
(1) Komponenten bilden die zentralen Einheiten zum Aufbau einer Softwarearchitektur und im Speziellen auch der Architektur von Web-Anwendungen. Der erste Schritt von statischen Web-Seiten zu Web-Anwendungen führt zu den dynamischen Web-Seiten. Hierbei wird beim Aufruf einer Web-Seite eine Funktionalität durch Aufruf einer Anwendung auf dem Server ausgeführt.

(2) Die Java Enterprise Edition (JEE) liefert einem Software-Entwickler Basisfunktionalität in Form von sog. JEE-Containern und JEE-Komponenten, durch die sich eine 3-Schichten-Komponentenarchitektur in Java effizienter erstellen lässt.

(3) Servlets sind Java-Programme, die als Teil der JEE-Architektur auf dem Web-Server laufen und als Mittelschicht (engl. middle layer) Anfragen vom Web-Browser auf der einen Seite (Client) und Anfragen von Anwendungen, Web-Services, Datenbanken auf der anderen Seite (Server) entgegennehmen und bearbeiten. Zur Ausführung von Servlets wird ein Anwendungsserver (hier: Tomcat) benötigt, auf dem das entwickelte Servlet verteilt (engl. deploy) und ausgeführt wird.

(4) Eine JavaServer Page (JSP) ist, vereinfacht ausgedrückt, eine HTML-Seite, in die Java-Code eingebettet ist. Auch wenn JSPs vom Aussehen her ganz anders wirken wie Servlets, stellen sie eine spezielle Art der Formulierung eines Servlets dar (eine JSP wird vor deren Ausführung in ein Servlet umgewandelt).

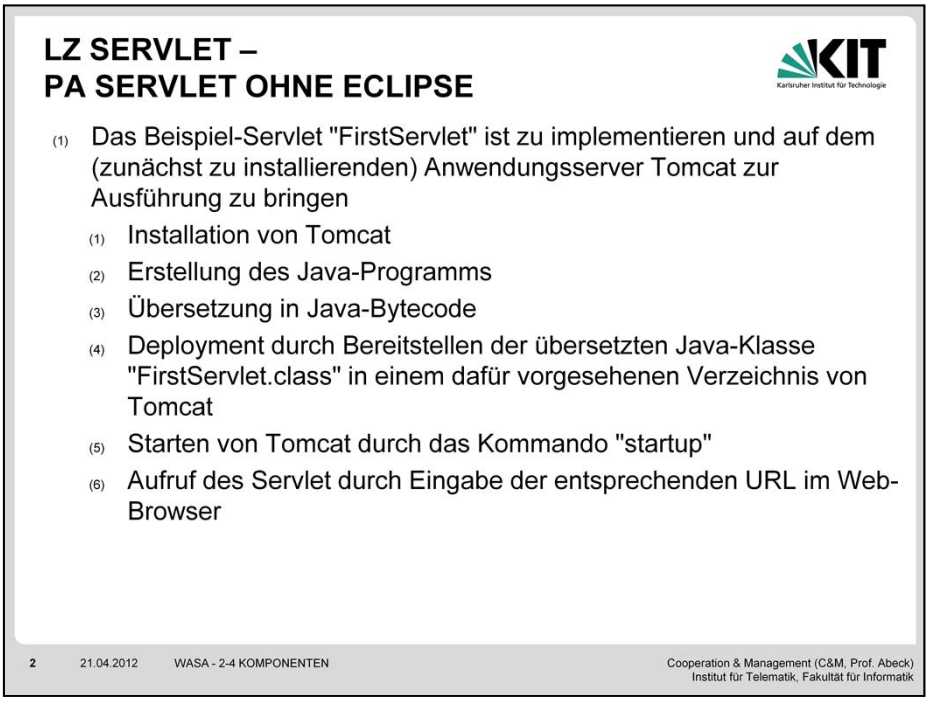
| | |
|---|---|
| HTML | HyperText Markup Language |
| JEE | Java Enterprise Edition |
| JSP | JavaServer Pages |

Hauptquellen:

[HB11] Marty Hall and Larry Brown: Core Servlets and JavaServer Pages, Free Online Version of Second Edition, http://pdf.coreservlets.com/, 2011.

[Li05] Y. Daniel Liang: Introduction to Java Programming; Companion Website: www.prenhall.com/liang, Pearson Prentice Hall, 2005.

[Sun-JEE5] Sun: Java EE 5 Tutorial, Sun Microsystems, September 2007.

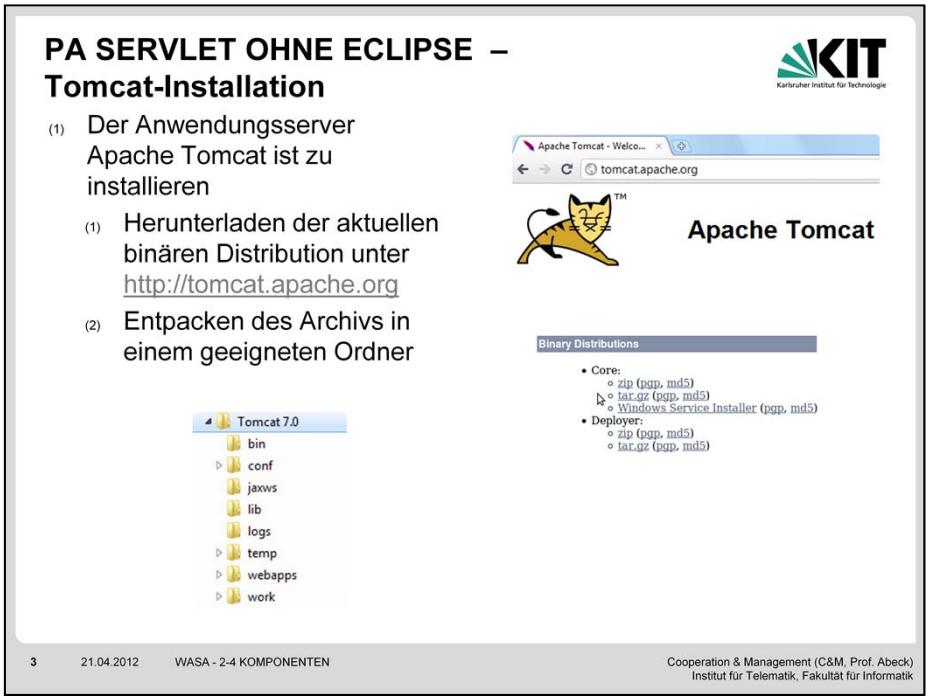



[Li05:1013]
Diese Praktische Aufgabe hat zum Ziel, alle notwendigen Schritte zur Ausführung eines einfachen (d.h. keine Funktion erbringenden) Servlets in einem Servlet-Container (hier: Tomcat) manuell durchzuführen, um das Zusammenspiel zwischen Servlet und Servlet-Container verstehen zu können.

