J2EE im Vergleich zu XANDRA® Technology - eine Standortbestimmung

von Jürgen Nicolai

V191101/jni/ht

main {GRUPPE} Gesellschaft für Informationsverarbeitung mbH

Stammsitz:
Lange Straße 54
70174 Stuttgart
☎ 0711 / 227 0 225
🖷 0711 / 227 0 496

Niederlassung:
Reudnitzer Straße 13
04103 Leipzig
☎ 0341 / 998 20 06
🖷 0341 / 998 20 07

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument stellt J2EE der XANDRA® Technology gegenüber. Ziel ist es, Gemeinsamkeiten und Unterschiede herauszuarbeiten. Dieses Dokument dient als Grundlage einer technischen Diskussion der beiden Ansätze.

Wenn Sie mit den prinzipiellen Ansätzen von Web-Applikationen und J2EE noch nicht vertraut sind, lesen Sie bitte zuerst ab Kapitel 3: Dort können Sie über eine einfach zu verstehende Einführung näheres über J2EE und Web-Architekturen erfahren. Kehren Sie dann zu diesem Kapitel zurück.

2 J2EE und XANDRA®: Die besondere J2EE-Lösung

XANDRA® Technology verwendet das in den J2EE Dokumenten beschriebene MVC- Muster, nutzt die Rechenleistung des Clients jedoch wesentlich besser aus.

2.1 MVC[1] mit XANDRA®

2.1.1 MVC mit J2EE

Folgende Abbildung zeigt das klassische MVC-Muster, wie es unter J2EE oft Verwendung findet:

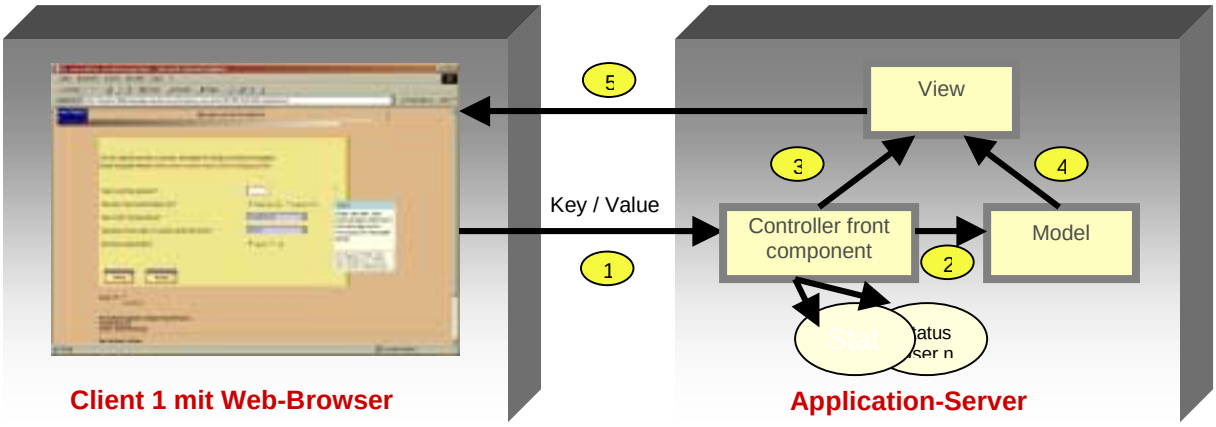

Abb. 1 : Ablauf einer Anfrage nach J2EE und dem MVC-Pattern:

1. Der Browser sendet einen HTTP-Request an ein zentrales „front-component“-Servlet. Bei diesem Aufruf wird ein für diesen Request eindeutiger Identifier mitgegeben. Dieser Identifier ermöglicht es dem Controller, pro User und Session[2] eine „screen-flow-engine“ anzulegen und so immer „zu wissen“, welches die nächste Seite sein soll. Meistens wird die Abfolge der Seiten über ein XML-Dokument festgelegt.

[1] MCV= **M**odel **V**iew **C**ontroller, ein Architekturvorschlag, der in Kapitel 4.2 ausführlich behandelt wird.
[2] Die Session beschreibt das, was der User während einer Sitzung eingibt. Wird der „Zurück-Button“ im Browser betätigt, werden die eingegebenen Daten am Server rekonstruiert. Der Browser macht das nicht „automatisch“, da HTTP eine sogenannte statusloses Protokoll ist, das nach dem : „fire and forget“ Prinzip arbeitet. Der Browser „weiß“ nicht, was auf der vorigen Seite eingegeben wurde. Nähere Informationen in Kapitel 3.

2. Der Controller übergibt die Kontrolle an die Businesslogik, die ihrerseits Änderungen am Model vornimmt. An dieser Stelle können z.B. EJB's oder auch „normale" Java-Objekte verwendet werden.
3. Danach gibt der Controller die Kontrolle an die Ergebnisseite ab. Die Ergebnisseite kennt der Controller aufgrund seiner „screen-flow-engine", die nun auf die Folgeseite verweist. Die Ergebnisseite ist in der Regel eine Java-Serverpage (JSP), die Zugriff auf das Model hat.
4. Die JSP kann benötigte Daten aus dem Model entnehmen und diese Daten in HTML konvertieren.
5. Im letzten Schritt wird der durch die JSP erzeugte HTML-Code an den Browser gesendet, der das Ergebnis anzeigt.

2.1.2 MVC mit J2EE: Vor und Nachteile

Vorteile:

- Trennung zwischen Daten und Präsentation. Jedoch immer noch ca. 20% Coding für die Präsentationsschicht.
- Wesentlich strukturierter als andere in J2EE vorgeschlagenen, sogenannte „web-centric" Modelle[3].
- Hohe Robustheit und Skalierbarkeit, wenn Infrastruktur leistungsfähig ist.
- In vielen Projekten erprobt.
- Viele Frameworks auf Basis MVC verfügbar, z.B. STRUTS von www.apache.org.

Nachteile:

- **Trennung zwischen Präsentation und Businesslogik meist nicht konsequent : Die Tendenz geht zu viel Java-Code in den JSP- Seiten:**
  In der J2EE-Literatur wird immer auf diese Gefahr hingewiesen und auch Verfahren beschrieben, wie man Java-Code in den JSP's vermeidet: Die Praxis zeigt jedoch, dass dies kaum durchzuhalten ist: In fast jedem von uns untersuchten Projekt findet irgendwann, wenn auch nur aufgrund des Termindrucks, eine Mischung von JSP-Code und Java-Code statt. Ein weiterer Hinweis ist die Tatsache, dass fast immer die Entwickler und nicht die Web-Designer die JSP-Seiten erstellen...
- **Sehr serverlastig:**
  Die gesamte Kontrolle (Session, Screen-Flow, Prüfungen) liegt am Server. Hierzu sind viele Server-Kontakte notwendig. Die meisten MVC-Architekturen benötigen aus diesem Grund besonders leistungsfähige Server und eine sehr gute Anbindung an das Internet.
- **Hohe Komplexität:**
  Pro Seite muss eine Menge Java-Source-Code geschrieben werden:
  Meistens werden pro Seite spezielle Java-Klassen entwickelt, die eine Steuerung der Businesslogik übernehmen. Bei Hunderten von Seiten führt dies zu vielen Java-Klassen, die in Abhängigkeit zueinander stehen, die verwaltet und verstanden werden müssen.
- **Relativ komplexes Deployment:**
  J2EE-Anwendungen werden sehr aufwändig auf den Applikation-Server installiert.

[3] Nähere Informationen hierzu im Abschnitt 4

- Hierzu ist unter J2EE eine eigene „Rolle“ vorgesehen, die viel Know-how braucht. Eine einfache XCOPY-Installation gibt es nicht mehr...
- **Hohe Lernkurve:**
  Die Entwickler müssen viele Technologien beherrschen: HTML, JSP, Java Script, Java, XML, DTD, RMI, EJB, Web-Technologie allgemein: “Die ersten Monate geht gar nichts...”
- **“oversized” für kleine und mittlere Projekte:**
  Es gibt i.A. eine Tendenz, auch 5-Seiten-Projekte mit MVC zu erstellen und erst mal Monate lang eine „Basisinfrastruktur“ zu entwickeln. Obwohl in den J2EE-Spezifikationen mehrmals darauf verwiesen wird, dass MVC nicht für alle Projekttypen sinnvoll ist, scheinen viele nach dem Motto „Viel hilft viel“ vorzugehen und jede noch so triviale Anwendung über MVC abzuwickeln.

2.2 MVC mit XANDRA®

XANDRA® Technology verlegt einige **Bestandteile** der MVC-Architektur **vom Server an den Client**. Hierzu wird zu Beginn einer Web-Anwendung ca. 100KB Java Script-Code mit dem XANDRA®-Basis-System in den Web-Browser geladen.

Das XANDRA®-Laufzeit-System umfasst:

- **einen XML-Parser**
- **Zugriff auf “Built-In” Web-Services wie Datenbank, E-mail, Authentifzierung, Termine u.a.**
- **einen Session Manager**
- **eine umfangreiche GUI-Bibliothek**
- **Realtime Tracking-Module[4]**

Nach dem Laden des Basissystems befindet sich der Code im Cache des Browsers und muss nicht mehr neu geladen werden.

2.2.1 Der Controller am Client

Das Basissystem kann nun dazu genutzt werden, um z.B. den Screen-Flow beim Start der Anwendung als XML-Daten in den Browser zu laden und dort über die ganze Session verfügbar zu machen. Damit entfallen die Server-Kontakte beim Ermitteln der nächsten Webseite, ohne die Logik über die HTML-Seiten zu „verstreuen“: Die Logik bleibt zentral in einem XML-File beschrieben. Dieses File wird lediglich durch den Client geladen und ausgewertet.
Ein weiteres Problem, welches mit dieser Technik umgangen wird, ist der wiederholte Serverzugriff im Fall eines Fehlerzustandes. Mit herkömmlicher JSP-Technik werden Eingabedaten und Zustände erst am Server verglichen und die Seite im Fehlerfall erneut aufgebaut. Im Fall von XANDRA® Technology wird der Fehler sofort am Client analysiert und darauf reagiert.

[4] Mit diesem Modul können die Aktivitäten der User bis auf die Ebene von Mausbewegungen analysiert werden. XANDRA® Technology ist weltweit das einzige Framework mit dieser Eigenschaft.

2.2.2 Das Session Management am Client

Beim klassischen MVC-Ansatz wird die Session am Server gehalten. In einem Cluster sind dazu aufwändige Cache-Mechanismen oder ein ständiges Schreiben der Session in eine temporäre Datenbank notwenig. Wenn auf einer Seite „zurück“ navigiert wird, muss der alte Zustand der Webseite aufwändig aus der Session „restauriert“ werden.
XANDRA® Technology hält die **gesamte Session solange wie möglich am Client**. Die Folge: Sie können Tausende Clients gleichzeitig Online halten, ohne den Server mit dem Session Management zu belasten. Erst zu sogenannten „save points“ wird die komplette Session in Form eines SOAP[5]-Commands an den Server geschickt und dort verarbeitet. Dadurch wird eine extreme Skalierbarkeit erreicht, die weit über der von klassischen J2EE Anwendungen liegt. Dies bestätigen auch Microsoft-Entwickler in einem Interview, bei dem es um die Skalierbarkeit einer der am häufigsten frequentierten Webseiten der Welt geht:

> **„Push the state away from middle-tier if possible. You'll get best scalability if the state is on he client**, which makes sense because you have one-to-one mapping…
> Quelle: http://msdn.microsoft.com/library/en-us/dnbda/html/bdadotnetarch10.asp

Das Verfahren der klassischen MVC-Architektur:

