WINCOR NIXDORF

Experience meets Vision

Jakarta Struts

Banking Division

Application Software

WINCOR NIXDORF

Experience meets Vision

Agenda

1. Kontext
2. Einführung - Servlets, JSPs, TagLibs, MVC
3. Web Frameworks und Tools im Java/J2EE-Umfeld
4. Struts
5. WSAD5

Banking Division

Application Software

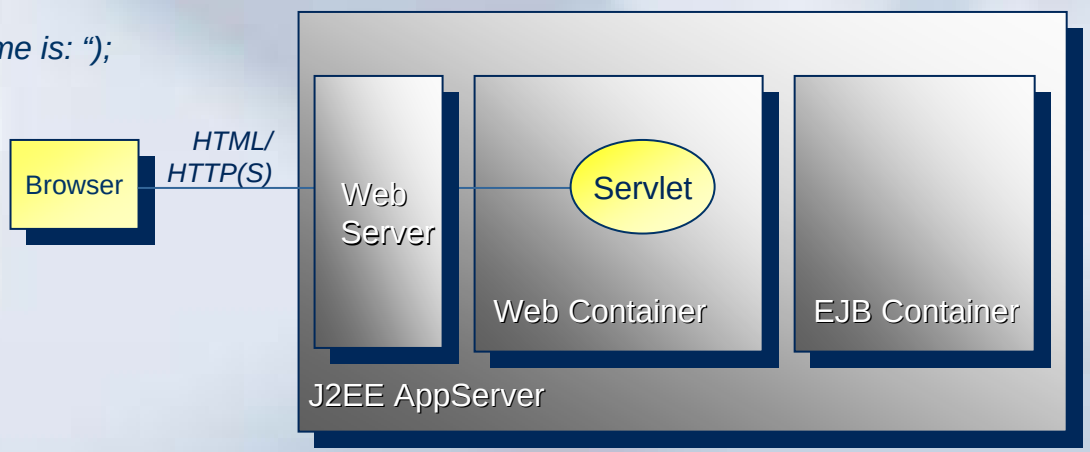
WINCOR NIXDORF

Experience meets Vision

Servlets

- serverseitige Java-Klassen
- Web-Komponenten in Web-Container
- nimmt Requests entgegen, bearbeitet diesen und liefert Response
- Beispiel:

```
public class MyServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res) {
         java.util.Date date = new java.util.Date();
         res.setContentType(„text/html“);
         PrintWriter out = res.getWriter();
         out.println(„<HTML>“);
         out.println(„<HEAD><TITLE> JSP Demo </TITLE></HEAD>“);
         out.println(„<BODY>“);
         out.println(„Hello!  The time is: “);
         out.println(date);
         out.println(„</BODY>“);
         out.println(„</HTML>“);
         out.close();
   }
}
```



Banking Division

Application Software

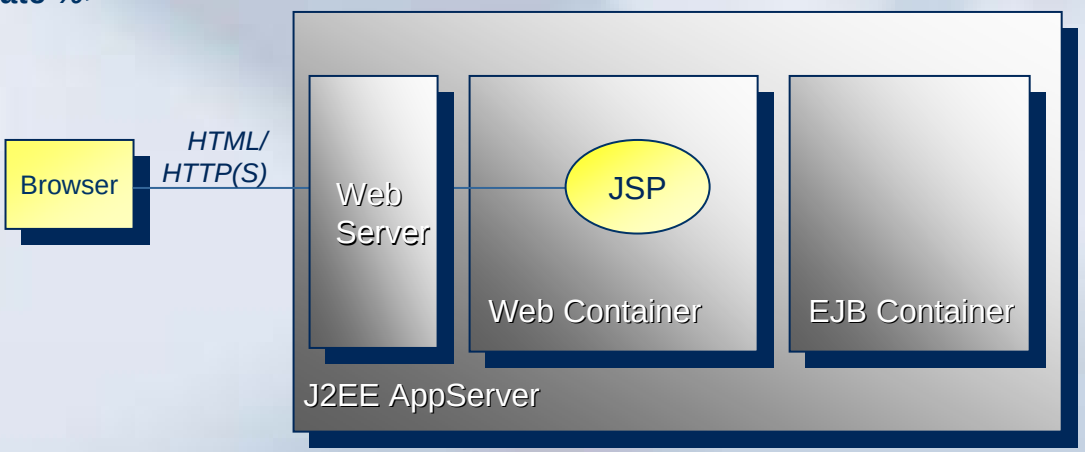
WINCOR NIXDORF

Experience meets Vision

JavaServer Pages (JSPs)

- HTML-Seite mit eingebettetem Java-Code
- Web-Komponenten in Web-Container
- wird vor der Ausführung in Servlet übersetzt
- Beispiel:

```
<HTML>
 <HEAD>
        <TITLE> JSP Demo </TITLE>
 </HEAD>
 <BODY>
        <% java.util.Date date = new java.util.Date(); %>
        Hello!  The time is <%= date %>
 </BODY>
</HTML>
```