(1.1) Die Installation ist ähnlich einfach wie die Installation der Eclipse-Umgebung.
(1.2) Die Erstellung erfolgt hier durch einen einfachen Text-Editor.
(1.3) Durch "javac FirstServlet" wird das Java-Programm "FirstServelt.jar" in "FirstServlet.class" übersetzt. Damit der Java-Compiler die von Tomcat bereitgestellten Servlet-Klassen findet, ist die Klassenvariable ("classpath") mit dem entsprechenden Pfad zu belegen. [:1014].
Hinweis: In einer nachfolgenden Praktischen Aufgabe wird Tomcat innerhalb von Eclipse genutzt; dann sorgt Eclipse automatisch dafür, dass die Bibliotheken gefunden werden.
(1.4) Das manuelle Verschieben oder Kopieren stellt die einfachste Form des Deployment (dt. Verteilung) dar und ist aber nur für sehr einfache Anwendungen geeignet.
Die Servlet-Klassen werden im Tomcat in einem Unterverzeichnis des "webapps"-Verzeichnisses abgelegt.
(1.5) Zum erfolgreichen Start benötigt Tomcat den Zugriff auf das JDK, weshalb in "Java_Home" der Verzeichnispfad von JDK zugewiesen sein muss. Das Kommando "startup" befindet sich im "/bin"-Verzeichnis von Tomcat.
(1.6) Der URL adressiert das Unterverzeichnis, in dem die Servlet-Klasse im Tomcat abgelegt ("deployt") wurde.
Beispiel: Liegt das Servlet "FirstServlet.class" in "<Tomcat-Verzeichnis>\webapps\examples\WEB-INF\classes", ist dieses über den URL "http://localhost:8080/examples/servlet/FirstServlet" aufzurufen.

| | |
|---|---|
| JDK | Java Development Kit |
| URL | Uniform Resource Locator |

[Li05] Y. Daniel Liang: Introduction to Java Programming; Companion Website: www.prenhall.com/liang, Pearson Prentice Hall, 2005.

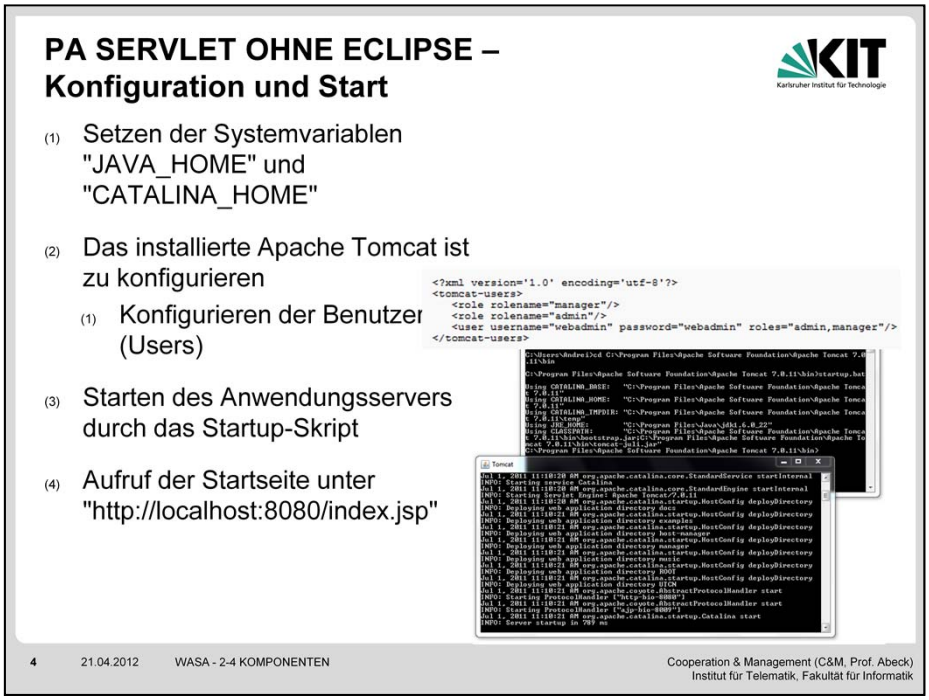



Viele der Verbesserungen in der neuen Version von Tomcat beziehen sich auf die Implementierung der neuesten JSR-Spezifikationen (Java Specification Request) zu Servlet 3.0, JSP 2.2 und EL 2.2 (Expession Language), die eine Vielzahl an neuen Features mitbringen. Tomcat 7 wurde auch im Hinblick auf Performanz und Sicherheit verbessert [TOMCAT7-CHANGES]. Die Installation von Apache Tomcat ist relativ einfach gehalten. Es muss lediglich ein gepacktes Archiv der Binärversion des Servers heruntergeladen und entpackt werden. Es sei darauf hingewiesen, dass hier die manuelle Installation von Tomcat durchgeführt wird. Betriebssystem-spezifische Installationswege, z.B. Installation als Windows-Service oder durch die Paketverwaltung von verschiedenen Linux-Distributionen, wird nicht behandelt und wird nicht empfohlen. Für weitere Informationen sei auf die Dokumentation von Apache Tomcat, sowie auf das FAQ und das Tomcat Wiki verwiesen.

(1.1) Das heruntergeladene Archiv (apache-tomcat-7.x) ist in einem geeigneten Verzeichnis (hier: "_Install_Downloads") abzulegen. Die hier beschriebene Installation trifft auf die Tomcat-Version 7.0.14 zu.
(1.2) Unter "_Install" wird ein Verzeichnis angelegt, in dem das Archiv entpackt wird. Als Bezeichnung wird der Name des Archivs gewählt, wobei der erste Buchstabe groß geschrieben ist (in diesem Falle also "Apache-tomcat-7.0.14").
Es wird empfohlen, die binäre Version des Anwendungsservers ("Binary Distributions" – "Core" - "zip") herunterzuladen. Da der Server zu Entwicklungs- und Testzwecken eingesetzt werden soll, wird nicht empfohlen, eine Version zu installieren, welche als Hintergrunddienst des jeweiligen Betriebssystems abläuft. Beim Herunterladen sollte aus Effizienzgründen darauf geachtet werden, einen Spiegelserver zu wählen, welcher nahe der eigenen Position gelegen ist.

JSR Java Specification Request

[TOMCAT7-CHANGES] Liste mit neuen Features in Tomcat 7, http://www.tomcat7.com/#changes-new-features




[BS+08:xxvi]
Hinweis: Je nach Installation und Version des Betriebssystems bzw. von Tomcat, müssen diese Konfigurationen nicht durchgeführt werden. Für den Fall, dass es doch nötig ist, die Systemvariablen zu setzen, werden aus Gründen der Vollständigkeit die Schritte nachfolgend erläutert.

(1) JAVA_HOME bezeichnet das Verzeichnis, in dem das JDK-Home-Verzeichnis liegt [Li-Tomcat:8].
- Das Verzeichnis ist z.B.: "C:_Install\Jdk-6u23-windows-i586".
- Über "Start -> Systemsteuerung -> System und Sicherheit -> System -> Erweiterte Systemeinstellungen -> Umgebungsvariablen" wird die Benutzervariable "JAVA_HOME", mit dem Wert des Installationspfades, angelegt.
CATALINA_HOME bezeichnet das Verzeichnis, in dem der Anwendungsserver Apache Tomcat liegt.
- Das Verzeichnis ist "C:_Install\Apace-Tomcat-7.0.14". Tomcat kann im Normallfall auch ohne diese Variable auskommen. Es gibt dennoch Fälle, in denen sie gesetzt werden muss.