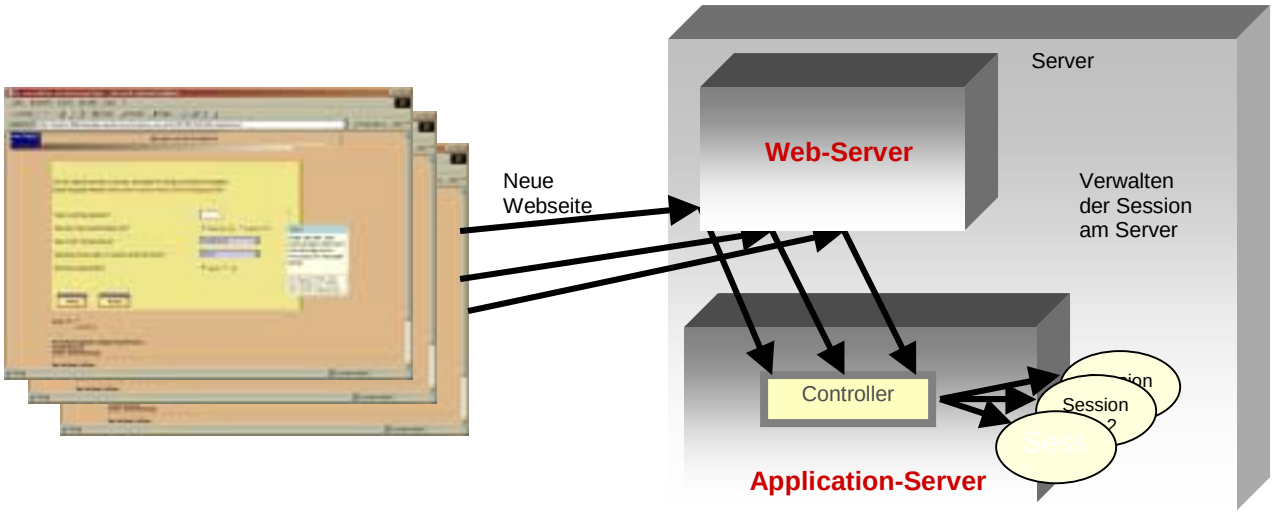

Abb. 2: Mehrere Sessions bei klassischer MVC-Architektur

Die Sessiondaten werden für jeden User über den Applikation-Server verwaltet. Wird eine neue Seite angefordert, muss die Session verändert werden. Dieser Vorgang erfordert eine hohe Rechnleistung und viel Rechenzeit. **Es gibt eine 1:N Zuordnung von Server zu Session**. Das Session-Handling wird in einer Cluster-Umgebung noch wesentlich komplexer, da die Session dann nicht mehr im Arbeitsspeicher eines Servers, sondern im ganzen Cluster verfügbar sein muss.

[5] SOAP: Simple Object Access Protocol. Ein auf XML basierendes Protokoll zum Austausch von Daten. SOAP ist standardisiert und wird von allen führenden Softwareanbietern unterstützt (IBM, Microsoft, SUN u.v.a.)

Bei XANDRA® Technology vereinfacht sich das Verfahren radikal:

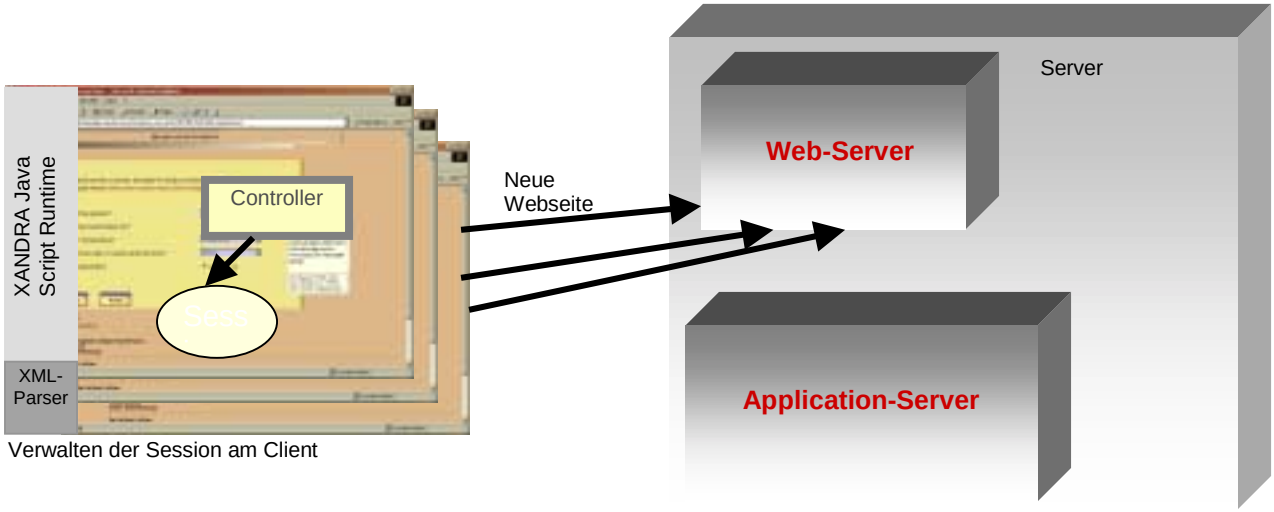

Abb. 3: Mehrere Sessions bei der Verwendung von XANDRA® Technology

Die Session wird pro User im Browser gehalten. **Es gibt eine 1:1 Zuordnung von Browser und Session.** Die Sessiondaten werden erst an den Server geschickt, wenn der User auf einen „Save"-Button drückt oder die Session - aus vom Entwickler festgelegten Punkten - nach dem „fire and forget"-Prinzip über einen Web-Service im Backend verarbeitet werden. Auch in diesem Fall ist die Serverbelastung sehr gering: Die Daten werden z.B. in einer Datenbank gespeichert, jedoch keinerlei Status über diesen Vorgang im Server gehalten[6]. Statuslose Systeme sind extrem skalierbar. Hierzu ein Zitat aus der Fachpresse:

„Systeme, die auf zustandslosen Requests und zustandslosen Services basieren, können eine sehr gute Verfügbarkeit und Skalierbarkeit erzielen"

Quelle: ObjektSpektrum, Dezember 2001, „Hochverfügbare und skalierbare Web/J2EE-Applikationen", Seite 59

XANDRA® Technology verfügt über eine Skalierbarkeit, welche die klassische MVC2-Architektur weit hinter sich lässt.

2.2.3 Die View wird an den Client verlagert

In der klassischen MVC-Architektur erzeugt die JavaServer-Page auf dem Server HTML-Code mit den Daten aus dem Model. Die JSP stellt die ‚View' dar, die für die Aufbereitung der Daten im Browser verantwortlich ist**.** An den Client wird HTML-Code geschickt, der diese Daten anzeigt:

[6] Das entspricht den Konzepten bei den stateless SessionBeans oder den Komponenten des Microsoft Transaction-Servers (MTS): Kein Status am Server führt sowohl nach Aussagen von SUN als auch von Microsoft zur besten Skalierbarkeit im Server-Bereich.

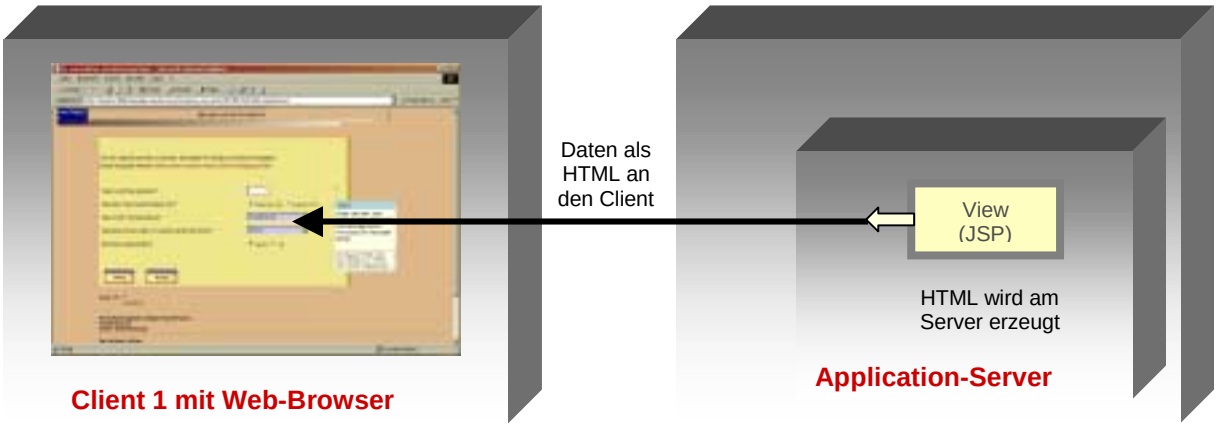

Abb. 4: Die View wird an den Client verlagert.

Die XANDRA®-Architektur verlagert die Aufbereitung des HTML-Codes an den Client. Mit Hilfe von DHTML[7] kann über Java Script HTML-Code im Browser „on the fly“ erzeugt werden. Dadurch wird der Server stark entlastet:

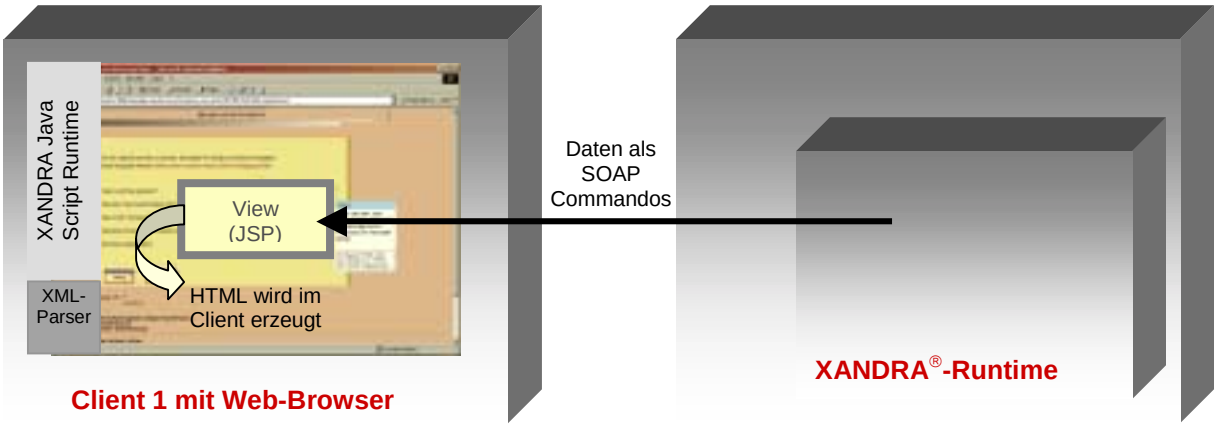

Abb. 5: Aufbau der View mit XANDRA® Technology

Zwischen Server und Client wird XML in Form von SOAP ausgetauscht. **Mit XANDRA® Technology können Web-Services direkt im Browser ohne eine Bearbeitung im Server verwendet werden.** Die Umsetzung von XML nach HTML wird zum großen Teil vollautomatisch durch die XANDRA®-Laufzeitumgebung durchgeführt. Die Vorteile dieses Ansatzes liegen in der erhöhten Skalierbarkeit, da der Server von der Aufgabe der Aufbereitung des HTML-Codes entlastet wird. Sehr viele Anwendungen können mit den XANDRA®-eigenen Web-Services[8] und ohne **eine einzige Zeile Server-Code entwickelt werden**.

[7] Dynamic HTML: Mit Hilfe von Java Script kann man im Browser HTML-Code erzeugen und diesen direkt zur Anzeige bringen. Dies erfolgt komplett im Speicher des Clients und belastet den Server nicht.

[8] Build-In Services sind: Authentifizierung, Datenbankzugriff, Messaging, E-mail, Verschlüsselung, Realtime Tracking, Realtime-Clickstream, PIN/TAN u.a. Mit diesen Services können sehr viele Anwendungen komplett umgesetzt werden: Es wird keinerlei serverseitigen Code mehr benötigt.

Ein weiterer Vorteil ergibt sich durch das einheitliche Programmiermodell und der Tatsache, dass es sich bei SOAP um ein generisches Protokoll handelt. Die Web-Services können in ihrem Verhalten und ihrer internen Implementierung geändert werden, ohne den Client und dessen Arbeitsweise zu ändern. Es muss mit XANDRA® Technology **kein Server-Code angepasst** werden. Beispiel: Am Browser wird ein Eingabeformular mit einem **zusätzlichen Datenfeld** versehen. Es wird lediglich ein Parameter mehr per SOAP an den Web-Service geschickt.

