev.java.net/framework-comparison/WebFrameworks.pdf

[3] Govind Seshadri: Understanding Model 2 Architecture: www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[4] de.wikipedia.org/wiki/42_(Antwort)

[5] Deepak Alur, John Crupi, Dan Malks: Core J2EE Patterns, Prentice Hall, 2003

[6] struts.apache.org/shale/

[7] AJAX: de.wikipedia.org/wiki/AJAX

[8] www.springframework.org

[9] www.picocontainer.org

[10] Martin Fowler: www.martinfowler.com/articles/injection.html

[11] Plain Old Java Objects: de.wikipedia.org/wiki/POJO

[12] myfaces.apache.org

[13] Orthogonality and the DRY Principle: www.artima.com/intv/dry.html

[14] Ted Husted: Is Struts performant?: husted.com/struts/resources/performant.htm

[15] Sven Haiges (Hrsg.): JavaServer Faces, S&S Verlag, 2004

Anzeige