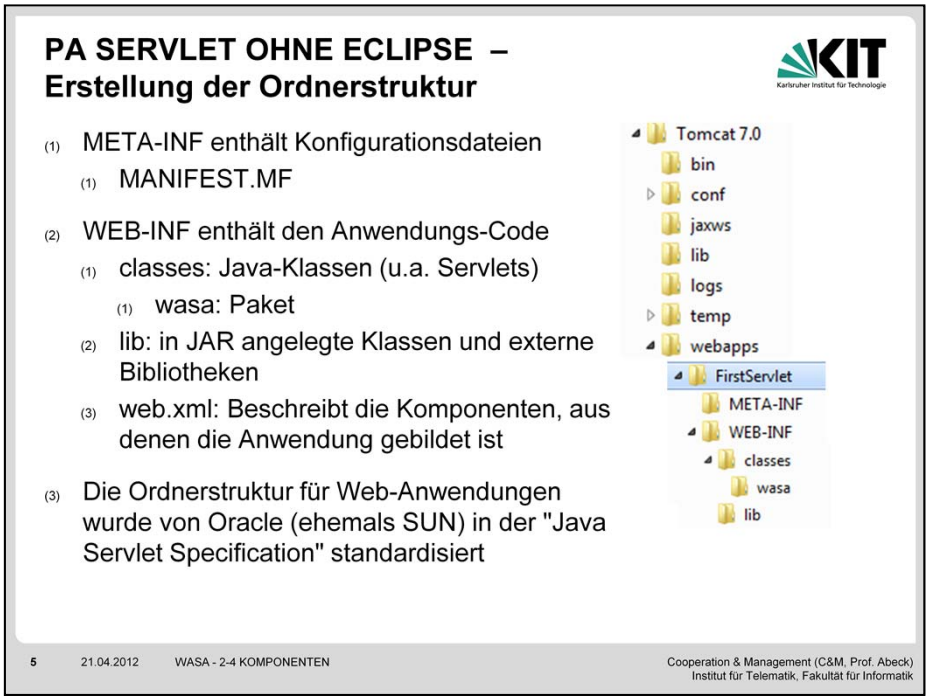
(2) Zur Konfiguration der Manager ist die Änderung der Tomcat-Konfigurationsdatei "tomcat-users.xml" im Verzeichnis "$TOMCAT_HOME/conf" notwendig.
(2.1) In "tomcat-users.xml" können neue Nutzer und ihre Rollen definiert werden.

Hinweis: Es ist hilfreich, die auf dem Server installierten Web-Anwendungen mittels einer Browser-basierten Oberfläche durch den Tomcat-Manager zu verwalten.

(3) Das Starten bzw. Hochfahren (engl. launch) von Apache Tomcat erfolgt im Windows-DOS-Eingabefenster, indem
(i) in das Tomcat-Verzeichnis gewechselt wird (z.B. "cd C:_Install\Apache-tomcat-7.0.x\bin")
(ii) das Startup-Skript aufgerufen wird: "startup" (bzw. "startup.sh" für Linux-Nutzer) [Li05:1015].

(4) Eingabe der Web-Adresse "http://localhost:8080/index.jsp" in einem Standard-Webbrowser. Die Startseite liegt im Verzeichnis "$TOMCAT_HOME/webapps/ROOT/index.jsp".

[BS+08] Bryan Basham, Kathy Sierra, Bert Bates: Head First Servlets & JSP, Second Edition, O'Reilly, 2008.
[Li-Tomcat] Y. Daniel Liang: Installing and Configuring JDK 1.6, Supplement for Introduction to Java Programming, http://cs.armstrong.edu/liang/intro7e/supplement/Supplement6eTomcat5.5.9.pdf.



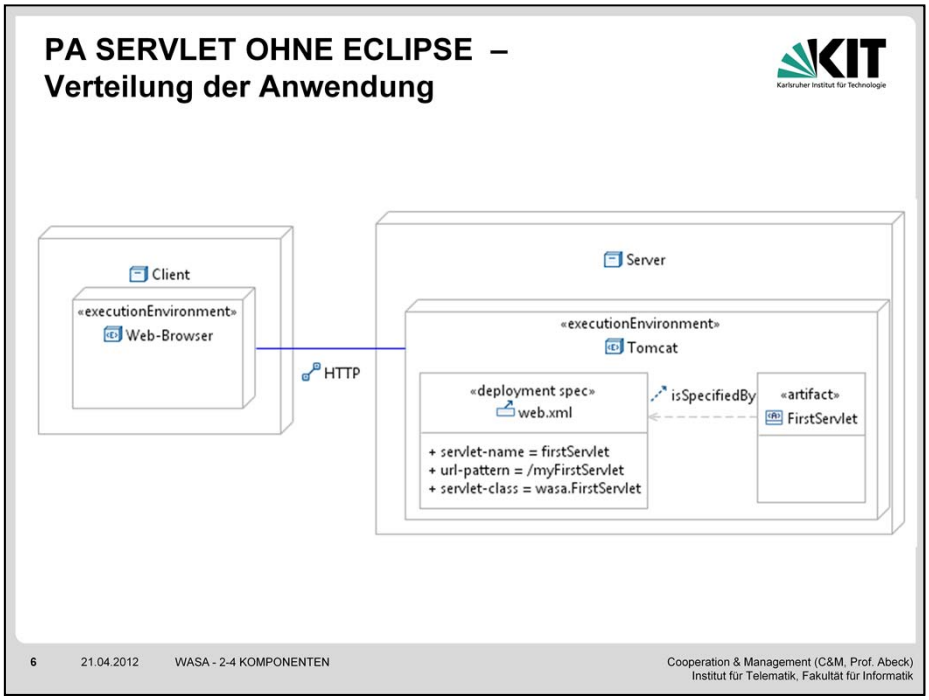

Um Web-Anwendungen in Tomcat ausführen zu können, muss der Code in dem "webapps"-Ordner unter der Tomcat-Installation abgelegt werden. Eine eigenständige Web-Anwendung (hier: "FirstServlet") wird durch einen Ordner im "webapps"-Ordner dargestellt. Der "FirstServlet"-Ordner ist der Wurzelordner. Dateien, die für den Benutzer sichtbar sein sollten (z.B. HTML oder JSP-Dateien), können direkt im Wurzelordner oder in einem Unterordner platziert werden.

(1) Im "META-INF"-Ordner werden Dateien zur Konfiguration eines Java-Archives (JAR) gespeichert [JAR-SPEC]. Ein Web-Archiv (WAR) ist ein JAR mit einer Struktur, die für Web-Anwendungen geeignet ist.
(1.1) Die "MANIFEST.MF"-Datei enthält Informationen zu den Paketen oder zu den Erweiterungen der Anwendung. Bei der Erstellung eines JAR (Java Archive) durch das Kommando "jar cf <jar-name> <input-file(s)>" wird die Datei automatisch vom Übersetzer erzeugt. Das Kürzel "cf" stammt von "create file". Analog erfolgt die Erstellung der WAR-Dateien durch das Kommando "jar cf <war-name>.war <input-file(s)>". Die "MANIFEST.MF"-Datei ist nicht notwendig, um das "FirstServlet" auszuführen. Da die Datei gemäß [JAR-SPEC] gefordert wird, erstellt das Kommando die Datei und ergänzt sie um die Standardzeile "Manifest-Version: 1.0".