2.2.4 Einfachstes XCOPY- Deployment

Während J2EE-Applikationen sehr aufwändig konfiguriert und installiert werden müssen, bestehen XANDRA®-Applikationen aus wesentlich weniger Dateien, die nicht konfiguriert werden müssen. XANDRA®-Applikationen werden durch einfaches Kopieren installiert und können sehr einfach verteilt werden.

2.2.5 Leichtgewichtige technische Infrastruktur

XANDRA® Technology benötigt keine umfangreiche technische Infrastruktur. **Grund-voraussetzung** ist lediglich ein **Web-Server, eine Java-Runtime-Umgebung** und eine **Servlet-Engine**. Jeder darüber hinausgehende Bedarf wird allein von den zu betreibenden Applikationen bestimmt. Wenn Sie eine Datenbank benötigen, dann werden Sie sich eine installieren oder auf (die meist schon vorhandene) Datenbank zugreifen.

Für komplexe Anwendungen mit einem hohen Bedarf an Integration von verschiedenen Systemen (z.B. typische EAI-Projekte) wird man gegebenenfalls umfangreichere Infrastruktur mit CORBA-Unterstützung, integrierten MOM usw. einsetzen. In diesem Fall arbeitet XANDRA® Technology auch bedenkenlos im Kontext eines Application-Servers.

2.3 30% Code gespart: Eine Login-Maske mit J2EE und XANDRA®

Im folgenden Abschnitt soll gezeigt werden, wie eine einfache Login-Steuerung erzeugt wird: Zum einen nach klassischem MVC-Muster und zum anderen durch XANDRA® mit dem Java Script-Framework[9]. Das Beispiel zeigt, wie stark sich die Entwicklung mit XANDRA® Technology vereinfacht. Für J2EE wird STRUTS als klassisches MVC- Framework verwendet. Die nachfolgende Tabelle zeigt die Übersicht, einige detaillierten Listings entnehmen Sie bitte Anhang 8.2

Bei den Codezeilen wurden die Kommentare z.T. entfernt, die Zeilenzahlen sind nur Näherungswerte. Je nach verwendetem MVC-Framework können sich die Zahlen ändern. Viele MVC-Architekturen tendieren aber zu noch mehr Dateien und höherer Komplexität.

Bereich	**STRUTS**	**XANDRA®**
View	50 Zeilen JSP-Code mit STRUTS-spezifischem Syntax (Tags). Den Source-Code entnehmen Sie bitte Kapitel 8.	190 Zeilen HTML und XANDRA®-spezifischem Java Script-Code; incl. ausgelagerten, wieder verwendbaren Unterprogrammen.
Controller	XML-Datei für den Controller	XML-Datei / **entfällt**[10]
Sessiondaten	ca. 130 Zeilen Java-Code mit STRUTS-spezifischen Aufrufen.[11]	**entfällt**

[9] Alternativ wäre auch die Entwicklung als Applet möglich. Das Ergebnis ist jedoch ähnlich.

[10] Bei einfachen Anwendungen wird der Controller in Java Script in einer zentralen Routine zusammengefasst. Bei komplexen Applikationen wird eine XML-Datei an den Browser geschickt.

[11] Siehe Anhang: LogonForm.java

Steuerung des Businesslogik	ca. 109 Zeilen Java-Code mit STRUTS-spezifischen Aufrufen[12]	**entfällt**
Businessdaten	ca. 150 Zeilen Code mit STRUTS-spezifischen Aufrufen[13]	**entfällt**
Datenbank-anbindung	JDBC/EJB/Connectors: Einige z.T generierte Java-Dateien	Built-In Web-Services **kein zusätzlicher Code**
Installation	Die Anwendung wird mit einem Werkzeug „packetiert“ und muss installiert werden.	Einfaches Kopieren, sog XCOPY-Deployment.

Als Ergebnis ergibt sich selbst bei diesem groben Vergleich:

- eine Code-Ersparnis von ca. 30%
- reduzierte Komplexität durch geringere Anzahl von verwendeten Technologien
- stark vereinfachte Installation
- vereinfachter Zugriff auf „Backend-Systeme“ durch die Verwendung von Web-Services. Viele XANDRA®-Applikationen werden ohne zusätzlichen Code am Server entwickelt.

2.4 Zusammenfassung

Folgende Abbildung zeigt noch einmal zusammenfassend den Ablauf einer XANDRA®-Anfrage:

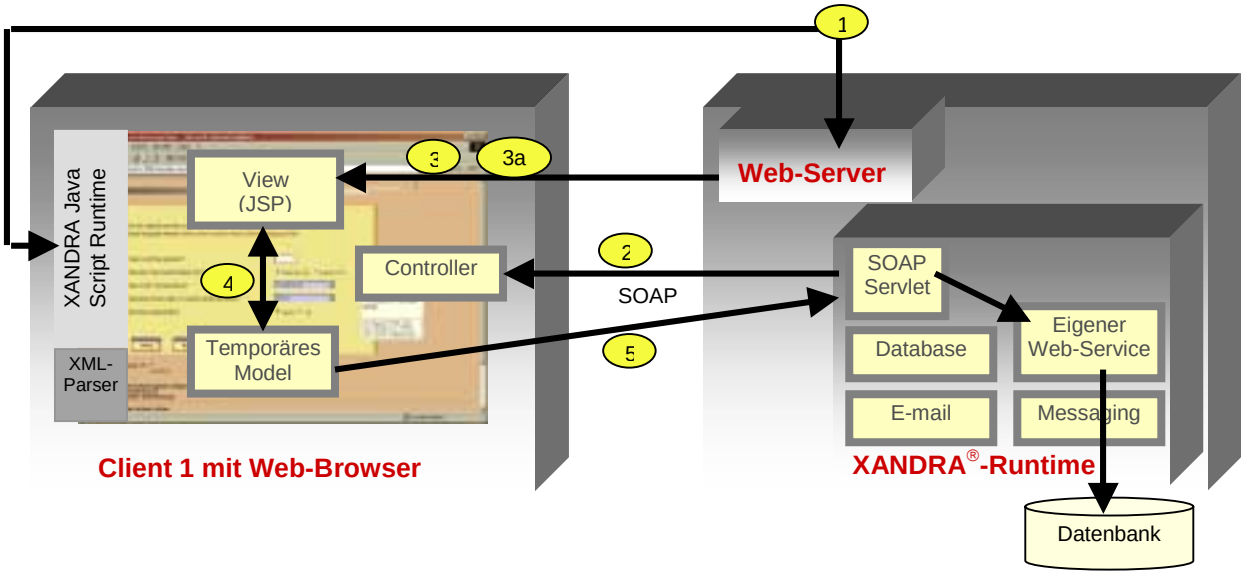

Abb. 6: Ablauf einer XANDRA®-Anfrage.

1. Der Browser lädt die XANDRA®-Client-Runtime als Java Script-Bibliothek.
2. Der Browser lädt den Controller zum Client. Der Controller wird in Form von Java Script oder XML beschrieben.
3. Die View wird vom Controller vom Web-Server geladen und im Browser über DHTML aufgebaut.

[12] Siehe Anhang: LogonAction.java
[13] User.java

4. Die erfassten Daten werden in ein **temporäres Model am Browser** gespeichert. Die nächste View wird vom Controller über den Web-Server geladen: Es gibt keinen weiteren Server-Kontakt, um die Daten der View abzuspeichern. Die Daten werden solange wie möglich am Client gehalten. So können ganze Dialog-Abfolgen durchgeführt werden, ohne Session Management am Server.
5. An sogenannte **„SavePoints**“, die der Entwickler selbst bestimmt, werden die Daten des temporären Modells als SOAP-Call an den entsprechenden Service der XANDRA®-Runtime versendet. Je nach angesprochenem Service werden Backendsysteme genutzt.

2.5 Die Vorteile von XANDRA® Technology

Zusammenfassend ergeben sich folgende Vorteile:

⇒ Eine **extrem hohe Skalierbarkeit** durch Speicherung der Session am Client. Dadurch erfolgt eine 1:1 Zuordnung zwischen Client und Session. Der Server muss nicht tausende von Sessions verwalten. Das System kann am Server komplett zustandslos umgesetzt werden.[14]

⇒ Extrem schnelle Ladezeiten der Seiten.

⇒ J2EE-konform.

⇒ Weltweit das einzige Framework, welches **im Browser** ohne zusätzliche Software mit sogenannte **Web-Services** arbeitet.

⇒ Durch Web-Services entfallen Teile des Server-Codings: Dies hat eine **Reduktion** des **Entwicklungsaufwandes am Server um 30-50%** zur Folge.

⇒ **Umfangreiche Web-Services „Built-In“** verfügbar.
(DB-Zugriff, Authentifizierung, E-mail, Verschlüsselung, Messaging, PIN/TAN, LDAP, Clickstream-Analyse, Realtime Tracking u.a.)

⇒ Reduzierte Komplexität im Server-Bereich

⇒ Kurze Lernkurve: Nach zwei Tagen können erste XANDRA®-Anwendungen erstellt werden.

⇒ Keine Cookies notwendig

⇒ **Realtime Tracking** und **Clickstream**-Analysen bis auf die Ebene von Mausbewegungen; einzigartige Kombination von Clickstream- mit Businessdaten. Für Marketingabteilungen sind diese Daten extrem wertvoll.

⇒ Das Framework arbeitet mit allen **gängigen Application-Servern** zusammen (BEA, Webshere, Inprise, TOMCAT u.a.)

3 Das Umfeld: Welches Problem gilt es zu lösen?

Transaktionen und Geschäftsprozesse über Internet-Technologien abwickeln, ist vereinfacht gesprochen die Vision, die dem Internet diese Dynamik verleiht.

Internet-Technologien sind **die Integrationsplattform**, um

- unterschiedlichste Systeme miteinander zu koppeln

[14] Es klassischen J2EE-Ansatz nur sehr schwer möglich, eine Applikation vollkommen zustandslos zu implementieren. Die Daten müssen im Hauptspeicher gehalten oder permanent in eine Datenbank geschrieben werden, was den Server stark belastet.

- mit dem Endkunden direkt zu kommunizieren
- unterschiedlichste Endgeräte anzusprechen

Internet-Technologien besitzen Eigenschaften, die sie geradezu prädestinieren, für große verteilte Systeme die Basis-Architektur der Wahl zu sein. Es sind z.B.:

- Plattformunabhängigkeit
- keine Installationen notwendig (Thin-Client -Technologie)
- breite Unterstützung aller Software-Hersteller
- gleiche Technologie für Inter- / Intra- und Extranet

Leider wurde die Technologie nicht für diese komplexen Aufgaben geschaffen, sondern als „one-way"-Medium konzipiert, um Informationen „read-only" zur Verfügung zu stellen. Die nächsten Kapitel sollen den Weg vom reinen Informationsmedium zu einem Ersatz für klassische Client / Server-Architekturen aufzeigen:

3.1 Am Anfang war reines HTML...

Das Internet war ursprünglich als „read-only"- Medium konzipiert. Die Ausrichtung der Architektur lag auf:

- einer hohen Anzahl von gleichzeitigen Usern
- Kommunikation über unsichere Netze
- effiziente Verlinkung der Informationen (über Hyperlinks)
- einfache Erstellung der verlinkten Dokumente
- Anzeige der Dokumente in einer einheitlichen Umgebung (dem Browser)

Das Ergebnis ist HTML als Sprache zur Erstellung der Dokumente und HTTP als Protokoll zur Datenübertragung. Dabei gestaltet sich der Ablauf eines Dokuments mit dem Inhalt:

```
<html>
<head>
<title>Der Titel</title>
</head>
<body >
<img src="hello.gif" width=30 height=30 border=0>
</body>
</html>
```

wie folgt:

