„Best-Practice“-Architekturen für Web-Anwendungen

Jörg Hettel

Hänchen & Partner Beratungsgesellschaft
für Wirtschaftsinformatik mbH
Calwer Str. 1 , 71034 Böblingen

joerg.hettel@haenchen.softwarezentrum.de

Kategorien: Entwurfsmuster, Frameworks, Web-Applikationen

Abstract. Für die Entwicklung von Web-Applikationen haben sich bereits einige grundlegende Architekturentwürfe und Patterns etabliert. Der Vortrag stellt zwei, auf der J2EE-Architektur basierende Entwürfe vor: Eine Architektur zur server-seitigen Eingabevalidierung und eine Architektur zur server-seitigen Client-Steuerung. Beide Ansätze basieren auf dem Model 2 und dem Single-Servlet-Ansatz.

1. Die Charakteristik von Web-Applikationen

Bei der Erstellung von Web-Applikationen wird der Entwickler mit einer ganzen Reihe von Problemen konfrontiert. Oft handelt es sich hierbei um bekannte Problemstellungen, die jetzt allerdings in einer neuen Technologie gelöst werden müssen. Dies sind z.B. Eingabevalidierungen, Mehrsprachigkeit, Authentifizierung und Autorisierung oder Benutzerführung und Abbildung von Business-Transaktionen, um nur einige zu nennen. Die Verwendung von http als zustandsloses Protokoll und HTML als Client-Sprache lassen in der Regel eine Übernahme der in der klassischen Client/Server-Entwicklung existierenden Problemlösungen nicht zu. Ein Anwender kann sich z.B. in einer Web-Applikation jederzeit „wegklicken“, indem er einfach im Browser eine neue URL angibt. Bei einer schlecht konzipierten Anwendung kann er durch Betätigung des Back- oder Reload-Buttons des Browsers z.B. eine Business-Transaktion mehrfach auslösen. Eine korrekte Benutzerführung ist besonders für dialogorientierte, transaktionale Anwendungen eine nicht-triviale Herausforderung. Weiter können z.B. Eingabevalidierungen nicht an Events einer graphischen Oberfläche gekoppelt werden. Hier kann man zwar durch den Einsatz von client-seitigen Script-Sprachen einige der gewohnten Funktionalitäten nachbilden, Validierungen gegenüber dem Business-Modell lassen sich aber hierdurch auch nicht immer realisieren.

Für einige der hier genannten Probleme haben sich mittlerweile allgemeingültige Lösungen in Form von „Best-Practice“-Architekturen herauskristallisiert. Insbesondere wurden von Rossi et al. [6,7] eine Vielzahl von spezifischen, wiederkehrenden Problemstellungen bei Web-Anwendungen durch Patterns charakterisiert. Dieser Beitrag konzentriert sich auf technische Umsetzungen. Hierzu werden ein auf der Java 2, Enterprise Edition (J2EE) basierter Mechanismus zur server-seitigen

Eingabevalidierung und eine Möglichkeit zur server-seitigen Client-Steuerung vorgestellt, wie sie von uns in Projekten eingesetzt werden. Die Architekturen basieren auf den von Dai und Ellis vorgestellten Struktur-Patterns [2], die von uns an einigen Stellen verändert und erweitert wurden.

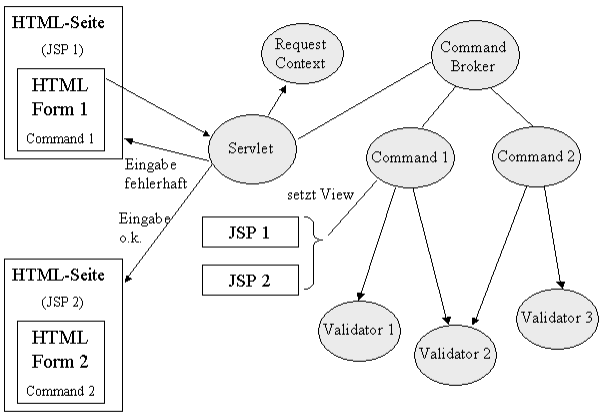


Abb. 1: Beim Single-Servlet-Approach werden alle Requests vom selben Servlet entgegengenommen. Das Servlet analysiert den Request und wendet sich an einen Command-Broker, um ein entsprechendes Command-Objekt anzufordern. Im Command-Objekt findet die eigentliche Verarbeitung statt