Banking Division

Application Software

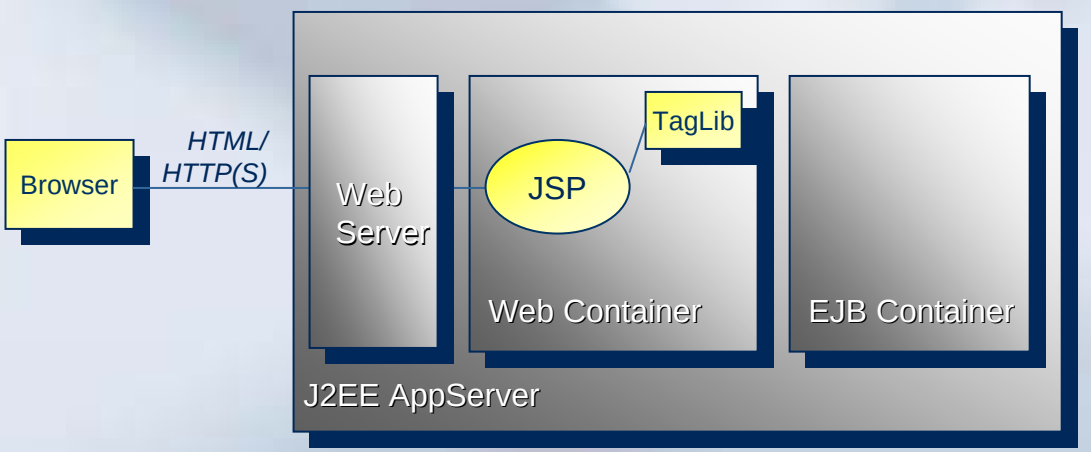
WINCOR NIXDORF

Experience meets Vision

Tag Libraries

- „Herausziehen“ des Java-Codes aus der HTML-Seite in eigene Bibliotheken
- Beispiel:

```
<%@ taglib uri="/WEB-INF/logic-taglib.tld" prefix=„logic" %>
<HTML>
  <HEAD>
        <TITLE> JSP Demo </TITLE>
  </HEAD>
  <BODY>
        Hello!  The time is <logic:date>
  </BODY>
</HTML>
```



Banking Division

Application Software

WINCOR NIXDORF

Experience meets Vision

MVC (Model View Controller)

- Architektur-Muster
- dient zur Trennung der Applikation in:
  - Model: Basisfunktionalität einer Applikation und zugehörige Daten
  - View: Präsentationsschicht
  - Controller: Bearbeitung des Execution-Flows und des Austauschs von Informationen zwischen Model und View
- Vorteile:
  - ´separation of concerns´
  - Teile sind austauschbar
  - Teile sind parallel und spezialisiert bearbeitbar
  - Teile sind wiederverwendbar

MVC Web Architekturen (1)

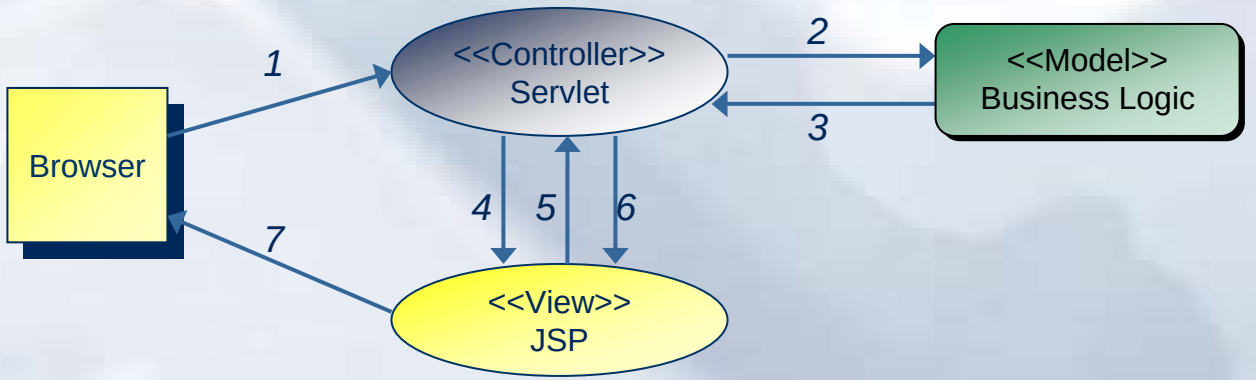
- *Model 1*:
  - JSPs + TagLibs
  - JSP nimmt Request entgegen, ruft Business Logic auf und gibt Response zurück
- *Model 2*:
  - Servlet + JSPs
  - Servlet nimmt Request entgegen, wählt Folge-JSP aus, welche die Business Logic aufruft und sich als Response zurück gibt (´*dispatcher view*´; nur für einfache Szenarien)
  - Servlet nimmt Request entgegen, ruft Business Logic auf, wählt Folge-JSP aus, welche sich die beschafften Daten holt und sich als Response zurückgibt (´*service-to-worker*´)
- *Model 2X*:
  - Servlet + XSL
  - Servlet nimmt Request entgegen, ruft die Business Logic auf, übergibt die dabei beschafften Daten als XML-Dokument an ein XSL-Prozessor, der das entsprechende Format erzeugt