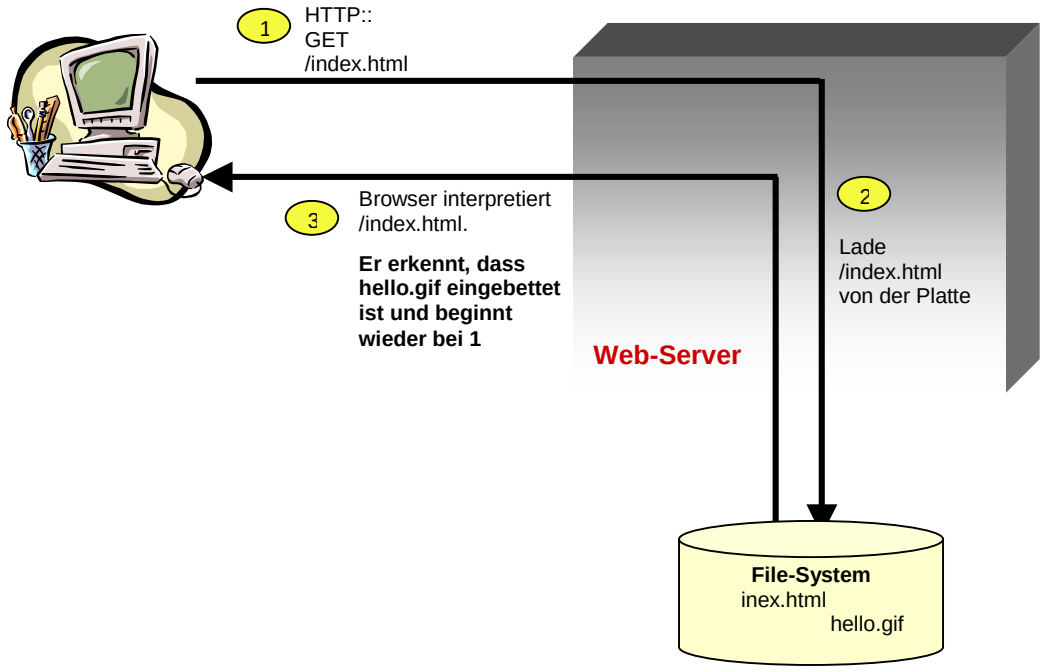

Abb. 7: Der grundlegende Ablauf einer HTTP-Anfrage.

Die hervorstechenden Eigenschaften dieser Architektur sind:

- **Extrem hohe Skalierbarkeit**: Ein einzelner Web-Server kann hunderte User gleichzeitig bedienen.
- **Geringe Komplexität** der Gesamtarchitektur: Im Vergleich zu etablierten Architekturen wie DCOM[15] oder CORBA[16] zeichnet sich die Web-Technologie durch eine erfrischende Einfachheit aus.
- **Äußerst leichte Softwareverteilung**: Browser sind heute Bestandteil jedes Betriebssystems.
- **Vollkommene Plattformunabhängigkeit.**

Diese Einfachheit hat jedoch ihren Preis: Das Gespann HTML und HTTP hat einige Eigenschaften, die den Aufbau von komplexen Systemen erheblich erschweren. Die wichtigsten sind:

- **HTTP ist zustandslos.** Der Web-Server hält keinerlei Informationen, welche Anforderungen logisch zusammengehören. Es gibt keine feste Session, die einem User zuzuordnen wäre. Alleine die Verwendung des „Zurück"-Buttons des Browsers führt dazu, dass die *Daten der Vorgängerseite* nicht korrekt angezeigt werden. Klassische Client / Server-Architekturen ordnen jedem User eine feste Verbindung zu, welche die Programmierung dieser Logik wesentlich vereinfacht. Die „Simulation" dieses Verhaltens in Web-Applikationen führt zu komplexen Architekturen für das Session Management.
- **HTML** stellt eine **Mischung aus Daten und Präsentation** der Daten dar. Dies verletzt architektonische Grundprinzipien, die von einer strikten Trennung von Daten und Präsentation ausgehen.

[15] Distributed Component Object-Model: Ein klassischer Client / Server Ansatz von Microsoft

[16] Component Object Broker: klassischer Client / Server Ansatz.

Fazit:
Die Web-Architektur ist „eigentlich“ nicht dafür entworfen worden, wozu sie heute verwendet wird: Als umfassende Basis-Architektur und Ersatz für klassische Client / Server-Systeme. Die Vorteile der Architektur wiegen diese Nachteile jedoch auf.

Die Nachteile der ursprünglichen Architektur werden über Erweiterungen der ursprünglichen Ansätze ausgeglichen. Davon handeln die folgenden Kapitel...

3.2 Komplexe Anwendungen mit der Web-Architektur.

Das Web bewegte sich sehr schnell vom reinen Anzeigemedium hin zu einem transaktions-orientierten Medium, bei dem Inhalte nicht mehr statisch im Dateisystem, sondern aus Datenbanken oder aus den vorhandenen Systemen erzeugt werden müssen. Es haben sich eine Reihe von Technologien etabliert, die diese Anbindung zu leisten vermögen: CGI[17]-Scripts, ISAPI/NSAPI[18]-DLL’s, ActiveServerPages, (ASP) oder Java-Servlets. Der Ablauf einer Anfrage gestaltet sich in diesem Szenario wie folgt:

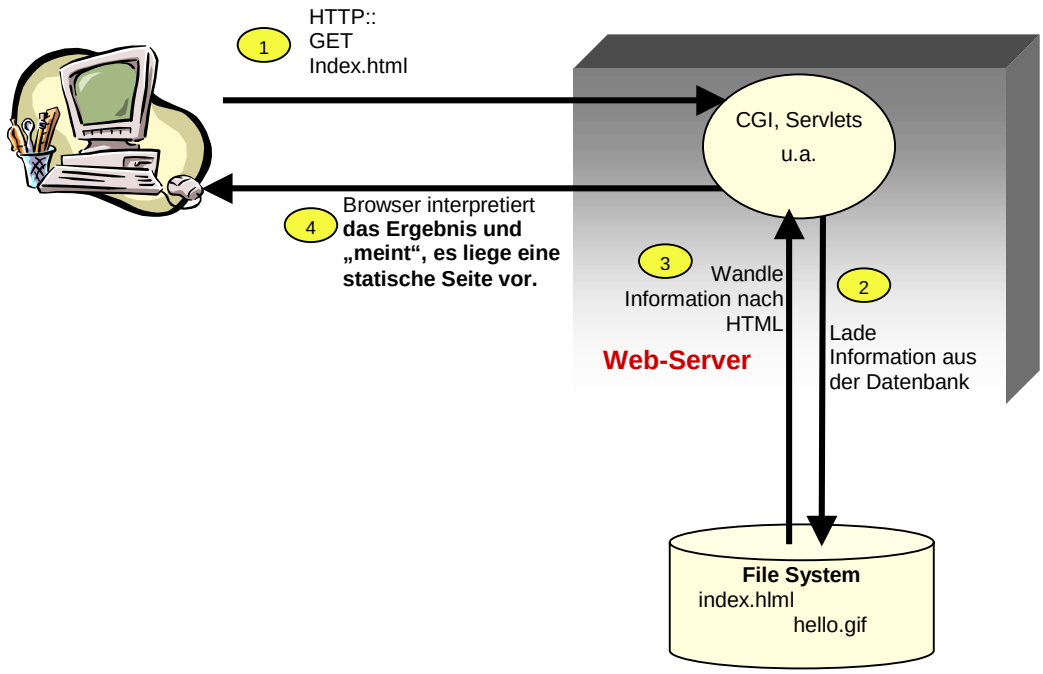

Abb. 8: Ablauf einer Datenbank-Abfrage

1. Der Web-Browser fordert die HTML-Seite von einem CGI-Script an.
2. Das CGI-Script öffnet eine Datenbank und liest die Informationen.
3. Die Informationen werden in HTML umgewandelt.
4. Der HTML-Code wird an den Browser zurückgeschickt. Der Browser „meint“, er hätte eine normal Datei erhalten. Die Komplexität der Datenbankzugriffe wird durch den Server „versteckt“.

[17] Common Gateway Interface
[18] Ähnlich CGI, nur auf Basis von meist in C oder C++ implementierten Modulen.

Der Web-Server fungiert in diesem Szenario zunehmend als **Integrations-Plattform**, um die unterschiedlichsten Datenquellen mit dem Web zu verbinden. Der Web-Server bekommt immer neue, anspruchsvollere Aufgaben, dem einfache CGI-Scripte nicht mehr gewachsen sind[19]: Der Webserver wird **zum Application-Server.**

3.3 Application-Server: Universelle Integrationsplattform

Der einfache Web-Server zu Beginn des Web-Zeitalters hat sich zu einer universellen Plattform zur Integration von Systemen entwickelt. Application-Server haben folgende Aufgaben:

- Unterstützung von vielen Clients gleichzeitig
- Robustheit
- Lastverteilung, Ausfallsicherheit
- Sicherheit
- Unterstützung von Transaktionen
- Verbindung zu Backend-Systemen

Um diese Eigenschaften zu erreichen, sind aufwändige Technologien und Algorithmen notwendig. Diese Technologien wurden in Standard-Architekturen gegossen, damit sich die Entwickler auf die Umsetzung ihrer Geschäftsprozesse konzentrieren können. Eine der bekanntesten Architekturen im Bereich von Web-Applikationen ist die **Java 2 Plattform, Enterprise Edition (J2EE).**

4 Java 2 Plattform, Enterprise Edition (J2EE)

Die Java-Enterprise Edition, auch J2EE genannt, **ist kein Produkt, sondern eine Spezifikation,** die von SUN Microsystems initiiert wurde**.** Sie können J2EE nicht von der Webseite von SUN herunterladen, sondern erhalten umfangreiche Dokumente, die den Standard beschreiben. Das J2EE-Rahmendokument beschreibt auf 159 Seiten eine Reihe von Technologien und Verfahren, wie Web-Anwendungen effizient erstellt werden können. Die Bestandteile der Architektur werden in weiteren z.T. sehr umfangreichen Dokumenten beschrieben.[20]

SUN stellt zwar sog Referenz-Implementierungen zur Verfügung, weist jedoch darauf hin, dass diese Implementierungen keine Produktionsqualität besitzen. Produktionsqualität besitzt das SUN-Produkt I-Planet, welches jedoch nicht frei verfügbar ist.

Hersteller von Application-Servern passen ihre Produkte in zunehmendem Maße an die J2EE-Spezifikation an und müssen eine umfangreiche Test-Suite bei SUN durchlaufen, um das J2EE-Zertifikat zu erhalten.
Die Anforderungen für J2EE-Server sind sehr hoch: Auch große Hersteller wie IBM haben die Zertifizierung für die Websphere–Produktfamilie erst relativ spät erhalten. Inzwischen existieren jedoch sehr viele Produkte mit J2EE-Zertifikat. Eine Liste finden Sie unter 8.1

[19] „Nicht gewachsen“ heißt nicht „unmöglich“. Angenommen: Ein CGI-Script muss für eine Transaktion zwei Datenbanken, eine davon ORACLE, die andere DB2 unter MVS in Form einer CICS-Transaktion modifizieren. Ist dieser Schritt erfolgreich, sollen zwei E-mails an interne Fachabteilungen geschickt werden. Schlägt eine Aktion fehl, müssen alle Änderungen rückgängig gemacht werden: Derartige Anforderungen sind mit CGI sehr aufwändig: Für diese Aufgaben sind Applikation-Server sinnvoll.

[20] JavaServerPages: 158 Seiten, Enterprise Java Beans: > 300 Seiten

J2EE wird über ein spezielles Verfahren weiterentwickelt, den **Java Community Process (JCP)**. Bei diesem Verfahren haben die Hersteller von J2EE-Servern die Möglichkeit, auf die zukünftige Entwicklung Einfluss zu nehmen und den Standard im Hinblick auf den aktuellen Markt weiterzuentwickeln.

Nachdem ein Produkt J2EE-konform geworden ist, sind damit Zahlungen an SUN verbunden. Insofern handelt es sich bei J2EE nicht um einen Standard wie ihn etwa die ISO vergibt, sondern um einen quasi-Standard, hinter der sehr wohl kommerzielle Interessen stehen.

Das Ziel von J2EE wird in [1] wie folgt beschrieben:

"The Java™ 2 Platform, Enterprise Edition (J2EE™) reduces the cost and complexity of developing multitier, enterprise services. J2EE applications can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures."