2. J2EE basierte Architekturen für Web-Applikationen

Die Java 2, Enterprise Edition (J2EE) bietet heute mit der Servlet- und JavaServer Page-Technologie eine effektive und elegante Möglichkeit zur Erstellung von Web-Applikationen [8,9]. Für den grundlegenden Aufbau von Web-Anwendungen stehen im Prinzip zwei Basis-Architekturen zur Verfügung. Im sogenannte Page-Centric-Approach ("Model 1") werden eingehende Client-Requests direkt von der JavaServer Page entgegen genommen, die auch für den View-Response, d.h. die Darstellung, verantwortlich ist. Diese Architektur führt dazu, dass oft auch in der JSP Logik codiert wird, was zu sehr wartungsintensiven Anwendungen führt. Die Einführung von sogenannten Worker Beans kann die JSP von der Logikcodierung entlasten [1]. Im sogenannten Dispatcher-Approach ("Model 2") gehen Client-Requests an ein Servlet, das diese analysiert und an bestimmte Command-Beans weiter delegiert. Das Servlet übernimmt hier die Rolle eines Controllers und eine JSP die Rolle der View, so dass dieser Ansatz im Prinzip dem klassischen Model-View-Controller Paradigma entspricht. Bei diesem Vorgehen existieren zwei Varianten. In der ersten Variante wird für jeden Client-Request ein spezielles Servlet bereit gestellt. Dies führt zu einer Servlet-View-Verkettung in der Basis-Architektur. Im sogenannten Single-Servlet-Ansatz existiert dagegen für die gesamte Anwendung im Extremfall nur ein einziges Servlet, das alle Requests entgegennimmt. Üblicherweise verteilt das Servlet die

eingehenden Requests mit Hilfe eines Command-Brokers an die jeweilig zuständigen Command-Objekte, die alle von einer abstrakten Command-Klasse abgeleitet sind und die Business-Logik enthalten (vgl. Abb. 1) [4].
Der Single-Servlet-Ansatz bietet durch seinen hohen Abstraktionsgrad eine gute Flexibilität bezüglich Anpassungs- und Erweiterungsmöglichkeiten. Das hier verwendete Servlet stellt lediglich eine generische Verteilungsfunktionalität zur Verfügung. Änderungen und Austausch der Business-Funktionalität beziehen sich nur auf die Command-Objekte und eventuell JSPs, falls Änderungen in der Darstellung vorgenommen werden müssen. Der Command-Broker kann darüber hinaus noch mit einem eigenen ClassLoader ausgestattet werden, so dass bei Bedarf geladene Command-Klassen gegen neue Command-Klassen on-line ausgetauscht werden können. Weiter kann der Broker sehr einfach um eine Trace-Funktionalität erweitert werden. Die von dem Broker verwalteten Command-Objekte können durch Decorator-Objekte verdeckt werden, die z.B. die Aufrufe mitprotokollieren [4].

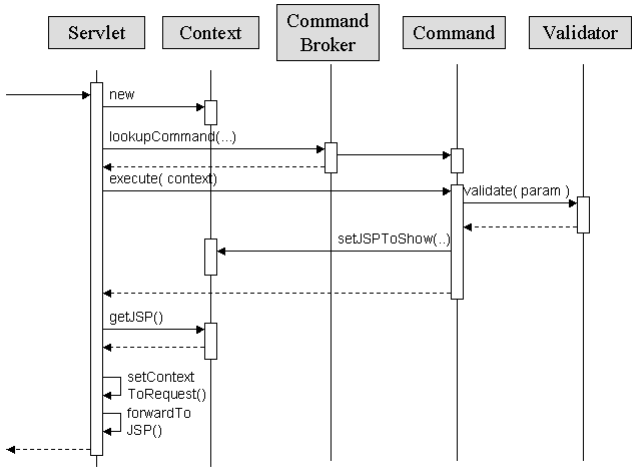


Abb. 2: Bei einer server-seitigen Eingabevalidierung erzeugt das Servlet ein Context-Objekt mit allen Informationen über den Request. Über den Command-Broker wird ein Command-Objekt angefordert, das die Verarbeitung übernimmt. Mit Hilfe von Validatoren werden die Eingaben überprüft. Aufgrund der Überprüfung wird die JSP ausgewählt.