(2) Der "WEB-INF"-Ordner ist von der Client-Seite aus nicht zugreifbar. Alles, was unter diesem Ordner liegt, muss in der "web.xml"-Datei angegeben werden, um zugreifbar zu sein (vgl. Servlet-Deklaration auf späteren Seiten). Der "WEB-INF"-Ordner ist ein geeigneter Platz, um Ressourcen abzulegen, die nur auf der Server-Seite sichtbar sein sollten [TOMCAT-DEPL].
(2.1) Die Java-Klassen, die von der Anwendung gebraucht werden und gegebenenfalls auch die Quelldateien werden in den "WEB-INF/classes/"-Ordner hinterlegt. Diese können Servlet-Klassen sein oder nicht, die aber nicht zu einem JAR zusammengesetzt wurden. Die Einteilung in Paketen muss sich in der Verzeichnisstruktur widerspiegeln. Wenn eine Klasse "com.servlet.MyServlet" heißt, dann muss sie im Verzeichnis "/WEB-INF/classes/com/servlet/MyServlet.class" liegen.
(2.1.1) Das FirstServlet wird in ein Paket mit der Bezeichnung"wasa" gelegt.
(2.2) Java "class"-Dateien, die in JARs abgelegt sind, wie z.B. externe Bibliotheken oder JDBC-Treiber, die in der Anwendung benutzt werden, sollten im "WEB-INF/lib/"-Ordner abgelegt werden. Dies ist auch der Ort, an dem Tomcat bei der Ausführung zuerst nach externen Bibliotheken sucht.
(2.3) Der Web-Anwendungsdeskriptor ist eine XML-Datei Namens "web.xml". Die Datei beschreibt die Servlets oder andere Komponenten, die die Web-Anwendung ausmachen. Zudem können auch Initialisierungsparameter und Sicherheitseinschränkungen spezifiziert werden.

(3) Die JAR-Spezifikation von Oracle kann in [JAR-SPEC] gefunden werden und Näheres zu dem WAR-Format kann in [SERVLET-SPEC], [ORACLE-WAR] und [TOMCAT-DEPL] gefunden werden.

[TOMCAT-DEPL] Tomcat Deployement Guide, http://tomcat.apache.org/tomcat-7.0-doc/appdev/deployment.html
[JAR-SPEC] Oracle Java Archive Spezifikation, http://download.oracle.com/javase/1.5.0/docs/guide/jar/jar.html
[ORACLE-WAR] Oracle Web Archive Deployement,
http://download.oracle.com/docs/cd/A91202_01/901_doc/java.901/a90213/war.htm
[JAR-FILE] Guide on Java Archive Files, http://download.oracle.com/javase/tutorial/deployment/jar/build.html
[SERVLET-SPEC] Java Servlet Specification. http://www.oracle.com/technetwork/java/javaee/servlet/index.html




Durch das UML-Verteilungsdiagram (engl. deployment diagram) lassen sich die an der zu entwickelnden Software-Lösung beteiligten Rechensysteme in Form von Knoten (engl. node) und darauf ablaufenden Laufzeitumgebungen und Artefakte übersichtlich in graphischer Form darstellen.

(Client) Der Client-Knoten mit einem Web-Browser als Ausführungsumgebung (engl. execution environment). Die Ressourcen werden mit Hilfe des Web-Browsers aufgerufen.

(Tomcat) Tomcat ist die Ausführungsumgebung der Servlets. Die Ausführungsumgebung wird mit Hilfe des Stereotyps <<executionEnvironment>> dargestellt [Ke09:161].

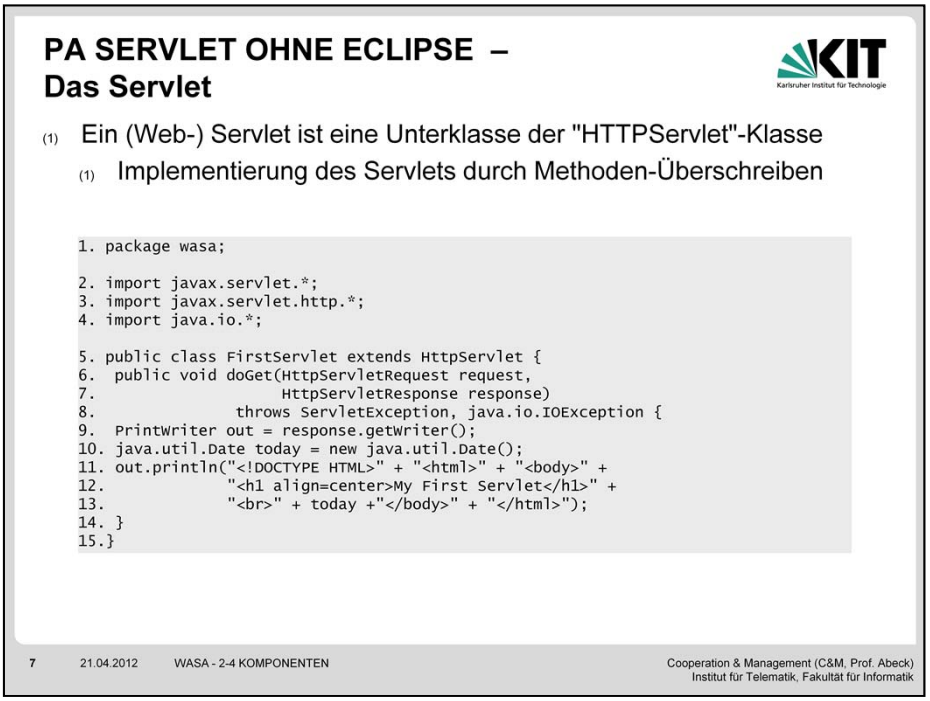
(HTTP) Ein Kommunikationspfad (engl. communication path), der durch das HTTP-Protokoll die Kommunikation des Web-Browsers mit dem Web-Container "Tomcat" ermöglicht.

(<<deployment spec>>) Die Spezifikation für das Deployment der Artefakte, wie z.B. das FirstServlet-Servlet. Die Konfigurationsdaten des FirstServlet-Servlets werden, bei der Implementierung, in der Deployement-Spezifikation web.xml eingetragen.

(isSpecifiedBy) Dies ist eine Beziehung zwischen der <<deployment spec>> und dem zu auf dem Tomcat zu verteilenden (engl. deploy) Artefakt, dem Servlet "FirstServlet".

(FirstServlet) Das FirstServlet ist das Servlet, das in Tomcat ausgeführt wird. Die Artefakte werden mit Hilfe des Stereotyps <<artifact>> dargestellt [Ke09:161].

[TOMCAT-UG] Tomcat - A Minimalistic User's Guide, http://www.jajakarta.org/tomcat/tomcat3.2-4.0/tomcat-3.2.3/doc/uguide/tomcat_ug.html
[Ke09] Christoph Kecher: UML2 Das umfassende Handbuch, 3. Auflage, Galileo Computing, 2009.

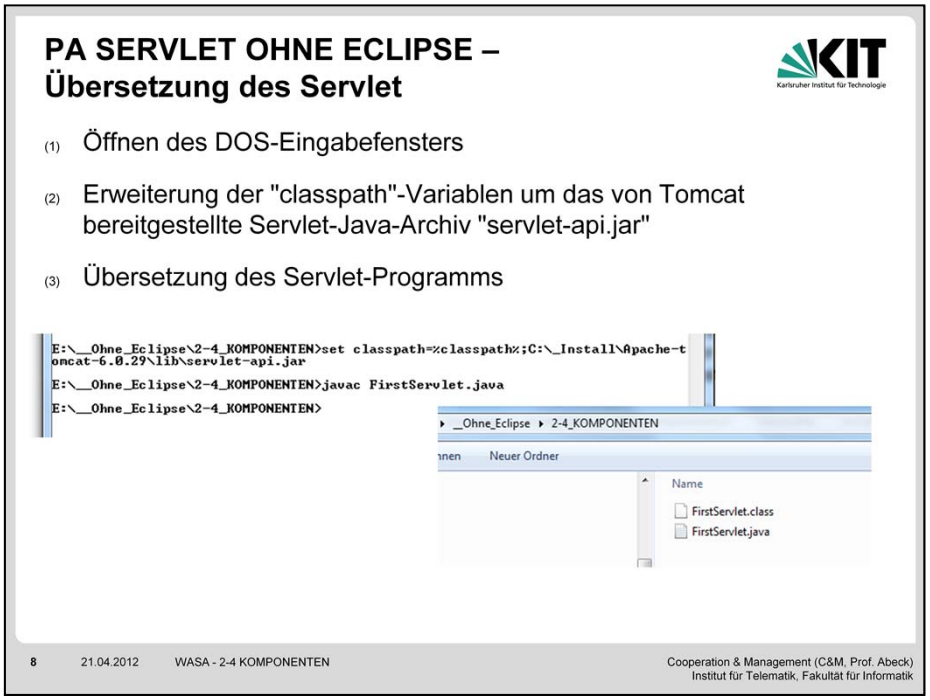



[BS+08:44]
Der Java-Code des Servlets "FirstServlet" ist in einem einfachen Editor zu erfassen und nachfolgend zu kompilieren. Unter Windows kann hierzu der unter "Start -> Programme -> Zubehör -> Editor" zu findende Editor genutzt werden
(alternativ: Rechtsklick auf "Datei -> Öffnen mit -> Editor")