WINCOR NIXDORF

Experience meets Vision

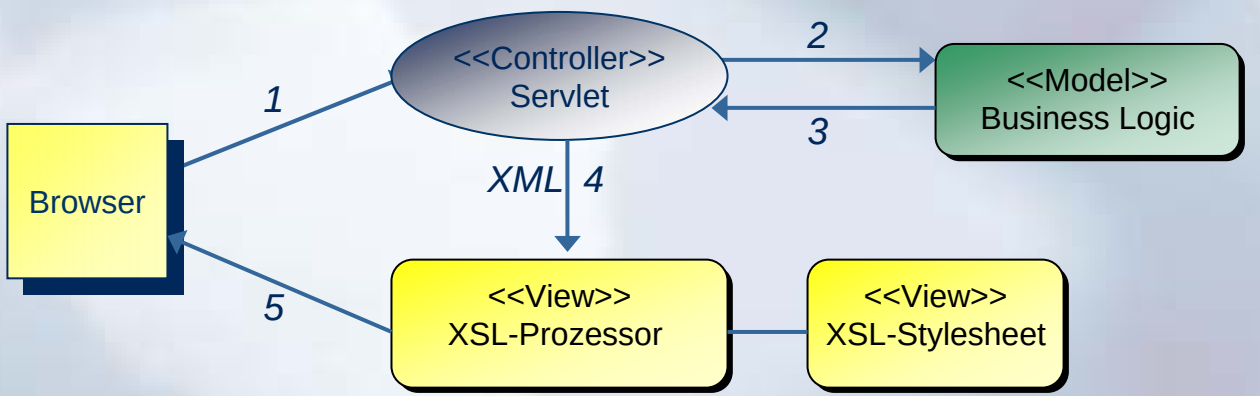
MVC Web Architekturen (2)

Model 1



Model 2

(service-to-worker pattern)



Model 2X

Banking Division

Application Software

WINCOR NIXDORF

Experience meets Vision

Web Frameworks im Java/J2EE-Umfeld (1)

- ´Eigenbau´
- Struts
- Cocoon
- Turbine
- Jetspeed
- Slide
- Hammock
- WingS
- Barracuda
- PowerForms
- JavaServer Faces
- TREND
- Hawk
- Scope
- JATO
- WebWork
- Expresso
- Tapestry
- Freemarker
- WebMacro
- Velocity
- Sitemesh
- Maverick
- IBM Sash Weblications
- StXX
- OXF
- ...

Web Frameworks im Java/J2EE-Umfeld (2)

- Aussichtsreichsten gemäß:
  - der Erfüllung unser Anforderungen
  - der Kosten
  - des Lernaufwand
  - und der Framework-Weiterentwicklung

  sind (nach Recherche):
  - JavaServer Faces <- Model 2
  - **Struts** <- Model 2
  - Struts CX <- Model 2X
  - Turbine <- Model 2
  - Expresso <- Model 2

WINCOR NIXDORF

Experience meets Vision

Evaluierte Entwicklungswerkzeuge

- XML/XSLT-Editoren:
  - *XMLSpy*
  - *Stylevision*
  - *XSelerator*
  - ***Stylus Studio:*** mächtig, übersichtlich, bedienbar)
- JSP/TagLib-Editoren:
  - ***WSAD:*** fundamentale TagLib-Unterstützung
  - *JTagStudio:* nicht installierbar
  - *JTagManager*: nicht installierbar
- Entwicklungsumgebungen für Model2 / Struts:
  - *Adalon:* proprietärer MDA-Ansatz, zu komplex
  - *Struts Console:* nur Basisfunktionalität
  - ***Camino:*** Flow-Design-Unterstützung, keine TagLib-Tools
  - *JBuilder:* keine TagLib-Tools
  - ***WSAD:*** mächtig, übersichtlich, bedienbar, TagLib-Tools
- Entwicklungsumgebungen für Model2X / Struts+XML:
  - ?

WINCOR NIXDORF

Experience meets Vision

Ergebnisse der Evaluation (1)

- ***Struts* (´Model 2´) erfüllt größtenteils unsere Anforderungen, hat aber die bekannten Nachteile der JSP:**
  - keine 100%-ige Trennung zwischen Applikationslogik und Präsentation
  - nicht 100% XML-konform, dadurch keine weitere Aufbereitung in Pipelines
  - proprietäre TagLibs sind zu erlernen
  - gemischte XML- und Text-Konfiguration
- **´Model 2X´ beseitigt die Nachteile der JSP**
- vorhandene ´Model 2X´ Frameworks:
  - *StrutsCX*: zu komplex, nicht Struts-konform
  - *StXX*: nicht Struts-konform, nicht installierbar
  - *OXF*: Struts-konform, proprietär, closed-source, Kosten
  - *Eigenentwicklung*: Struts-konform, proprietär, höherer Aufwand

  => **´Model 2X´-Frameworks sind leider noch nicht reif!**

WINCOR NIXDORF

Experience meets Vision

Ergebnisse der Evaluation (2)

Vorschlag:

- **1. Verwendung von ´Struts´ (Model 2) und Warten auf reifes ´Model 2X´- Framework**
- 2. Ggf. Eigenentwicklung der benötigten ´Model 2X´-Erweiterungen
- 3. Untersuchung: Verwendung von XML-konformen JSPs und Redirect dieser auf eine XML-Pipeline

Struts - Überblick (1)

- ´Model 2´-Framework
- existiert seit Ende 2000
- Open-Source
- ´Marktführer´
- ´lightweight framework´
- einfach zu erlernen
- größter Literatur-Fundus; größter Tool-Fundus
- versprochene Weiterentwicklung mit Orientierung an den Standards (z.B. JSR127 – JavaServer Faces; JSR052 – Java Standard Tag Library)

Struts - Überblick (2)