Zu beachten ist, dass J2EE nicht nur die technische Architektur einer Web-Anwendung beschreibt, sondern auch auf **organisatorische Probleme** wie der Verteilung der Software oder Rollen innerhalb des Entwicklungsteams eingeht. J2EE ist somit mehr als eine technische Architektur.

4.1 J2EE : Der grundlegende architektonische Ansatz

J2EE beschreibt eine klassische n-tier-Architektur und stellt eine Reihe von Technologien zur Verfügung, um den Anforderungen von Web-Anwendungen gerecht zu werden. Die wichtigsten Komponenten sind:

- **Java-Servlets:**
  dienen als Ersatz für die unter 3.2 beschriebenen CGI- Programme.
- **JavaServer-Pages (JSP's):**
  Bauen auf dem Servlet-API auf und ermöglichen eine einfache Generierung von HTML-Code.
- **Enterprise Java Beans (EJB's):**
  Bilden die Geschäftslogik ab. Vereinfachen die Entwicklung von Businesslogik im Application-Server.
- **JDBC / Connectoren:**
  Erlauben den Zugriff auf Backend-Systeme und relationale Datenbanken.

Neben den oben beschriebenen Technologien bietet J2EE noch eine Reihe weiterer Schnittstellen, die bei Internet-Applikationen sinnvoll sein können.

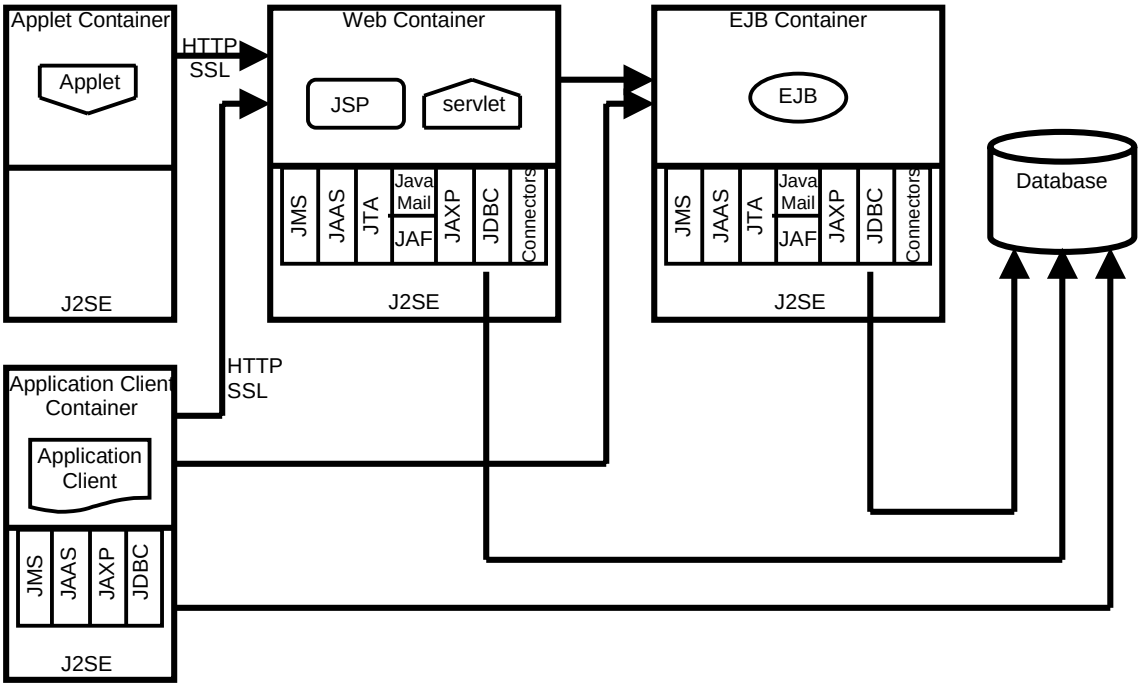

Abb. 9: J2EE-Architektur im Überblick, Quelle: [1]

Selbstverständlich benötigt nicht jede Anwendung all diese Schnittstellen. Dennoch hat sich im J2EE-Umfeld folgender Ablauf einer HTTP-Abfrage herauskristallisiert:

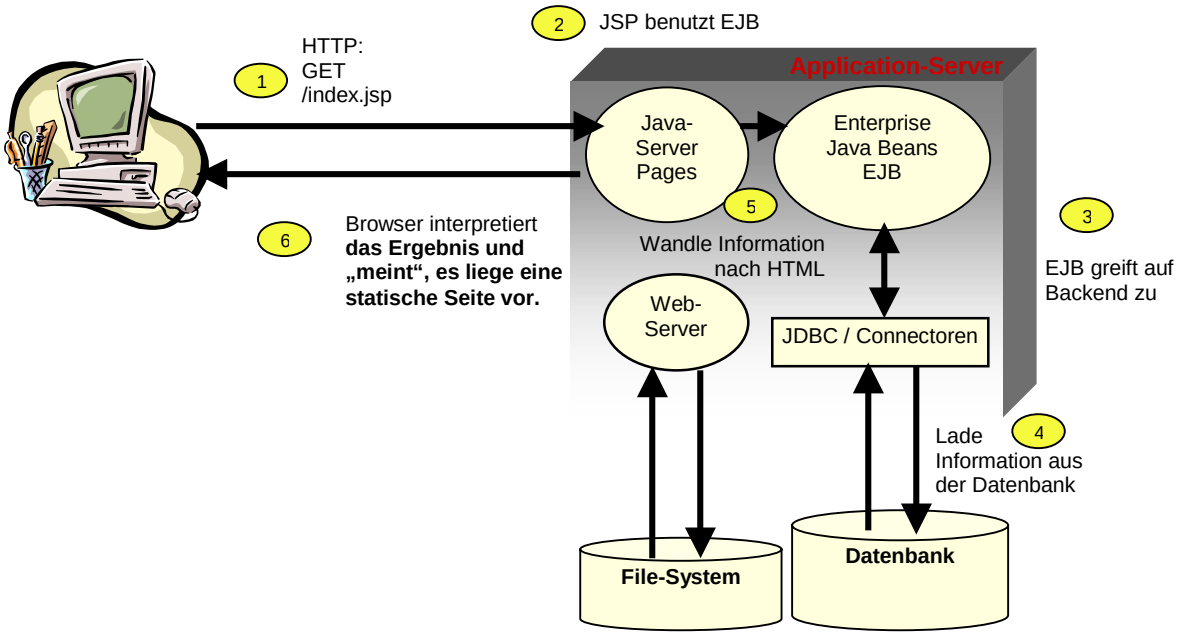

Abb. 10: Eine HTTP-Anfrage mit J2EE.

1. Der Browser stellt eine Anfrage und gibt als Ziel keine HTML-Seite, sondern eine sogenannte **JavaServer-Page (JSP)** an. Eine JavaServer-Page ist ähnlich einer HTML-Seite aufgebaut, kann jedoch Java-Code enthalten und auf weitere Komponenten zugreifen. JSP's stellen die Präsentation der Businessdaten in Form von HTML zur Verfügung.

2. Die JSP arbeitet mit speziellen Java-Komponenten, den **Enterprise Java Beans (EJB)** zusammen. Das EJB kapselt die Businesslogik und sollte eine wiederverwendbare Komponente darstellen. Je nach Aufgabenstellung existieren verschiedene Arten von EJB's: EJB's für den Zugriff auf Datenbanken (entity beans) und solche, die die aktuelle Sitzung widerspiegeln (session beans).
3. Die EJB's greifen auf Backend-Dienste zu. Dies können Datenbanken oder andere Systeme sein. Schnittstellen zu Backend-Systemen sind z.B. **JDBC**, **Java Messaging Service (JMS)** oder die brandneue **Connector-Architektur**.
4. Die Inhalte werden aus dem „Backend" oder der Datenbank geladen.
5. Die JavaServer-Pages wandeln die „nackten" Daten wieder in HTML um.
6. Der Browser empfängt „nomales" HTML und zeigt sie als Seite an.

Der Web-Server liefert in diesem Szenario lediglich die unveränderbaren Inhalte wie Grafiken. Die Tendenz geht jedoch dahin, selbst diese „statischen" Bestandteile „on the fly" aus Datenbanken in Form sogenannter Content-Managment-Systeme (CMS)[21] zu generieren.

4.2 Model-View-Controller und J2EE

J2EE schreibt nicht vor, mit welcher der vorgeschlagenen Technologien eine Anwendung umgesetzt werden soll. So ist z.B. die Verwendung von EJB's bei einfachen Applikationen „oversized". Je nach Komplexität und Anforderungen werden unterschiedliche Vorgehen in [2] empfohlen:

Four general types of Web applications can be implemented with the J2EE platform: basic HTML, HTML with basic JSP pages, JSP pages with Java Beans components, and highly-structured applications that use modular components and enterprise beans. The first three types of applications are considered to be ***Web-centric***, whereas the last type is ***EJB-centric****:*

Dabei wird nicht verschwiegen, dass ein Preis für zunehmende Robustheit und Skalierbarkeit zu zahlen ist: Die steigende Komplexität der Architektur.

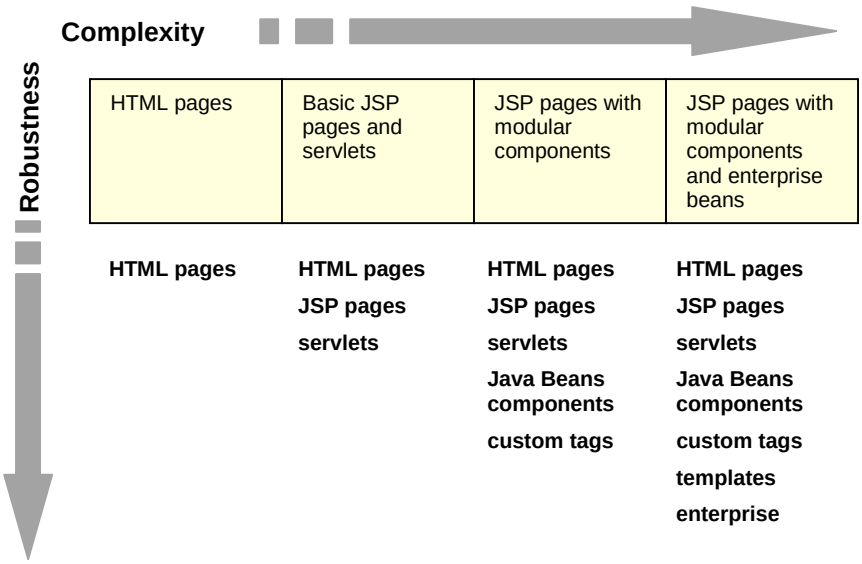

Abb. 11: Architekturansätze unter J2EE, Quelle: SUN, [2]

[21] Ein CMS ist jedoch noch für wesentlich mehr verantwortlich: Freigabe von Texten zu bestimmten Terminen, Vier-Augen-Prinzip u.v.m.

Gerade für die letzte Variante wurden eine Reihe von Mustern[22] entwickelt, um die Komplexität zu reduzieren. Das bekannteste Muster ist das sogenannte **Model-View-Controller**-Muster[23] (MVC), welches zu einer klaren Trennung zwischen Daten und Präsentation der Daten führt.

4.2.1 Das Model

Das Model repräsentiert bei diesem Ansatz die Daten der Applikation und wird mit EJB's implementiert. Das Model hat oft Zugriff auf Datenbanken oder „Legacy-Systeme". Diese Zugriffe können direkt oder auch indirekt über andere Systeme erfolgen (z.B. MOM[24]).

4.2.2 Die View

Die View ist für die Aufbereitung der Daten des Models für einen speziellen Client verantwortlich. Für Web-Anwendungen wird die View durch JSP's implementiert. JSP's haben Zugriff auf das Model und bereiten dessen Daten so auf, dass sie in einem Browser angezeigt werden können. Wichtig ist darauf hinzuweisen, dass in der View keine Businesslogik implementiert werden sollte. Leider zeigt die Praxis, dass dieser Anspruch in den seltensten Fällen eingehalten wird.