(1) (5.) Die Unterklassen- bzw. Vererbungsbeziehung wird in Java durch "extends" ausgedrückt.
(1.1) (6.) Im Beispiel wird die Methode "doGet()" überschrieben.

(1. package) Die Paket-Deklaration
(5. ... extends) Die Klasse "FirstServlet" erweitert (erbt von, verfeinert) die im importierten Paket "javax.servlet.http" (2.) enthaltene Klasse "HttpServlet". Praktisch alle (laut [BS+08:44] 99,99999%) sind HttpServlets.
(6. ... doGet) Diese Methode wird vom Servlet-Container aufgerufen, wenn ein Web-Browser eine HTTP-GET-Methode an das Servlet schickt. Hierzu muss im Browser der URL des Servlets ("http://"<rechner>:<port>"/"<Verzeichnispfad>"/FirstServlet", z.B.
"http://localhost:8080/examples/servlet/FirstServlet") angegeben werden.
Nahezu alle Servlets überschreiben entweder die Methode "doGet()" oder die Methode "doPost()".
(6, ... request, 7. ... response) Referenzen auf die Objekte (in den Abbildungen zuvor als "req" und "rsp" bezeichnet), die vom Container erzeugt wurden.
(9. ... response.getWriter) Vom "response"-Objekt, das das Servlet vom Container bekommt, kann man einen "PrintWriter" erhalten.
(11. out.println) Über "PrintWriter" lässt sich HTML-Text in das "Response"-Objekt schreiben.
Hinweis: Es kann dem ausgebenden Browser explizt angezeigt werden, dass der Inhalt in der Sprache HTML verfasst ist (z.B. mit dem HTML 5 Doctype-Tag). Einfacher Text wird beispielsweise durch "text/plain", GIF-Bilder werden durch "images/gif" angegeben.

[BS+08] Bryan Basham, Kathy Sierra, Bert Bates: Head First Servlets & JSP, Second Edition, O'Reilly, 2008.

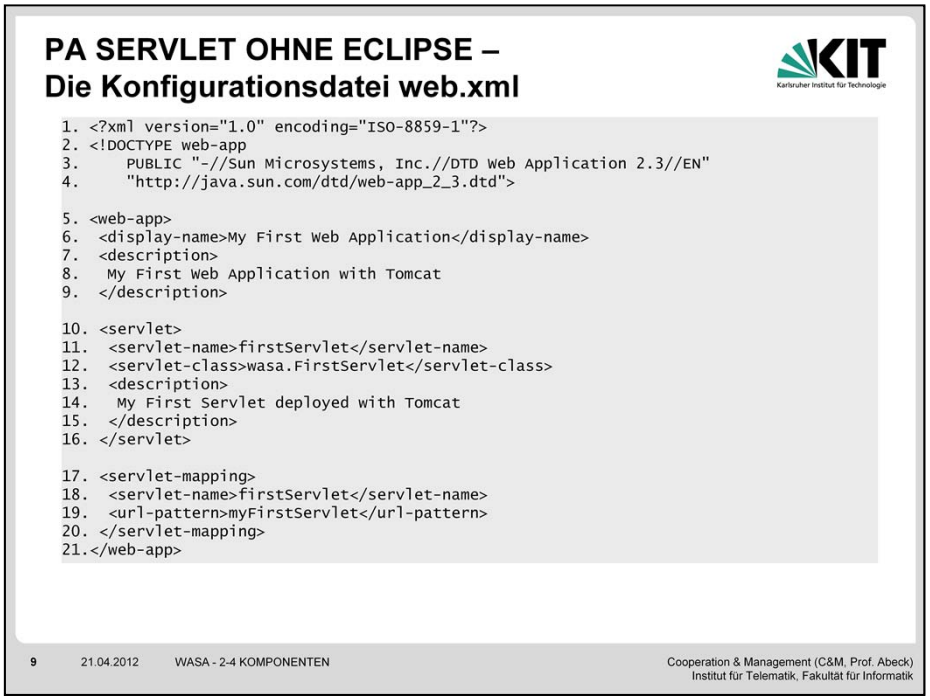



Die kompilierte Servlet-Klasse soll im "WEB-INF/classes/wasa"-Ordner gespeichert werden.

(1) "Start" -> "Zubehör" -> "Eingabeaufforderung"
Ggf. Wechseln in ein anderes Laufwerk, z.B. "C:\> E:" (d.h. kein Befehl, nur der Laufwerksbezeichner).

(2) Das Servlet-Java-Archiv, das mit dem Tomcat-Server ausgeliefert wird, beinhaltet die nötigen (in einem JAR enthaltenen) Java-Klassen, um Servlets übersetzen zu können.
"set classpath=%classpath%;C:_Install\Apache-tomcat-7.0.14\lib\servlet-api.jar"

(3) Ergebnis der Übersetzung ist die Datei "FirstServlet.class".

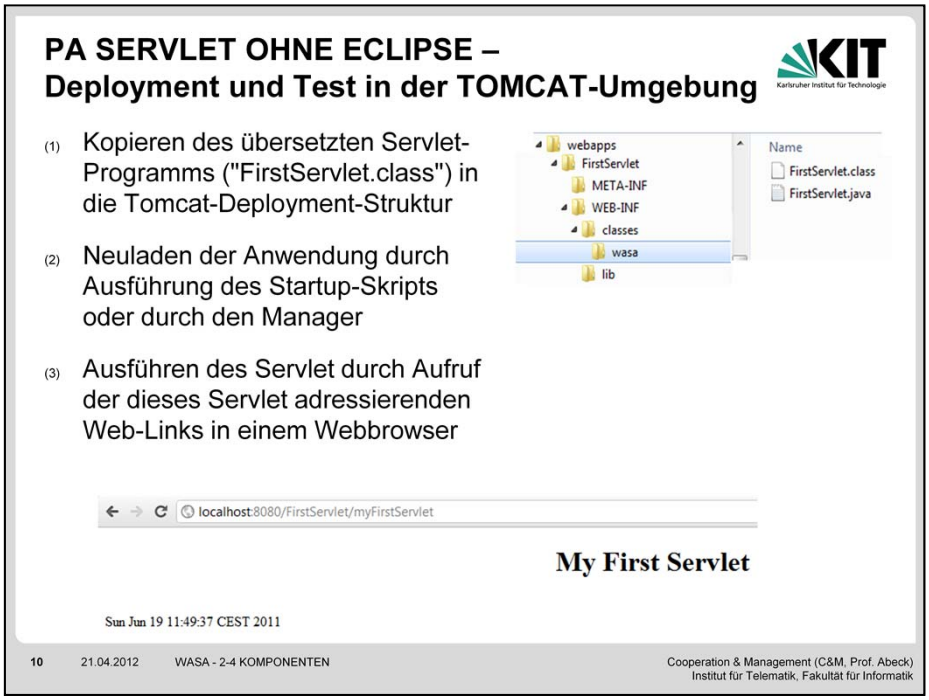



Tomcat muss nicht unbedingt neu gestartet werden, damit Änderungen an der Konfigurationsdatei in Kraft treten. Nur die Web-Anwendung muss neu deployed werden. Durch den Tomcat-Manager (unter "/manager" (z.B.: http://localhost:8080/manager)) kann eine einzige Web-Anwendung re-deployed werden, andernfalls werden beim Neustart alle Anwendungen re-deployed. Aus Einfachheitsgründen und weil die Umgebung als Entwicklungsumgebung benutzt wird, ist es einfacher, einen Server-Neustart durchzuführen.