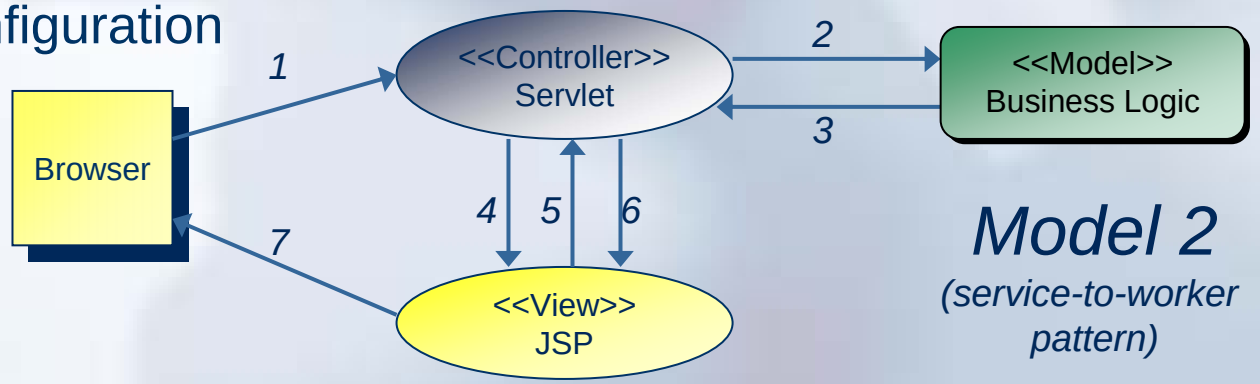
- absolut standardkonform (J2EE komplementär)
- entspricht einem standardkonformen Eigenbau unter Berücksichtigung folgender Patterns/Idioms:
  - MVC
  - Front Controller
  - View Helper
  - Service-to-Worker
- entlastet bei wiederkehrenden ´Web-Aufgaben´ wie:
  - Mapping von HTTP-Parametern auf JavaBeans
  - Validierung
  - Fehleranzeige
  - Internationalisierung
  - URL-Organisation

WINCOR NIXDORF

Experience meets Vision

Statik - Überblick

- *Bestandteile des Frameworks:*
  - Struts-Basis
    - ActionServlet <- Controller
    - TagLibraries
  - Struts-basierte-Applikation
    - ActionForm <- Model
    - Action <- Controller
    - JSP <- View
    - Ressourcen
    - Konfiguration

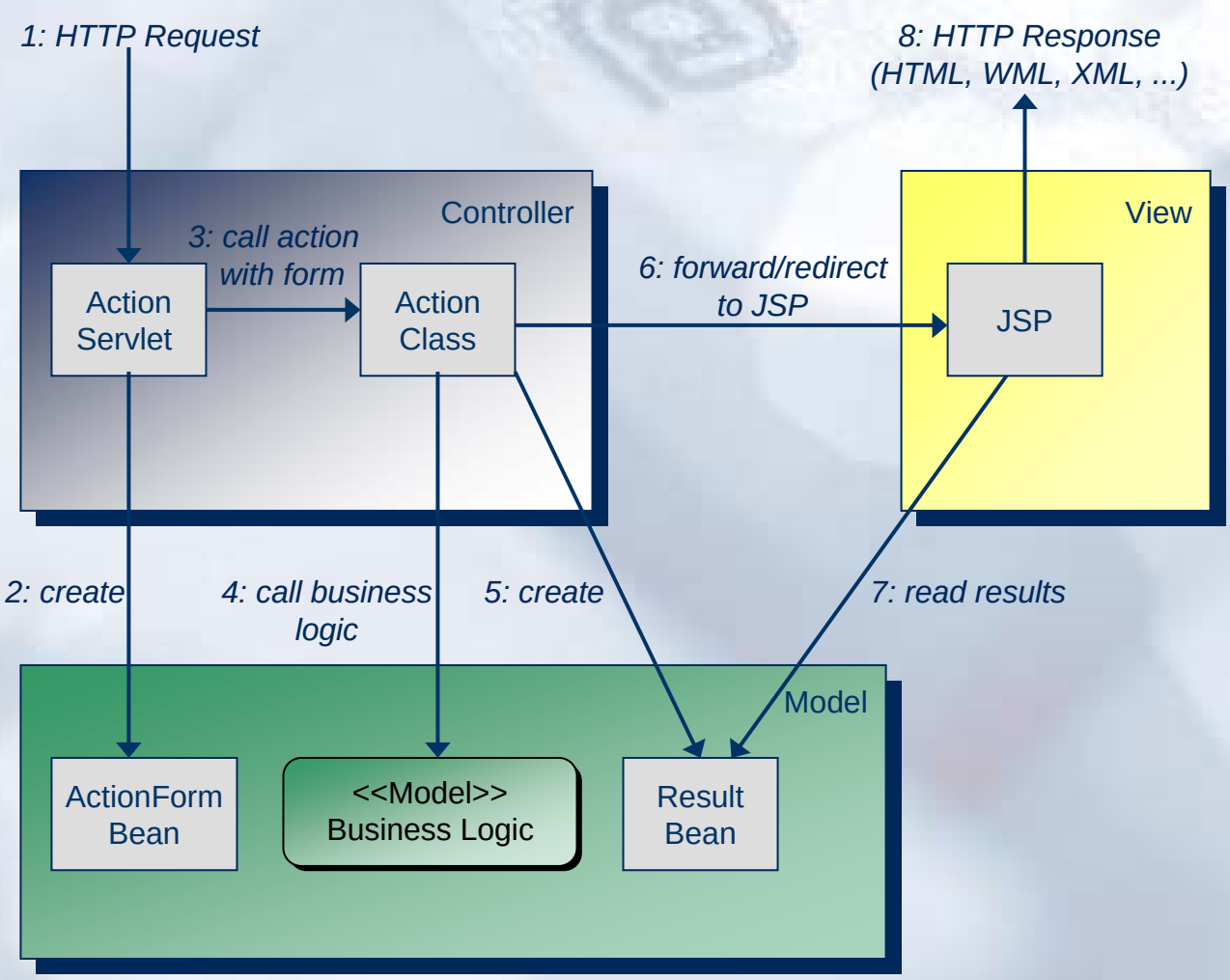


Model 2
(service-to-worker pattern)