2.1 Server-seitige Eingabevalidierung

Validierungen von Client-Eingaben in HTML-Forms werden in den zugehörigen Command-Objekten vorgenommen. Zu jedem Client-Request wird vom Servlet ein Context-Objekt erzeugt, das alle request-spezifischen Informationen enthält. Insbesondere enthält das Context-Objekt alle Eingabeparameter mit entsprechend gesetzten Werten. Das Command-Objekt enthält eine konfigurierte Liste der zu überprüfenden Parameter, wobei jedem Parameter ein geeignetes Validierungs-Objekt zugeordnet ist. Alle Validierungs-Objekte implementieren ein gemeinsames Interface. Die eigentliche Validierung wird von der abstrakten Command-Klasse durchgeführt,

von der alle Command-Objekte abgeleitet sind. In den konkreten Unterklassen wird noch festgelegt, wie auf eine korrekte bzw. fehlerhafte Eingabe reagiert werden soll. Insbesondere wird von den Command-Objekten die nächste anzuzeigende View (JSP) im Context-Objekt verankert. Fehlen erforderliche Eingaben oder führen eingegebene Parameterwerte zu Fehlern, kann nun die selbe Eingabeseite vom Command-Objekt nochmals an den Client zurück geschickt werden. Der JSP wird hierbei das Context-Objekt übergeben, so dass bereits gemachte Angaben in der Seite angezeigt und fehlerhafte bzw. nicht gemachte Eingaben mit einem entsprechenden Kommentar versehen werden können (vgl. Abb. 2).

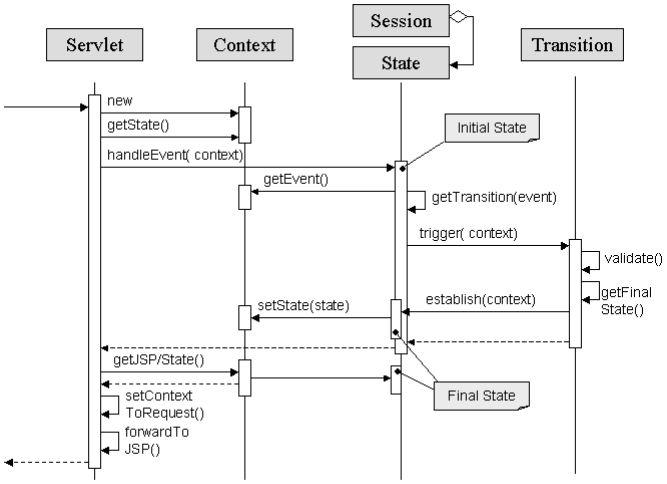


Abb. 3: Das Sequenz-Diagramm zeigt vereinfacht den Ablauf einer Request-Verarbeitung, wenn die Anwendung als Zustandsautomat konzipiert ist. Der aktuelle Zustand wird in der Session verwaltet. Client-Requests entsprechen Ereignissen, die eine Zustandsänderung hervorrufen können. Der aktuelle Zustand erhält das aufgetretene Event zur Verarbeitung. Der Zustand sucht nach der zugehörigen Transition und führt diese aus. Die Transition kann Validierungen vornehmen und verankert den neuen Zustand in der Session.

2.2 Die Web-Applikation als Zustandsautomat

Bei dialogorientierten Abläufen muss der Client oft eine bestimmte Reihenfolge der Seiten durchlaufen. Das Zurückspringen über den Back-Button muss insbesondere bei transaktionalen Abläufen kontrolliert oder in manchen Fällen sogar verhindert werden. Da aber die Funktion des Back-Buttons nicht vom Server aus abgeschaltet werden kann – man kann den Button zwar ausblenden, entsprechende Tastenkombinationen funktionieren aber immer noch – wird eine server-seitige Ablaufsteuerung benötigt. Betrachtet man die Anwendung abstrakt als Zustandsautomat, so kann durch eine einfache Erweiterung des Single-Servlet-Ansatzes diese Anforderung erfüllt werden. In dieser Betrachtungsweise entspricht die Web-Applikation einem Zustandsautomat, der über Client-Requests „Events“ erhält. Je nachdem in welchem Zustand sich die Anwendung befindet, kann auf den Client-Request verschieden reagiert werden. Die