(1. <?xml ...) Die Standard XML-Deklaration (Prolog).
(2. <!DOCTYPE ...) Der Dokumenttyp, der eine Konfigurationsdatei für Java-Web-Anwendungen definiert.
(5. <web-app> ) Das Wurzelelement "web-app" enthält die Beschreibungen der Web-Anwendung.
(7. <display-name>) Ein Kurzname kann für die Web-Anwendung angegeben werden.
(8. My First ...) Jedem Element in der Konfigurationsdatei kann eine textuelle Beschreibung durch das "description"-Element hinzugefügt werden.
(10. <servlet>) Das "servlet"-Element deklariert ein Servlet, das durch (i) den Namen ("servlet-name"-Element) und (ii) die Klasse ("servlet-class"-Element) definiert wird.
(11. <servlet-name>) Mit dem im "servlet-name"-Element deklarierten Namen kann das Servlet in der Konfigurationsdatei referenziert werden (siehe weiter unten bei "servlet-mapping", Zeile 18.).
(12. <servlet-class) Der Pfad der Klasse relativ zu dem "WEB-INF/classes"-Ordner kann durch die Punkt-Notation (".") beschrieben werden. Der hier angegebene Wert "wasa.FirstServlet" bedeutet, dass das Servlet mit Namen "firstServlet" in der Ordnerstruktur "WEB-INF/classes/wasa/" liegt.
(17. <servlet-mapping>) Das "servlet-mapping"-Element bestimmt die URL, mit der das Servlet aufgerufen wird.
(18. <servlet-name>) Der im "url-pattern"-Element angegebene Wert bedeutet, dass das Servlet mit Namen "firstServlet" durch die URL <host_name>:<port>"/"<web_app_name>"/<url-pattern> (im Beispiel: "localhost:8080/FirstServlet/myFirstServlet") aufgerufen werden kann.
Hinweis: <web_app_name> ist der Name des Ordners direkt unterhalb des "webapps"-Ordners in der Tomcat-Deployment-Struktur.

Näheres zu den Deskriptor-Elementen ist in [MW-WEB-XML] beschrieben und eine Beispiel-web.xml-Datei von der Apache Foundation kann unter [TOMCAT-WEB-XML] gefunden werden.

[TOMCAT-WEB-XML] Apache Software, Beispiel Konfigurationsdatei für den Tomcat 7 Servlet-Container, http://tomcat.apache.org/tomcat-7.0-doc/appdev/web.xml.txt
[MW-WEB-XML] MetaWerx, Beschreibung der möglichen Deskriptoren einer Konfigurationsdatei für den Tomcat Servlet Container, http://wiki.metawerx.net/wiki/Web.xml

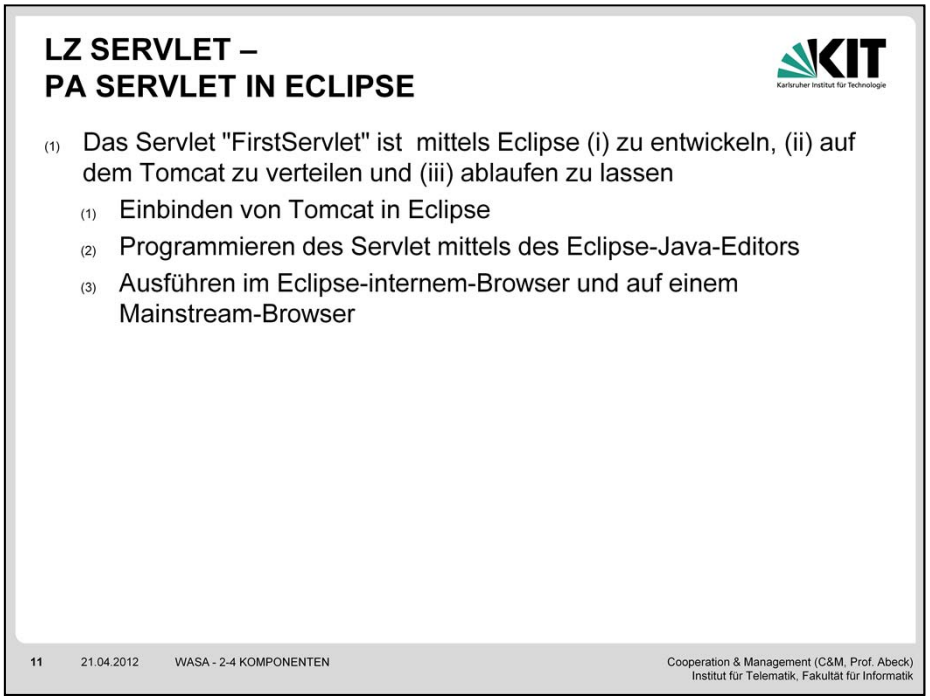



Unterhalb von "$TOMCAT_HOME\webapps" sollte der "FirstServlet"-Ordner liegen, der die Web-Anwendung repräsentiert.

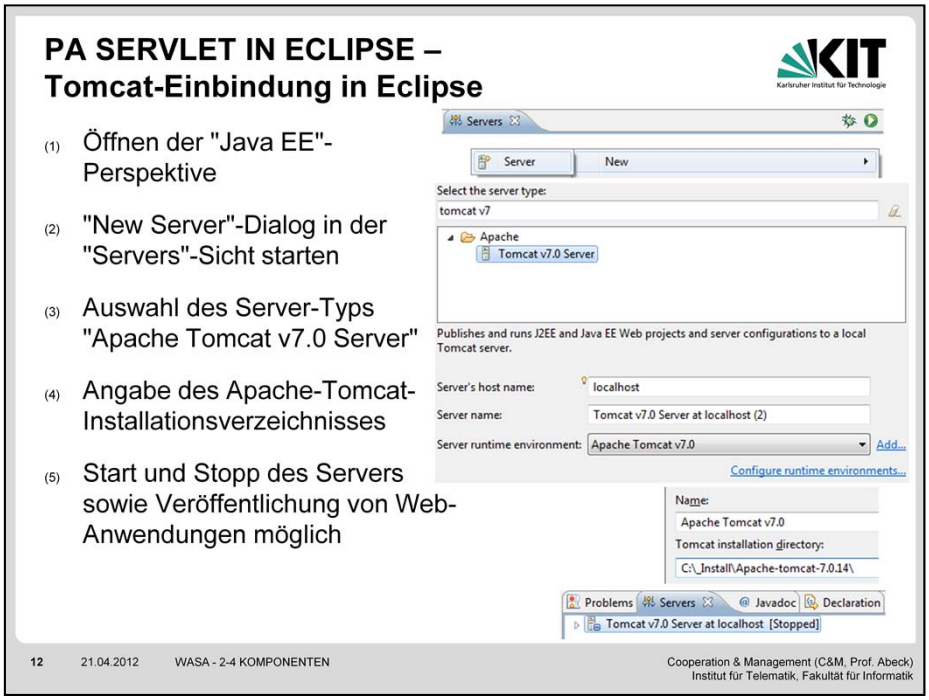
(1) Hierdurch wird sichergestellt, dass die übersetzte Servlet-Klasse zur Laufzeit gefunden wird. Laut "web.xml"-Konfigurationsdatei muss es unter "FirstServlet/WEB-INF/classes/wasa" liegen.