WINCOR NIXDORF

Experience meets Vision

Dynamik - Überblick



Banking Division

Application Software

WINCOR NIXDORF

Experience meets Vision

Statik - ActionServlet

- *ActionServlet*:
  - Umfeld:
    - läuft im WebContainer; enthalten im WAR
  - Aufgabe:
    - Front Controller
    - liest applikationsspezifische Konfiguration aus ´*struts-config.xml*´ (´*ActionMapping*´)
    - nimmt alle Requests entgegen, füllt ActionForm-Datencontainer, delegiert Bearbeitung an Action weiter, koordiniert den nächsten View
  - Schnittstellen:
    - ist in der ´*web.xml*´ konfiguriert
    - Standard Servlet-Interface
  - Aufbau:
    - Standard Servlet

Statik - ActionForm

- *ActionForm*:
  - Umfeld:
    - läuft im WebContainer; enthalten im WAR
    - wird im Shared Context abgelegt
  - Aufgabe:
    - Datencontainer als Repräsentant eines Eingabeformulars
    - Validierung der Eingabe
  - Schnittstellen:
    - // Zurücksetzen aller Attribute auf Defaultwerte
      void reset(ActionMapping m, HttpServletRequest r);
    - // Validierung, aufgerufen durch ActionServlet
      ActionErrors validate(ActionMapping m, HttpServletRequest r);
    - ...
  - Aufbau:
    - JavaBean

WINCOR NIXDORF

Experience meets Vision

Beispiel - ActionForm

```
public class LoginForm extends ActionForm {

    private String username = null;
    private String password = null;

    public LoginForm() { super(); }

    public String getUsername() { return username; }
    public void setUsername(String u) { username = u; }

    public String getPassword() { return password; }
    public void setPassword(String p) { password = p; }

    public void reset(ActionMapping mapping, HttpServletRequest request) {
            username = null; password = null;
    }

    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
            ActionErrors errors = new ActionErrors();
            if (username.equals("") || password.equals("")) {
               errors.add("login", new ActionError("error.field.required"));
            }
            return errors;
    }
}
```

Banking Division

Application Software

Statik - Action

- *Action*:
  - Umfeld:
    - läuft im WebContainer; enthalten im WAR
    - es existiert immer nur genau eine Instanz der Action -> threadsafe programmieren!
  - Aufgabe:
    - Hilfs-Controller für das ActionServlet
    - für jeden Use-Case sollte eine Action existieren
    - verarbeitet ActionForms, kommuniziert mit BusinessLogic und verweist auf Folgeseite
  - Schnittstellen:
    - // HTTP Request verarbeiten oder weiterleiten
      ActionForward execute(ActionMapping m, ActionForm f, HttpServletRequest rq, HttpServletResponse rp);
    - // Speicherung der Fehlermeldungen im Scope void *saveErrors(HttpServletRequest r, ActionErrors e);*
    - ...
  - Aufbau:
    - Java Klasse

WINCOR NIXDORF

Experience meets Vision

Beispiel - Action

```
public class LoginAction extends Action {
    public LoginAction() { super(); }

    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                 HttpServletRequest request, HttpServletResponse response) {

        ActionErrors errors = new ActionErrors();
        ActionForward forward = new ActionForward();
        LoginForm loginForm = (LoginForm) form;
        String user = loginForm.getUsername();
        String pwd  = loginForm.getPassword();

        // get data from business logic
        if (BusinessLogic.isKnownUser(user, pwd)) { // user is known
            forward = mapping.findForward("success");
        }
        else { // user is unknown
            errors.add("login", new ActionError("error.inputerror"));
            forward = mapping.findForward("error");
        }

        // saveErrors in request
        if (!errors.empty()) { saveErrors(request, errors); }

        return (forward);
    }
}
```

Banking Division

Application Software

Statik - JSP

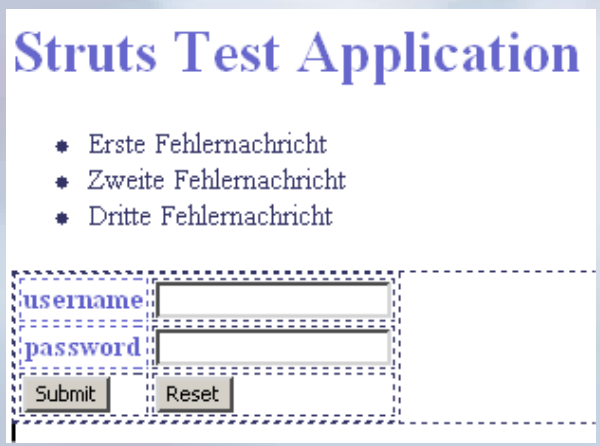
- *JSP*:
  - Umfeld:
    - läuft im WebContainer; enthalten im WAR
  - Aufgabe:
    - repräsentiert den View
    - nimmt Benutzereingaben entgegen und leitet diese an das ActionServlet
    - eine JSP pro dynamische HTML-Seite
  - Schnittstellen:
    - Verwendung von TagLibs
  - Aufbau:
    - JavaServer Page