4.2.3 Der Controller

Der Controller stellt die zentrale Steuerung der Applikation dar. Meistens werden Anfragen der Clients immer zuerst über den Controller geleitet, der die weitere Steuerung der Anwendung übernimmt. In vielen Fällen verwendet der Controller eine „state-engine", die den Ablauf der Anwendung (welche Seite folgt auf die nächste?) an einer zentralen Stelle festlegt. Die „wilde Verlinkung" von Webseiten wird damit unterbunden.
Der Controller wird oft als **front component** bezeichnet, da er die einzigste Schnittstelle zum Browser hin darstellt[25].

Die folgende Abbildung zeigt den prinzipiellen Aufbau:

[22] Muster oder Pattern: Eine Beschreibung, wie man immer wiederkehrende Probleme am Besten löst. Bekannte Muster sind die GoF-Muster von Gamma.

[23] MVC stammt eigentlich aus der Smalltalk-Welt. Um es gegen das ursprüngliche Muster abzugrenzen, wird es manchmal auch als MVC2 bezeichnet.

[24] MOM= Message Oriented Middleware: Ein Verfahren, bei dem Informationen nicht synchron, sondern asychron versendet werden. Man schickt Informationen weg, schalten den Rechner aus, schalten ihn 2 Wochen später wieder an und erhält die Antwort auf die Nachricht: Das ist MOM...

[25] Der Controller wird i.R. als Servlet implementiert. Alle Anfragen werden an dieses Servlet gesendet und eine eindeutige „Steuer-ID" mitgegeben. Das Servlet kann aufgrund dieser Informationen die nächste Seite bestimmen, Businesslogik ausführen und dann diese Seite aufrufen.

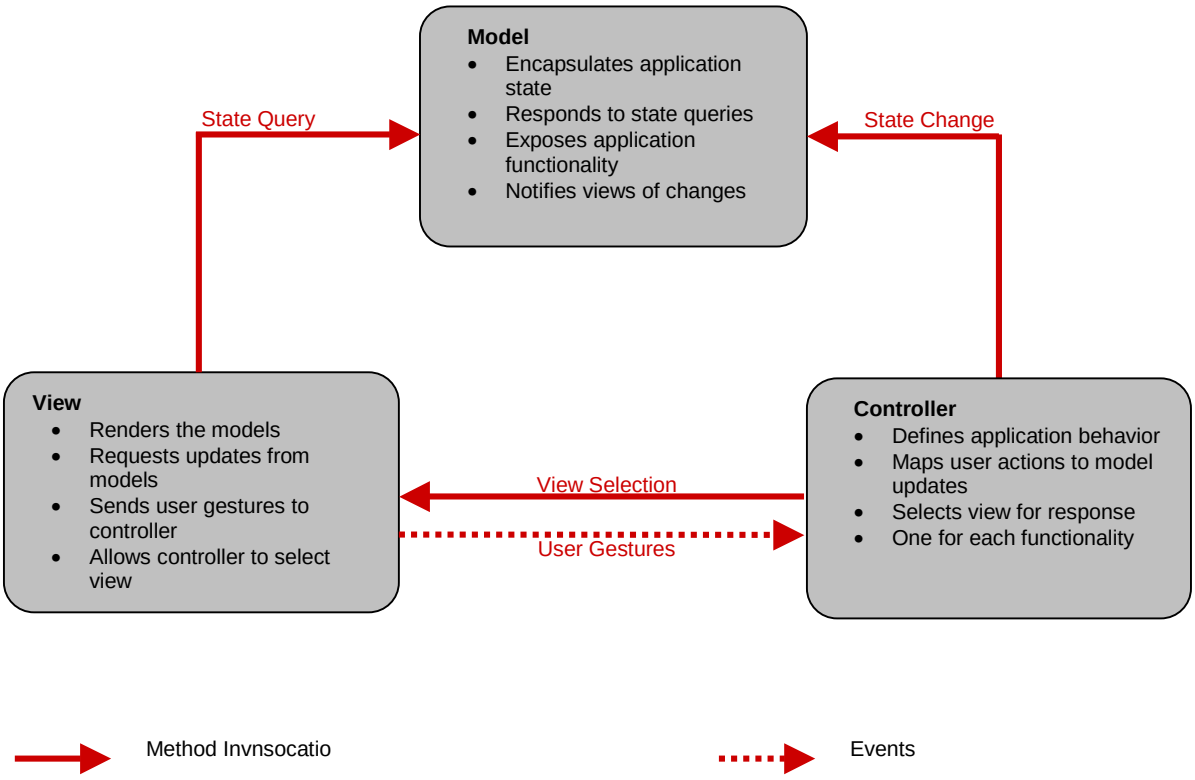

Abb.12: MVC, prinzipieller Aufbau. Quelle: SUN, [2]

In den Unterlagen von SUN wird darauf hingewiesen, dass gerade das MVC-Modell viele Vorteile besitzt, weshalb es zum bevorzugten Ansatz in Verbindung mit J2EE-Architekuren geworden ist:

The MVC architecture allows for a clean separation of business logic, data, and presentation logic. This design also enables content providers and application developers to focus on what they do best. Quelle: [2]

4.3 Frameworks auf Basis des Model-View-Controller-Modells

Am Markt existieren eine Reihe von Frameworks, die exakt auf dem MVC-Modell beruhen. Ein bekannter Vertreter ist das OpenSource-Projekt STRUTS. Die grundlegende Architektur lässt den MVC-Ansatz erkennen, wie er unter [2] beschrieben ist.

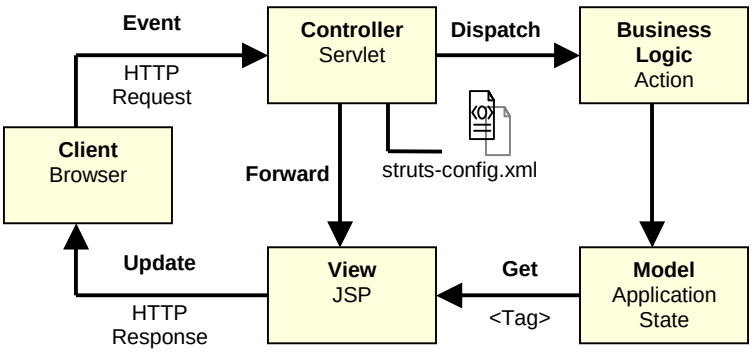

Abb. 13: Architektur von STRUTS.

STRUTS verwendet massiv sogenannte „Customs Tags“, das sind Frameworkspezifische Erweiterungen in JSP-Seiten. Diese Tags erlauben die Kapselung von Businesslogik in einer einzigen Anweisung. Mit STRUTS wird eine umfangreiche Bibliothek dieser Tags mitgeliefert. Customs Tags werden über Java-Code implementiert. Weiterhin steuert bei STRUTS der Controller den Ablauf der Applikation über ein XML[26]-File mit dem Namen struts-config.xml.