(2) Das Startup-Skript ist unter "C:_Install\Apache-tomcat-7.0.14\bin> startup" zu finden. Das Starten/Stoppen von Anwendungen kann auch durch den Tomcat-Manager ausgeführt werden. Der Manager ist zu finden unter "http://localhost:8080/manager/". Der Benutzername und das Passwort von Tomcat müssen dafür eingegeben werden (Standard Account: root, Passwort: root).

(3) Die Eingabe der Adresse "http://localhost:8080/FirstServlet/myFirstServlet" in einem Web-Browser sollte die Servlet-Seite aufrufen.




(1) Das Einbinden von Tomcat in Eclipse ermöglicht eine leichtere Entwicklung, da der Tomcat-Container und die darin verteilten (engl. deploy) Anwendungen direkt über die Eclipse-IDE gesteuert werden können.
(1.1) Hierdurch lässt sich Tomcat u.a. einfach starten (und stoppen) und Web-Anwendungen können effizient veröffentlicht werden.
(1.2) Effizienzsteigerung u.a. durch die fortlaufende Syntax-Überprüfung
(1.3) Als Mainstream-Browser werden gängige Browser wie Google Chrome, Mozilla Firefox oder Microsoft Internet Explorer bezeichnet.




Das Eclipse WTP-Plugin (Web Tools Platform) liefert bereits die nötigen Erweiterungen zur Entwicklung von dynamischen Java-basierten Web-Anwendungen. In Eclipse werden die Web-Anwendungen für eine bestimmte Laufzeitumgebung entwickelt, welche die notwendigen Bibliotheken bereitstellt. Eine solche Laufzeitumgebung wird von einem Applikationsserver bereitgestellt. Eclipse WTP liefert keine solche Laufzeitumgebung. Es ist also notwendig, einen geeigneten Applikationsserver zu installieren und dessen Laufzeitumgebung in Eclipse zu konfigurieren. Standardmäßig wird bei C&M zu Entwicklungs- und Testzwecken als Laufzeitumgebung der Applikationsserver Apache Tomcat eingesetzt.

(1) In der "Java EE"-Perspektive von Eclipse befindet sich standardmäßig im unteren Bereich die "Server"-Sicht, durch die Applikationsserver in Eclipse verwaltet werden können. Das Öffnen der "Java EE"-Perspektive erfolgt durch "Window" -> "Other Perspective" -> "Java EE".

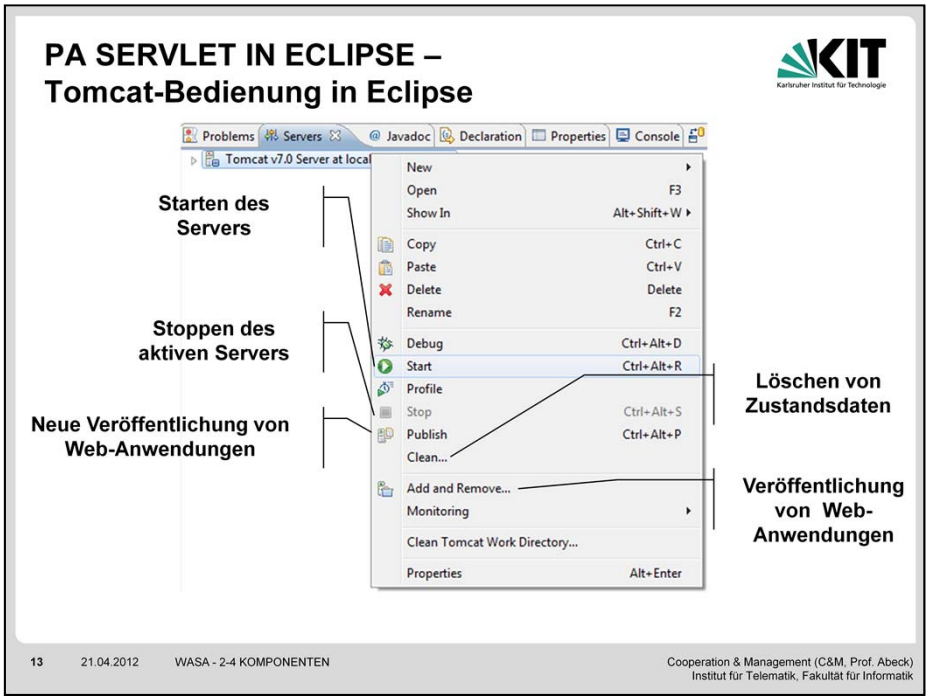
(2) In der "Servers"-Sicht kann durch die Betätigung der rechten Maustaste ein Kontextmenu geöffnet werden, welches die Definition einer neuen Serververbindung ermöglicht. Eine andere Möglichkeit besteht in der Auswahl des Menüs "File"->"New"->"Server".

(3) In dem sich öffnenden Dialog kann aus einer Liste von möglichen Anwendungsservern ausgewählt werden. Durch Eingabe des Suchbegriffs "tomcat v7" in das Filterfeld "type filter here" lässt sich der Apache-Tomcat-Anwendungsserver einfach selektieren. Wie bei der zuvor manuell (d.h. ohne Eclipse) durchgeführten Installation und Konfiguration von Apache Tomcat wird die Version 7 von Tomcat eingesetzt. Die Auswahl muss mittels "Next" bestätigt werden.

(4) Eclipse benötigt zur Verwaltung von Tomcat 7 das Installationsverzeichnis von Tomcat. Wie bei der zuvor gezeigten Installation und Konfiguration von Apache Tomcat beschrieben, ist dieses durch die Variable $TOMCAT_HOME festgelegt. Die Angaben müssen wiederum durch das Betätigen von "Next" bestätigt werden.

(5) Nach dem Schließen des Dialogs erscheint der Eintrag "Tomcat v7.0 Server at localhost" in der "Server"-Sicht der "Java EE"-Perspektive. Durch die Betätigung der rechten Maustaste auf dem Eintrag gelangt man zu einem Kontextmenü, mit dem der Server gestartet und gestoppt werden kann sowie Anwendungen auf dem Server bereitgestellt werden können.