WINCOR NIXDORF

Experience meets Vision

Beispiel - JSP

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

<BODY>
    <h1><bean:message key="global.title"/></h1>
    <html:errors/>

    <html:form action="/LoginAction">
        <TABLE border="0">
        <TBODY>
        <TR><TH>username</TH><TD><html:text property='username'  /></TD></TR>
        <TR><TH>password</TH><TD><html:password property='password' /></TD></TR>
        <TR><TD><html:submit property="submit" value="Submit" /></TD>
            <TD><html:reset /></TD>
        </TR>
        </TBODY>
        </TABLE>
    </html:form>

</BODY>
</html:html>
```



Banking Division

Application Software

Statik – Tag Library

- *Tag Library*:
  - Umfeld:
    - läuft im WebContainer; enthalten im WAR
    - Struts-TagLibs (HTML, Bean, Logic, Template, Nested)
    - Java Standard TagLibraries (JSTL)
    - Custom-Tags
  - Aufgabe:
    - ziehen Logik aus der JSP heraus und machen diese über definierte Schnittstellen zugreifbar; dadurch bessere Separation der Präsentation
  - Schnittstellen:
    - Deklaration in JSP und web.xml
    - Nutzung der Funktionalität über Tags
  - Aufbau:
    - Tag Handler: implementierende Java-Klasse
    - Tag Library Descriptor (TLD): Informationen über alle Tags
    - Application Deployment Descriptor: web.xml-Eintrag
    - Declaration: Deklaration in der JSP

Statik - Ressourcen

- *Ressourcen*:
  - Umfeld:
    - ´*ApplicationResources.properties*´ Datei im /WEB-INF/classes Verzeichnis
    - pro Locale eine Resourcen-Datei
  - Aufgabe:
    - enthält lokalisierte Einträge (z.B. Button-Texte, Textausgaben, Fehlerausgaben)
  - Schnittstellen:
    - /
  - Aufbau:
    - Text-Datei

WINCOR NIXDORF

Experience meets Vision

Beispiel - Resourcen

error.inputerror=Wrong userid or password
error.field.required=Field cannot be empty
error.exception=Exception occurred
global.title=Struts Test Application

Banking Division

Application Software

WINCOR NIXDORF

Experience meets Vision

Statik - Konfiguration

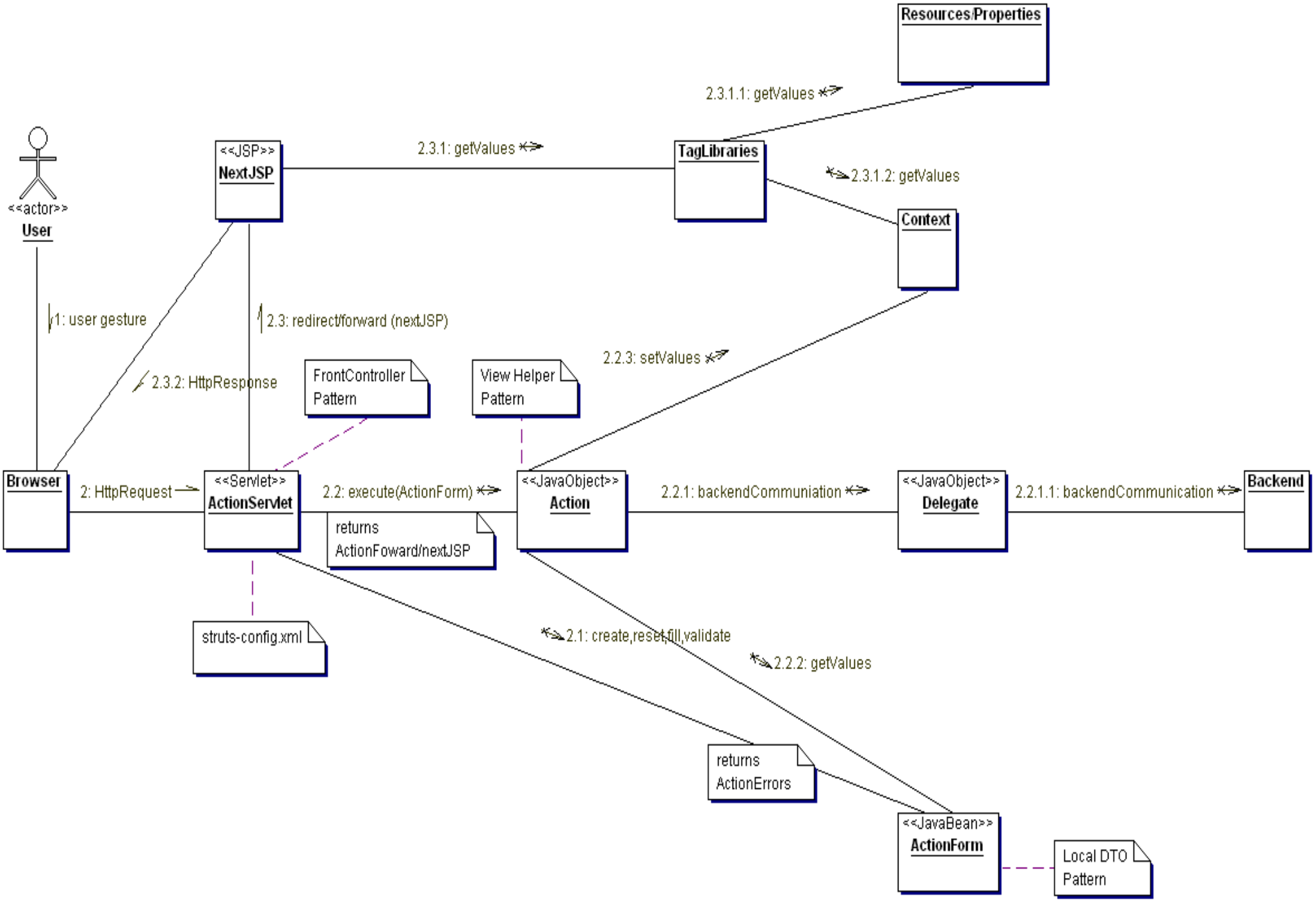
- *Konfiguration*:
  - Umfeld:
    - ´*struts-config.xml*´ Datei im /WEB-INF Verzeichnis
    - wird vom ActionServlet eingelesen und als ´ActionMapping´ zur Verfügung gestellt
  - Aufgabe:
    - zentrale Konfigurationsdatei ´*struts-config.xml*´
    - Konfiguration von:
      - ActionForms
      - ActionMappings
      - globale und lokale Forwards
  - Schnittstellen:
    - /
  - Aufbau:
    - XML-Datei

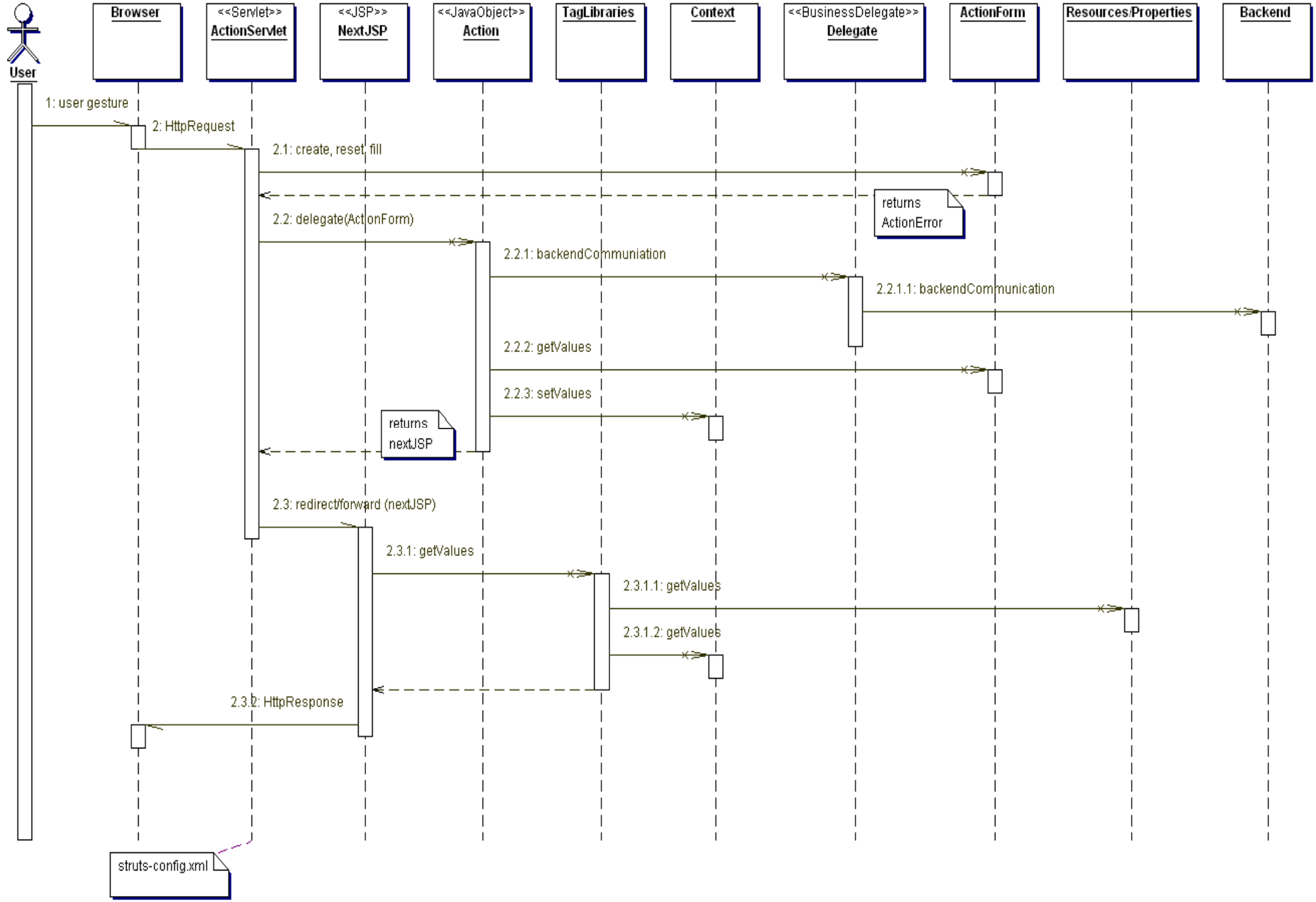
WINCOR NIXDORF

Experience meets Vision

Beispiel - Konfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<struts-config>

    <!-- Form Beans -->
    <form-beans>
        <form-bean name="loginForm" type="strutsdemo1.forms.LoginForm"></form-bean>
    </form-beans>

    <!-- Action Mappings -->
    <action-mappings>
        <action     path="/LoginAction"
                    type="strutsdemo1.actions.LoginAction"
                    name="loginForm"
                    scope="request"
                    unknown="false">
                    <forward name="success" path="/main.jsp"></forward>
                    <forward name="error" path="/login.jsp"></forward>
        </action>
    </action-mappings>

    <!-- Message Resources -->
    <message-resources parameter="strutsdemo1.resources.ApplicationResources"/>

</struts-config>
```