Der nachfolgende Ausschnitt vermittelt Ihnen einen Eindruck, wie STRUTS-JSP-Seiten aufgebaut sind. STRUTS-Spezifika sind fett hervorgehoben:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts.tld" prefix="struts" %>
<%@ taglib uri="/WEB-INF/struts-form.tld" prefix="form" %>
<html>
<head>
<title><struts:message key="join.title"/></title>
</head>
<body bgcolor="white">
<form:errors/>
<h3>Enter your email to join the group</h3>
<form:form action="join.do" focus="email" >
    <form:text   property="email" size="30" maxlength="30"/>
    <form:submit property="submit" value="Submit"/>
</form:form>
</body>
</html>
```

4.3.1 Ablauf einer HTTP-Anfrage nach dem MVC-Modell

Der prinzipielle Ablauf einer Abfrage ist bei fast allen MVC-Frameworks gleich. Folgende Abbildung zeigt den Vorgang am Beispiel von STRUTS:

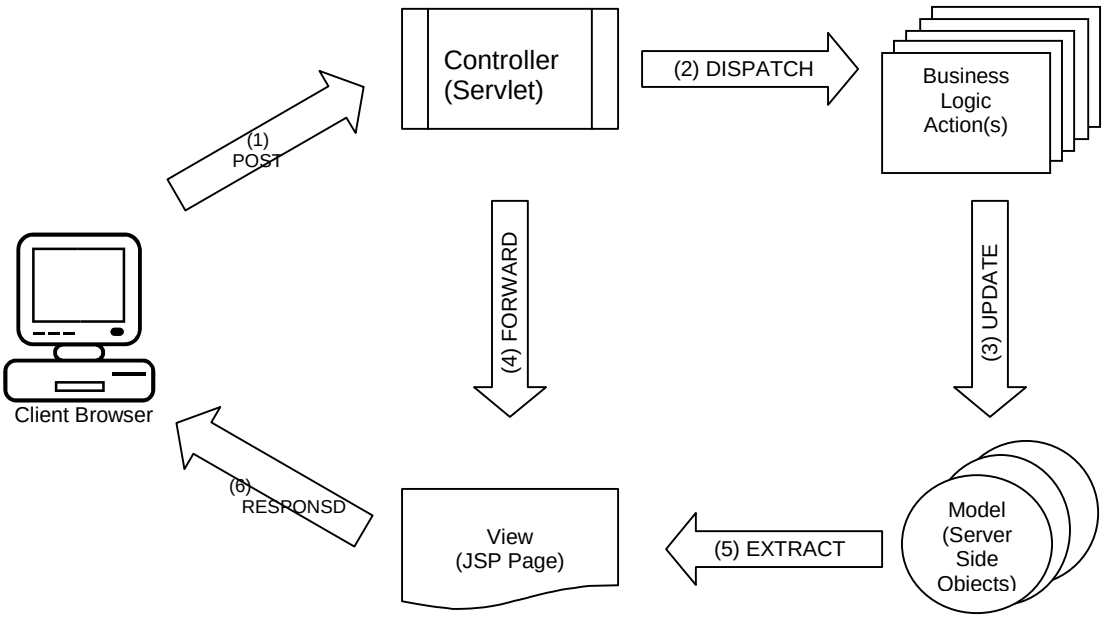

Abb. 14: Ablauf einer HTTP-Anfrage bei STRUTS, Quelle: [4]

[26] Extensible Markup Languge (XML): Eine Beschreibungssprache für komplexe Strukturen aller Art, die sich zunehmend als Standard durchsetzt.

1. Der Browser sendet einen HTTP-Request an ein zentrales „front-component“-Servlet. Bei diesem Aufruf wird ein für diesen Request eindeutiger Identifier mitgegeben. Dieser Identifier ermöglicht es dem Controller, pro User und Session eine „state-engine“ anzulegen und so immer „zu wissen“, welches die nächste Seite sein soll. Meistens wird die Abfolge der Seiten über ein XML-Dokument festgelegt.
2. Im nächsten Schritt übergibt der Controller die Kontrolle an die Businesslogik ab.
3. Die Businesslogik führt Änderungen am Model durch. An dieser Stelle können z.B. EJB’s oder auch „normale“ Java-Objekte verwendet werden.
4. Danach gibt der Controller die Kontrolle an die Ergebnisseite ab. Die Ergebnisseite kennt der Controller aufgrund seiner „state-engine“, die nun auf die Folgeseite verweist.
5. Die Ergebnisseite ist in der Regel eine Java-Server-Page (JSP) die Zugriff auf das Model hat. Die JSP kann benötigte Daten aus dem Model entnehmen und diese in HTML konvertieren.
6. Im letzten Schritt wird der durch die JSP erzeugte HTML-Code an den Browser versendet, der das Ergebnis anzeigt.
7. Über die Ergebnisseite kann nun der nächste Request an den Controller abgesetzt werden und das Spiel beginnt bei Punkt 1 neu.

Es soll an dieser Stelle nicht verschwiegen werden, dass eine derartige Architektur recht komplex ist. Sie ist schwierig zu verstehen und zu debuggen. Die Entwickler müssen eine ganze Reihe neuer Technologien erlernen. Die Lernphase ist relativ lange.

J2EE ist in dieser Ausprägung sehr weit von dem einfachen Modell entfernt, welches in Kapitel 3.1 beschrieben wurde. In den J2EE-Dokumenten wird auf diese Tatsache hingewiesen und auch einfachere Alternativen vorgeschlagen.

4.4 Bewertung von J2EE

Vorteile:

- J2EE definiert den Aufbau von Web-Applikationen sehr detailliert und gibt wertvolle Hinweise bezüglich technischen und organisatorischen Fragen.
- breite Unterstützung von den „Größen“ der IT-Branche (SUN,IBM, ORACLE u.v.m.)
- Umfangreiche Tool-Unterstützung; viele J2EE-Server verfügbar.
- J2EE-Server werden zunehmend robuster und leistungsfähiger.
- Plattformunabhängigkeit durch die Verwendung von Java.
- Viel Know-how und Literatur verfügbar.
- In der Praxis erprobt.

Nachteile:

- Konzentration auf Java als Implementierungs-Plattform: Die Einbindung „fremdsprachiger“ Systeme ist immer noch relativ aufwändig.
- (Noch) schlechte Unterstützung von Web-Services und XML.

- Tendenz zur Komplexität: Auch einfache Anwendungen werden z.T. mit zu komplexem architektonischen Überbau entwickelt. Die Lernkurve für die Entwickler ist sehr hoch.
- J2EE ist als Web-Architektur sehr serverlastig. Die Rechenleistung der Clients wird nicht genutzt. So werden z.B. Feldprüfungen oft am Server durchgeführt, obwohl dies auch am Client möglich wäre.
- J2EE definiert einige wichtige Aspekte nicht. In diesen Bereichen ist J2EE nicht plattform- und herstellerunabhängig (z.B. die Konfiguration der Application-Server, Clustering, Clickstream-Analysen, Hochverfügbarkeit, Überwachung u.a.). Der Übergang von einer Plattform auf eine andere ist bei weitem nicht so unproblematisch, wie es die Marketing-Strategien der J2EE-Verfechter den Kunden weismachen wollen.
- Keine Unterstützung durch Microsoft.
- Kommerzielle J2EE-Server sind relativ teuer. OpenSource-J2EE-Implementierungen kommen zunehmend unter Druck, da SUN Lizenzgebühren verlangt[27].
- Keine durchgängige Unterstützung auf Großrechnersystemen, obwohl gerade der Host die Rolle des robusten und skalierbaren Web-Servers übernehmen könnte.

5 Web-Services

J2EE ist sehr Java-lastig. Alle Komponenten setzten auf Java als Implementierungssprache. Die Kommunikation zwischen den Komponenten erfolgt über Java. Das Prinzip „Java everywhere“ entspricht aber nur teilweise den Erfordernissen bei der Implementierung von Internet-Anwendungen, bei der Dienste unterschiedlichster Hersteller zu einer neuen Anwendung zusammengefügt werden müssen.

Web-Services sollen an dieser Stelle einen Fortschritt bringen und sind in [3] so definiert:

“A web service is an application that accepts requests from other systems **across the Internet** or an Intranet, **mediated by lightweight, vendor-neutral communications technologies**. These communications technologies allow any network-enabled systems to interact.”

Web-Services haben folgende, grundlegende Eigenschaften:

- Web-Services sind über das Internet ansprechbar.
- Sie kommunizieren über ein einfaches, standardisiertes Protokoll.
- Sie definieren keine Sprache oder Architektur (z.B. J2EE[28]).
- Einfache Web-Services können zu komplexen Diensten kombiniert werden.

Beispiel:
Der Betreiber eines Internet-Shops könnte folgende Services verwenden, um einen Geschäftsprozess vollständig abzuwickeln:

⇒ den Passport-Service von Microsoft für die Authentifizierung (sicher kein J2EE)
⇒ den Kreditkartenservice von VISA (C++ und Cobol)
⇒ den Frachtverfolgungs-Service der Post (J2EE)

[27] Angeblich wird Enhydra, einer der besten OpenSource-J2EE-Applikation-Server, aus diesem Grund nicht mehr über OpenSource verfügbar sein: Siehe hierzu JavaSpektrum Okt.2001
[28] Damit hat auch Microsoft Interesse...

Dabei können die Services in jeder Web-Service-fähigen Umgebung implementiert sein: Lediglich die Schnittstelle nach außen muss den Spezifikationen der Web-Services genügen.

5.1 Wie Web-Services veröffentlicht werden

Damit obiges Szenario Wirklichkeit werden kann, muss der Entwickler natürlich wissen, dass die Post einen Web-Service zur Frachtverfolgung anbietet. Diese Information kann über im Web verfügbare Verzeichnisse gefunden werden. Diese Verzeichnisse werden über das sogenannte **Universal Description, Discovery and Integration (UDDI)** Konzept angesprochen. Man kann sich UDDI als einen Web-Service vorstellen, dessen Aufgabe die Beschreibung von Web-Services ist. UDDI ist der „Naming-Service" für Web-Services: Es beschreibt, welche Services wo gefunden werden können und wie sie anzusprechen sind.[29]

Alle technischen Details eines Web-Services werden über eine spezielle Sprache beschrieben, die **Web Services Description Language (WDSL)**: Das Format der Nachricht, die Art der Übertragung (HTTP, HTTPs etc.) und andere technische Details. WSDL ist eine Art Metasprache für Web-Services.

Folgende Abbildung zeigt den prinzipiellen Ablauf beim Zugriff auf einen Web-Service:

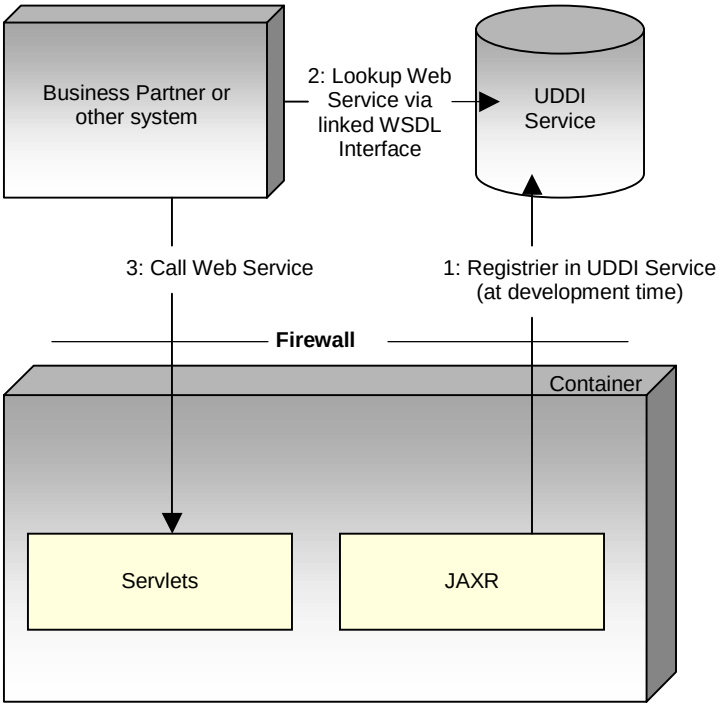

Abb. 15: Zugriff auf einen Web-Service, Quelle: [3]

5.2 Grenzen von Web-Services

Bei komplexen Abfolgen reichen die einfachen Definitionen von Web-Services nicht aus. Workflow-Steuerung, Transaktionsverwaltung, Berechtigungskonzepte sind nur einige Themen, die von weiteren Standards abgedeckt werden sollen: Z.B. *ebXML*, mit dessen Hilfe komplexe B2B-Transaktionen auf Basis von Web-Services definiert werden können.

[29] Insofern ist UDDI eine konsequente Weiterentwicklung klassischer Ansätze: RMI und CORBA bieten Naming-Services als Basis-Dienste an, über die Objekte gefunden werden können.

5.3 Auch für Microsoft: Web-Services sind die Zukunft

Während Microsoft J2EE ignoriert, haben sich die Redmonder voll zum Konzept der Web-Services bekannt:

- Microsoft war maßgeblich an der Entwicklung von Web-Services beteiligt.
- Microsoft ist nach wie vor sehr aktiv in diesem Bereich.
- Microsoft hat konkrete Pläne, erste Web-Services kommerziell anzubieten.
- die neue Plattform .NET[30] besitzt augenblicklich die beste Unterstützung von Web-Services überhaupt.

Folgende Abbildung zeigt einen Dialog aus der neuesten Entwicklungsumgebung von Microsoft: Web-Services sind auf der Ebene von normalen Programmen angesiedelt und nicht etwa ein PlugIn „am hintersten Ende“ der Umgebung:

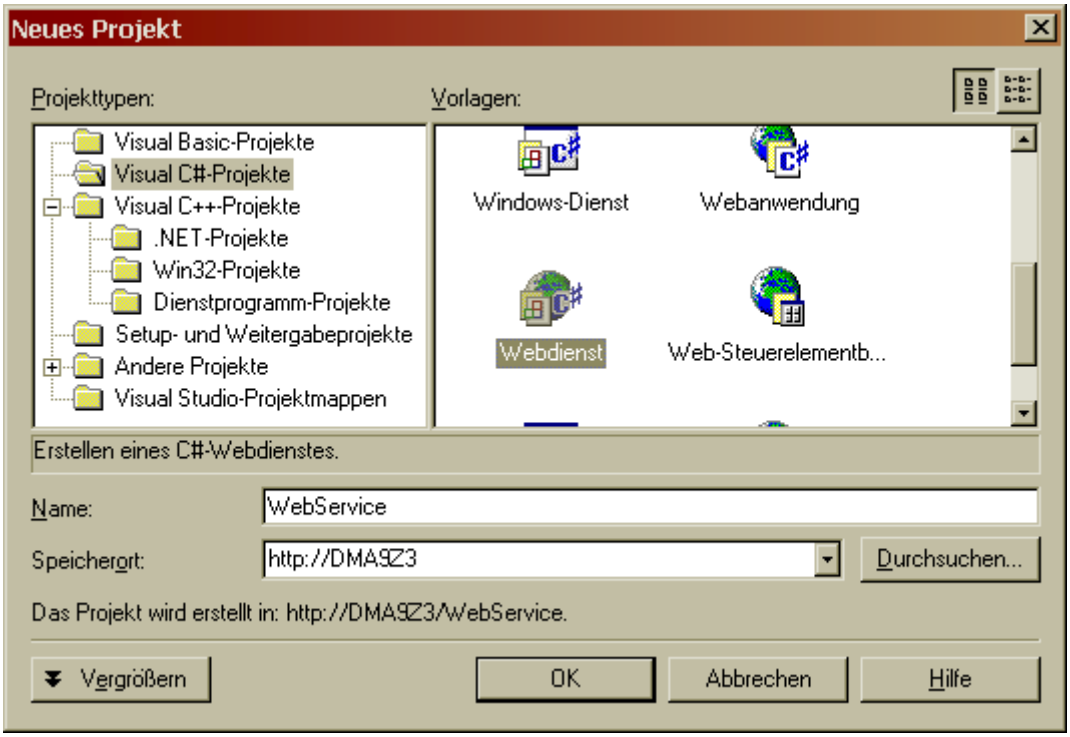

Abb.16: Dialog für Web-Services im neuen Developer-Studio von Microsoft.

5.4 Web-Services und J2EE

Die Bedeutung von XML und Web-Services wurde von SUN relativ spät erkannt. XML-API’s erscheinen erst in der neuesten Version von Java (JDK1.4) und die Unterstützung von Web-Services finden sich in [1] unter „Future Directions“:

“Support for web services is likely to be a primary focus for the next version of J2EE.
A number of JSRs contribute to the definition of web services, including:
• JSR-67 - Java APIs for XML Messaging 1.0 (JAXM)
• JSR-93 - Java API for XML Registries 1.0 (JAXR)
• JSR-101 - Java APIs for XML RPC (JAX-RPC)
• JSR-109 - Implementing Enterprise Web Services
These JSRs can all be found at http://java.sun.com/aboutJava/“

[30] .NET ist eine Plattform ähnlich wie J2EE, die ausschließlich mit Microsoft-Technologien arbeitet.

Diesen Zustand umgehen die Hersteller von J2EE-Entwicklungs-Tools, indem sie proprietäre Unterstützung von Web-Services implementieren. Die Art, wie ein Web-Service entwickelt und betrieben wird, hängt damit stark vom jeweiligen Hersteller der Entwicklungsumgebung ab.

5.5 Bewertung der Web-Services

Die Idee der Web-Services stellt eine konsequente Weiterentwicklung der Komponenten-Technologie auf Basis von Internet-Technologien dar. Alle namhaften Hersteller haben sich zu Web-Services bekannt, weshalb keine technologische Zersplitterung wie etwa bei CORBA und DCOM zu erwarten ist.
Auch wenn viele Fragen - etwa zur Transaktions-Steuerung - noch ungeklärt sind, dürften die Web-Services eine bedeutende Rolle bei der Entwicklung zukünftiger Web-Anwendungen spielen.

6 Literaturverzeichnis

[1] Java™ 2 Platform Enterprise Edition Specification

[2] J2EE Blueprints

[3] Informationen über Web-Services http://www.theserverside.com/resources/articles/WebServices-Dev-Guide/article.html

[4] Struts Dokumentation : www.apache.org

7 Index

8 Anhang

8.1 Liste von J2EE-Servern

BEA Systems	BEA WebLogic
IBM	Websphere
SUN	IPlanet
Inprise	Inprise Application Server
Bluestone	Sapphire/Web
Oracle	Oracle Application Server
Persistence	PowerTier for EJB
Progress	Aptivity, Webspeed
Silverstream	Silverstream Application Server
Valto Systems	Ejipt

Freie Server:

Jboss	www.jboss.org
Enhydra	www.enhydra.org

8.2 Listings

8.2.1 Login-View mit STRUTS

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:html locale="true">
<head>
<title><bean:message key="logon.title"/></title>
<html:base/>
</head>
<body bgcolor="white">

<html:errors/>

<html:form action="/logon" focus="username">
<table border="0" width="100%">

  <tr>
    <th align="right">
      <bean:message key="prompt.username"/>
    </th>
    <td align="left">
      <html:text property="username" size="16" maxlength="16"/>
    </td>
  </tr>

  <tr>
    <th align="right">
      <bean:message key="prompt.password"/>
    </th>
    <td align="left">
      <html:password property="password" size="16" maxlength="16"
                     redisplay="false"/>
    </td>
  </tr>

  <tr>
    <td align="right">
      <html:submit property="submit" value="Submit"/>
    </td>
    <td align="left">
      <html:reset/>
    </td>
  </tr>

</table>

</html:form>

</body>
</html:html>
```