Ablaufkontrolle kann durch die Festlegung erlaubter Übergänge, zumindest innerhalb der Anwendung, vollständig vom Server übernommen werden (vgl. Abb. 3). Die Betätigung des Back-Buttons würde dann z.B. als nicht erlaubter Übergang erkannt und somit vom Server ignoriert werden. Das Caching der Seiten muss für einen korrekten Ablauf allerdings verhindert werden, d.h. Requests müssen immer bis zum Server durchgereicht werden. Durch entsprechende Timeouts für die Session, in der der aktuelle Zustand hinterlegt ist, ist es auch möglich auf das Wegklicken aus der Anwendung und den erneuten Wiedereinstieg in die Applikation zu reagieren.
Man beachte, dass der eigentliche Zustandsautomat, die Web-Applikation, reentrend sein muss. Neben dem oben eingeführten Context-Objekt muss jetzt noch mit einem Zustandsobjekt gearbeitet werden. Das Zustandsobjekt enthält den kompletten aktuellen client-bezogenen Zustand und ist in der User-Session hinterlegt. In den meisten Fällen entspricht der Zustand quasi der gezeigten View (JSP). Der Zustandsautomat kann im wesentlichen voll konfigurierbar gestaltet werden. Hierzu müssen alle Zustände, deren erlaubte Übergänge (Transitions) und die Übergänge hervorrufenden Events hinterlegt werden (vgl. Abb. 4). Erhält die Applikation einen Client-Request (Event zur Zustandsänderung) so wird geprüft, ob der aktuelle Zustand auf das Event reagieren kann. Falls ein Übergang für das eingetroffene Event existiert, wird der Übergang vollzogen. Beim Übergang kann z.B. eine Validierung der Eingaben (pre-condition check) stattfinden. Bei erfolgreicher Validierung geht die Anwendung in den neuen Zustand über und die entsprechende Antwortseite wird an den Client zurückgesendet. Die Transition-Objekte entsprechen im wesentlichen den Command-Objekten bei der vorgestellten server-seitigen Eingabevalidierung. Eine Kombination der beiden Architekturen ist daher sehr einfach möglich.

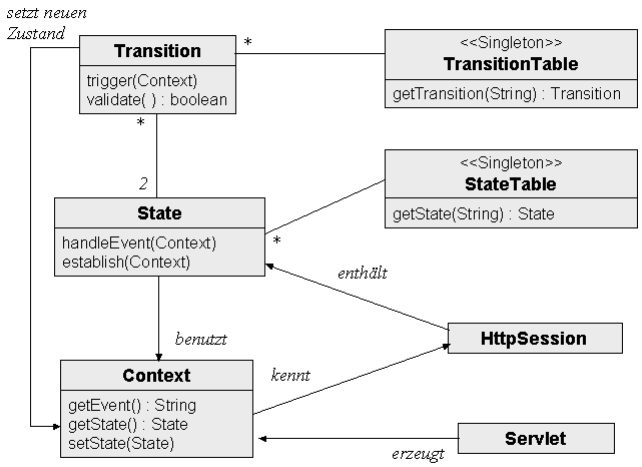


Abb. 4: Beteiligte Klassen zur Modellierung eines konfigurierbaren Zustandsautomaten. Zustände und mögliche Übergänge sind in zentralen Klassen hinterlegt. Eine Transition kennt seinen Anfangszustand und seinen Endzustand. Ein Zustand kennt die von ihm ausgehenden Übergänge und die zugehörigen auslösenden Events.

3. Zusammenfassung

In dem Beitrag wurden allgemeingültige, auf der J2EE basierende Architekturen zur server-seitigen Eingabevalidierung und zur server-seitigen Client-Steuerung vorgestellt. Viele der im Umfeld der Web-Applikations-Entwicklung entstandenen architekturbasierten Lösungen wurden mittlerweile auch in Frameworkentwicklungen aufgegriffen. Die hier vorgestellten Lösung für die Eingabevalidierung wird z.B. in ähnlicher Form im Struts Framework der Apache Jakarta Project Group verwendet [5]. Man kann sicherlich davon ausgehen, das sich in naher Zukunft im Bereich der Web-Anwendungsentwicklung aus den heute verwendeten „Best Practices“ standardisierte Patterns herausbilden werden.

Literatur

[1] AVEDAL, KARL ET AL.: *Professional JSP*, Wrox Press Ltd, 2000

[2] DAI, NACI UND ELLIS, MICHAEL: *Best Practices for Developing Web Applications Using Java Servlets*, Tutorial OOPSLA 2000.

[3] DAVIS, MALCOM: *Struts, an open-source MVC implementation*, IBM developerWorks Feb. 2001

[4] GAMMA ET AL.: *Design Patterns*, Addison Wesley, 1995.

[5] JAKARTA STRUTS HOMEPAGE: http://jakarta.apache.org/struts

[6] ROSSI, G. ET AL: *Patterns for E-commerce applications*, preprint

[7] SCHWABE, D ET AL.: *Improving Web Information Systems with navigational patterns*, International Journal of Computer Networks and Applications, 1999

[8] SUN'S SERVLET SPEZIFIKATION: http://java.sun.com/products/servlet

[9] SUN'S JSP SPEZIFIKATION: http://java.sun.com/products/jsp