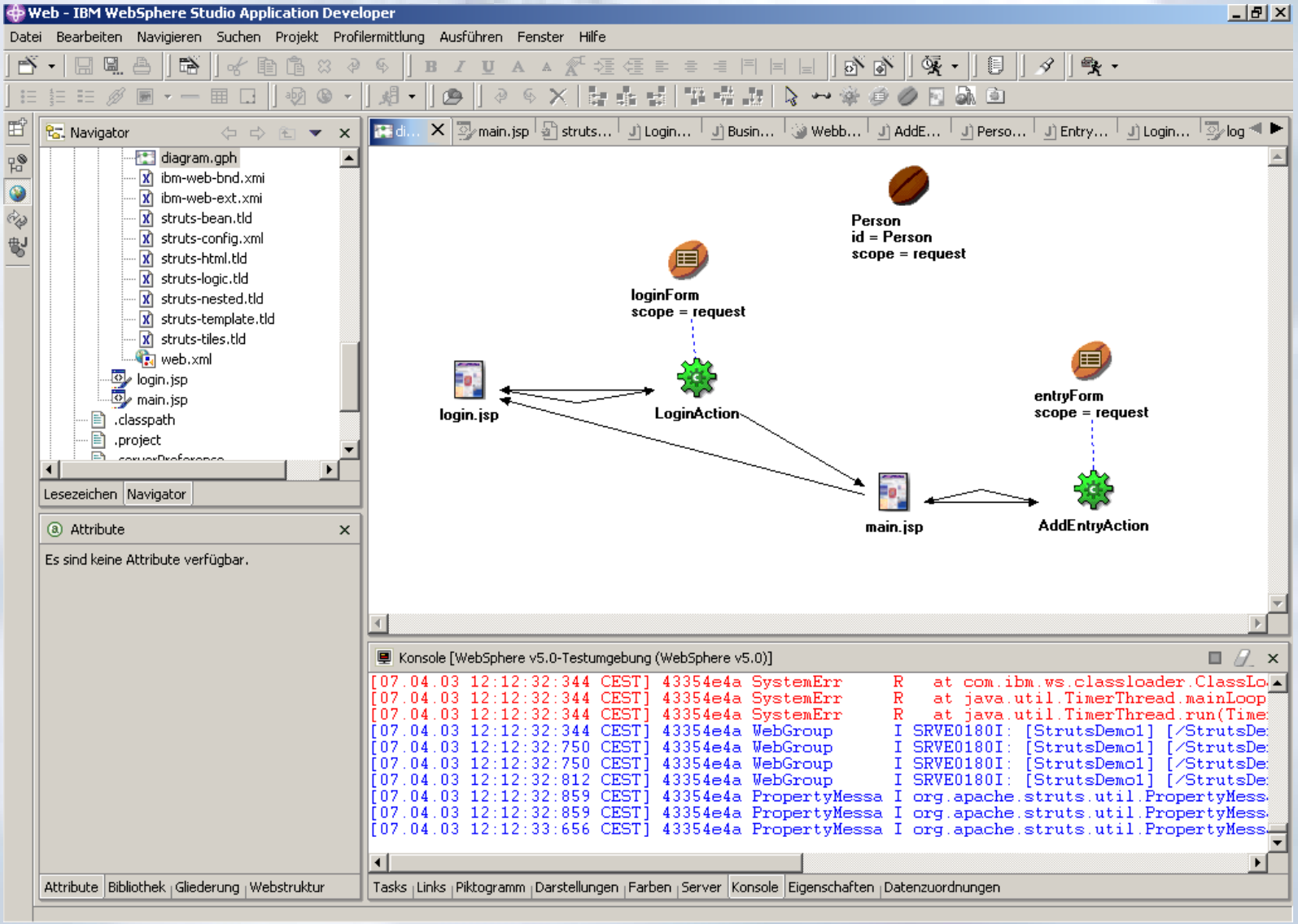

Resources/Properties
2.3.1.1: getValues
<<actor>>
User
<<JSP>>
NextJSP
2.3.1: getValues
TagLibraries
2.3.1.2: getValues
Context
1: user gesture
2.3: redirect/forward (nextJSP)
2.2.3: setValues
2.3.2: HttpResponse
FrontController
Pattern
View Helper
Pattern
Browser
2: HttpRequest
<<Servlet>>
ActionServlet
2.2: execute(ActionForm)
<<JavaObject>>
Action
2.2.1: backendCommuniation
<<JavaObject>>
Delegate
2.2.1.1: backendCommunication
Backend
returns
ActionFoward/nextJSP
struts-config.xml
2.1: create,reset,fill,validate
2.2.2: getValues
returns
ActionErrors
<<JavaBean>>
ActionForm
Local DTO
Pattern


User
Browser
<<Servlet>>
ActionServlet
<<JSP>>
NextJSP
<<JavaObject>>
Action
TagLibraries
Context
<<BusinessDelegate>>
Delegate
ActionForm
Resources/Properties
Backend
1: user gesture
2: HttpRequest
2.1: create, reset, fill
returns
ActionError
2.2: delegate(ActionForm)
2.2.1: backendCommuniation
2.2.1.1: backendCommunication
2.2.2: getValues
2.2.3: setValues
returns
nextJSP
2.3: redirect/forward (nextJSP)
2.3.1: getValues
2.3.1.1: getValues
2.3.1.2: getValues
2.3.2: HttpResponse
struts-config.xml

WINCOR NIXDORF

Experience meets Vision

IBM Websphere Studio Application Developer 5



Banking Division

Application Software