8.2.2 Login-View mit XANDRA®

```
<HTML>
<HEAD>
               <title>Anmelden des Users</title>
               <script language="javascript" SRC="../reuse/002xandra_loader.js"> </script>
               <SCRIPT language="javascript">
               XANDRA.setReuseJSPath("../reuse");
               XANDRA.importCSS("xandra_css",true);
               XANDRA.importCSS(XANDRA.getParamFromTopURL("csspath","user"),false,false);

               XANDRA.importBaseJS();
               XANDRA.importServerJS();
               </script>
               <script language="javascript" SRC="user_scripts.js"> </script>
               <script language="javascript">

               var dlg = new Dialog;
               var zielSeite;
               var quellSeite;
               var bereich;
               var userNachname;
               var nrKunden;
```

```
		function onButtonClick(xEvt,ctrlId,dlg)
		{
			if (ctrlId == "ANMELDEN")
			{
				var userName = dlg.getItemByName("USER_NAME").getData();
				var passWort = dlg.getItemByName("PWORD").getData();
				var login = new XLoginCmd();
				login.checkUser(onCheckUserWasSent,userName,passWort);

			}
		}

		function onCheckUserWasSent(result,context)
		{
			if (result.isValid())
			{

				var list = result.getResult();
				nrKunden = list[0]["KundenNr"];
				tablePersonal (nrKunden);

			}
			else
			{
				alert ("Sie sind noch nicht registriert.\nHolen Sie dies auf der
folgenden Seite nach!");
				var dataMgr=b_getDataMgr();

				var dlgMgr=b_getDialogMgr();
				var targetFr=b_getMainFrame();


				c_gotoURL(quellSeite,dlgMgr,targetFr);
			}
		}



		function run()
		{
			usr_init();
			dlg = new Dialog("DIALOG_LOGIN",8,4);
			usr_initDialog(dlg,"Anmelden Benutzer");
			var dlgItem=null;

		//Laden Dateimanager
			var dataMgr = b_getDataMgr();
			zielSeite = dataMgr.getData ("DATA_ZIEL_SEITE",null);
			quellSeite = dataMgr.getData ("DATA_QUELL_SEITE",null);

			dlgItem = usr_dlgText("t0","Sie sind leider noch nicht
angemeldet.<br>Holen Sie dies durch Eingabe Ihres Usernamens und des Passwortes nach.",null);
			dlgItem.setTDAttribute("colspan=4");
			dlg.addRowCol(0,0,dlgItem);

		//Username
			dlgItem = usr_dlgText("t1","Username");
			dlg.addRowCol(2,0,dlgItem);
			var tt = usr_ToolTip ("Geben Sie bitte Ihren Usernamen an","Hilfe");
			dlgItem = usr_dlgEdit("USER_NAME",20,tt,20);
			dlg.addRowCol(2,1,dlgItem);
			dlg.addRowCol(2,2,new
DialogToolTipConnector(dlgItem,"./image/helpaus.gif"));

		//Passwort
			dlgItem = usr_dlgText("t2","Passwort");
			dlg.addRowCol(3,0,dlgItem);
			var tt = usr_ToolTip ("Geben Sie bitte Ihr Passwort an","Hilfe");
			dlgItem = usr_dlgEdit("PWORD",20,tt,20);
			dlgItem.setIsPassword(true);
			dlg.addRowCol(3,1,dlgItem);
			dlg.addRowCol(3,2,new
DialogToolTipConnector(dlgItem,"./image/helpaus.gif"));

		//Button Anmelden
			dlgItem = new DialogButton("ANMELDEN","Anmelden",null,onButtonClick);
			dlg.addRowCol(6,3,dlgItem);
```

```
                    dlg.invalidate();
                    dlg.show();
                    dlg.close();
                    dlg.setFocusXY(2,1);

                }
                </SCRIPT>
</HEAD>
<BODY onload="run();">
                <DIV ID="DIALOG_LOGIN" NAME="DIALOG_LOGIN" CLASS="xAbs"> </DIV>
                <DIV ID="tipBox" NAME="tipBox" CLASS="tipBoxCSS"> </DIV>
                <DIV ID="EFFECT" NAME="EFFECT" class=xabs > </DIV>
</BODY>
</HTML>
```

8.2.3 Session-Daten und Businesslogik mit STRUTS

```
package org.apache.struts.webapp.example;


import java.io.IOException;
import java.util.Hashtable;
import java.util.Locale;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;
import org.apache.struts.util.MessageResources;



public final class LogonAction extends Action {


    // ----------------------------------------------------------- Public Methods


    public ActionForward perform(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
                throws IOException, ServletException {

                // Extract attributes we will need
                Locale locale = getLocale(request);
                MessageResources messages = getResources();
                User user = null;

                // Validate the request parameters specified by the user
                ActionErrors errors = new ActionErrors();
                String username = ((LogonForm) form).getUsername();
                String password = ((LogonForm) form).getPassword();
                Hashtable database = (Hashtable)
                  servlet.getServletContext().getAttribute(Constants.DATABASE_KEY);
                if (database == null)
            errors.add(ActionErrors.GLOBAL_ERROR,
                       new ActionError("error.database.missing"));
                else {
                    user = (User) database.get(username);
                    if ((user != null) && !user.getPassword().equals(password))
                       user = null;
                    if (user == null)
                 errors.add(ActionErrors.GLOBAL_ERROR,
                            new ActionError("error.password.mismatch"));
```

```
                }

                // Report any errors we have discovered back to the original form
                if (!errors.empty()) {
                    saveErrors(request, errors);
                    return (new ActionForward(mapping.getInput()));
                }

                // Save our logged-in user in the session
                HttpSession session = request.getSession();
                session.setAttribute(Constants.USER_KEY, user);
                if (servlet.getDebug() >= 1)
                    servlet.log("LogonAction: User '" + user.getUsername() +
                                "' logged on in session " + session.getId());

        // Remove the obsolete form bean
                if (mapping.getAttribute() != null) {
            if ("request".equals(mapping.getScope()))
                request.removeAttribute(mapping.getAttribute());
            else
                session.removeAttribute(mapping.getAttribute());
        }

                // Forward control to the specified success URI
                return (mapping.findForward("success"));

    }


}
package org.apache.struts.webapp.example;


import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;


/**
 * Form bean for the user profile page.  This form has the following fields,
 * with default values in square brackets:
 * <ul>
 * <li><b>password</b> - Entered password value
 * <li><b>username</b> - Entered username value
 * </ul>
 *
 * @author Craig R. McClanahan
 * @version $Revision: 1.2 $ $Date: 2001/04/14 12:53:08 $
 */

public final class LogonForm extends ActionForm {


    // ----------------------------------------------------- Instance Variables


    /**
     * The password.
     */
    private String password = null;


    /**
     * The username.
     */
    private String username = null;


    // ------------------------------------------------------------- Properties


    /**
     * Return the password.
     */
    public String getPassword() {
```

```
            return (this.password);

    }


    /**
     * Set the password.
     *
     * @param password The new password
     */
    public void setPassword(String password) {

        this.password = password;

    }


    /**
     * Return the username.
     */
    public String getUsername() {

            return (this.username);

    }


    /**
     * Set the username.
     *
     * @param username The new username
     */
    public void setUsername(String username) {

        this.username = username;

    }


    // ------------------------------------------------------- Public Methods


    /**
     * Reset all properties to their default values.
     *
     * @param mapping The mapping used to select this instance
     * @param request The servlet request we are processing
     */
    public void reset(ActionMapping mapping, HttpServletRequest request) {

        this.password = null;
        this.username = null;

    }


    /**
     * Validate the properties that have been set from this HTTP request,
     * and return an <code>ActionErrors</code> object that encapsulates any
     * validation errors that have been found.  If no errors are found, return
     * <code>null</code> or an <code>ActionErrors</code> object with no
     * recorded error messages.
     *
     * @param mapping The mapping used to select this instance
     * @param request The servlet request we are processing
     */
    public ActionErrors validate(ActionMapping mapping,
                                 HttpServletRequest request) {

        ActionErrors errors = new ActionErrors();
        if ((username == null) || (username.length() < 1))
            errors.add("username", new ActionError("error.username.required"));
        if ((password == null) || (password.length() < 1))
            errors.add("password", new ActionError("error.password.required"));

        return errors;

    }
```

```
}



package org.apache.struts.webapp.example;


import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;


/**
 * Object that represents a registered user of the mail reader application.
 *
 * @author Craig R. McClanahan
 * @version $Revision: 1.2 $ $Date: 2001/04/14 12:53:08 $
 */

public final class User implements Serializable {


    // ----------------------------------------------- Instance Variables


    /**
     * The EMAIL address from which messages are sent.
     */
    private String fromAddress = null;


    /**
     * The full name of this user, included in from addresses.
     */
    private String fullName = null;


    /**
     * The password (in clear text).
     */
    private String password = null;


    /**
     * The EMAIL address to which replies should be sent.
     */
    private String replyToAddress = null;


    /**
     * The set of Subscriptions associated with this User.
     */
    private Hashtable subscriptions = new Hashtable();


    /**
     * The username (must be unique).
     */
    private String username = null;


    // ------------------------------------------------------------- Properties


    /**
     * Return the from address.
     */
    public String getFromAddress() {

                return (this.fromAddress);

    }
```

```
    /**
     * Set the from address.
     *
     * @param fromAddress The new from address
     */
    public void setFromAddress(String fromAddress) {

                this.fromAddress = fromAddress;

    }



    /**
     * Return the full name.
     */
    public String getFullName() {

                return (this.fullName);

    }


    /**
     * Set the full name.
     *
     * @param fullName The new full name
     */
    public void setFullName(String fullName) {

                this.fullName = fullName;

    }


    /**
     * Return the password.
     */
    public String getPassword() {

                return (this.password);

    }


    /**
     * Set the password.
     *
     * @param password The new password
     */
    public void setPassword(String password) {

                this.password = password;

    }
}
```