WTP Web Tools Platform



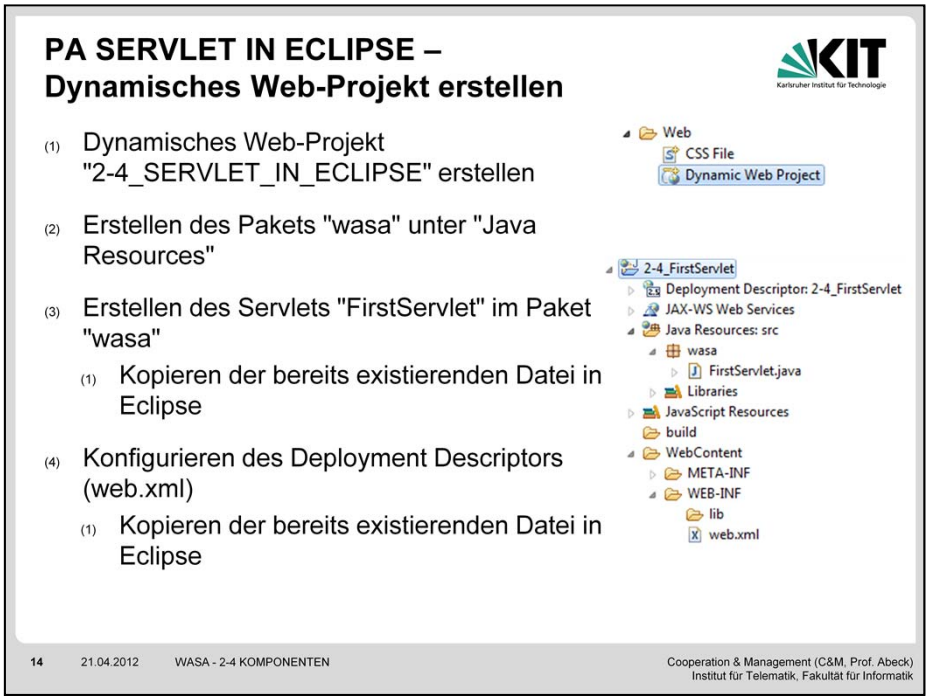

Innerhalb der Server-Sicht lässt sich Tomcat starten und stoppen oder im Debug-Modus ausführen. Weitere Einstellungsmöglichkeiten finden sich im Konfigurationsmenü des Servers, das durch Doppelklick auf den Server geöffnet werden kann.

(Add and Remove) Zusätzlich lassen sich in Eclipse vorhandene Web-Anwendungen auf dem Server veröffentlichen (durch den "Add and Remove…" Dialog). In diesem Dialog kann ausgewählt werden, welche der im Workspace vorhandenen Web-Projekte zu veröffentlichen sind.

(Clean) Beim Löschen ("Clean") der Zustandsdaten werden Informationen wie der Servlet-Kontext und die Session und die dazu gehörigen Session-Attribute entfernt.

(Clean Tomcat Work Directory) Durch das Löschen des Arbeitsverzeichnisses werden alle temporären Dateien gelöscht, die Tomcat erstellt hat. Hierdurch kann man Tomcat dazu zwingen, alle Dateien neu zu generieren.

(Debug) Im Debug-Modus kann z.B. bei Haltepunkten in den Java-Quelldateien die Programmausführung gestoppt und Variablen können inspiziert werden. Das Debuggen eines Servlets entspricht damit weitgehend dem Debuggen eines normalen Java-Programms.




Um eine bessere Übersicht im Workspace zu haben, ist für jede Praktische Aufgabe ein neues Projekt mit der Namenskonvention "<Kurseinheitennummer>_<PA-Kurzbezeichnung>" anzulegen.

Der Unterschied zwischen einem statischem und einem dynamischen Web-Projekt besteht darin, dass im statischen Web-Projekt keine JSP-Seiten und Servlets erlaubt sind. Ein statisches Web-Projekt kann in ein dynamisches Web-Projekt durch "Convert to a Dynamic Web Project" vom "Project"-Menü, konvertiert werden [ECLIPSE-DYN-WEB-PROJ].

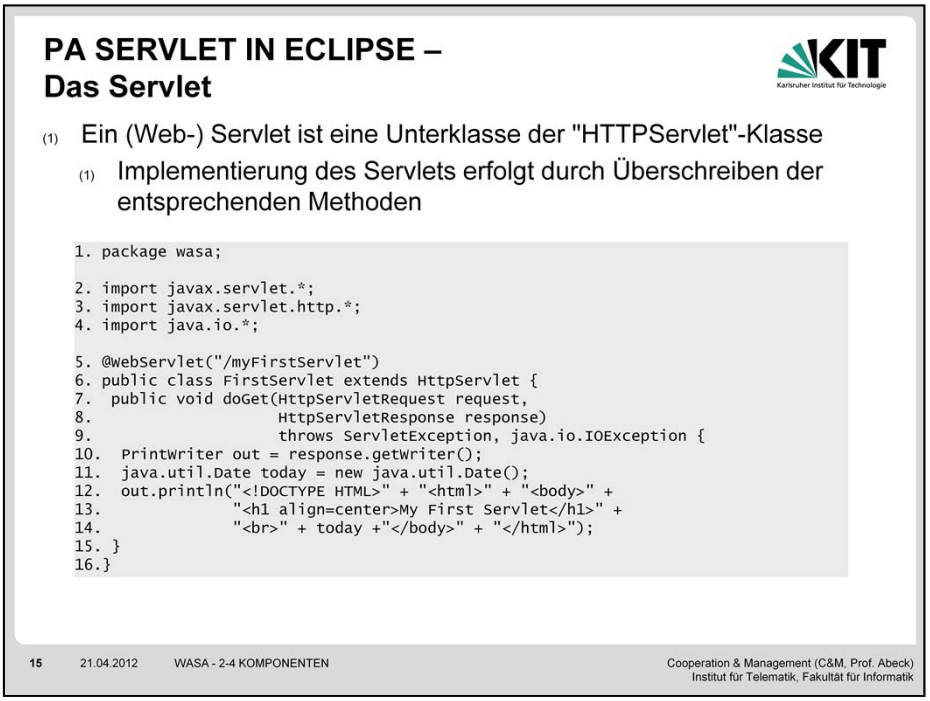
(1) In Eclipse wird im "Project-Explorer" mittels "Rechtsklick"->"New"->"Project" der Dialog "New Project" geöffnet und im Ordner "Web" wird das "Dynamic Web Project" ausgewählt. Es öffnet sich nun ein Wizard zur Erstellung des Projektes. Nachdem der Name für das Projekt eingegeben wurde, wählt man als "Target Runtime" den Apache Tomcat 7.0 Server aus. Anschließend kann der Wizard mit den Default-Einstellungen beendet werden.

(2) Ein Vergleich zeigt, dass die Projekt-Struktur in Eclipse nicht dieselbe ist wie die Struktur, die in Tomcat benutzt wird. Eclipse verwendet eine veränderte Struktur, um mehr Übersichtlichkeit zu schaffen. Beim Exportieren des Projekts in eine WAR-Datei wird die spezifische Struktur der Web-Anwendung erstellt, wie sie auch in der Servlet-Spezifikation [SERVLET-SPEC] festgeschrieben ist.
(Java Resources: src) Dies ist der Ordner, unter dem die Java-Quellcode-Dateien abgelegt werden. Um das Beispiel analog zur vorherigen PA (SERVLET OHNE ECLIPSE) zu gestalten, wird hier auch ein Paket namens "wasa" erstellt.

(3) Beim Erstellen eines neuen Servlets sollte die effizientere Entwicklung von Servlets unter Eclipse deutlich werden.
(3.1) Das "FirstServlet" kann auch aus der vorherigen PA (SERVLET OHNE ECLIPSE) übernommen werden.

(4) Es wird ein Deployment-Descriptor benötigt. Dieser kann von Eclipse beim Erstellen des Projektes erzeugt werden und befindet sich in "WebContent/WEB-INF". Er wird verwendet, um den gewünschten URL-Pfad mit dem Servlet zu verlinken (siehe Code).
(4.1) Die in der vorigen PA (SERVLET OHNE ECLIPSE) erstellte Datei "web.xml" kann unter "WebContent/WEB-INF" abgelegt werden.
Alternativ lässt sich auch ein Deployment-Deskriptor automatisch von Eclipse erstellen oder es könnte eine neue Datei erstellt werden.

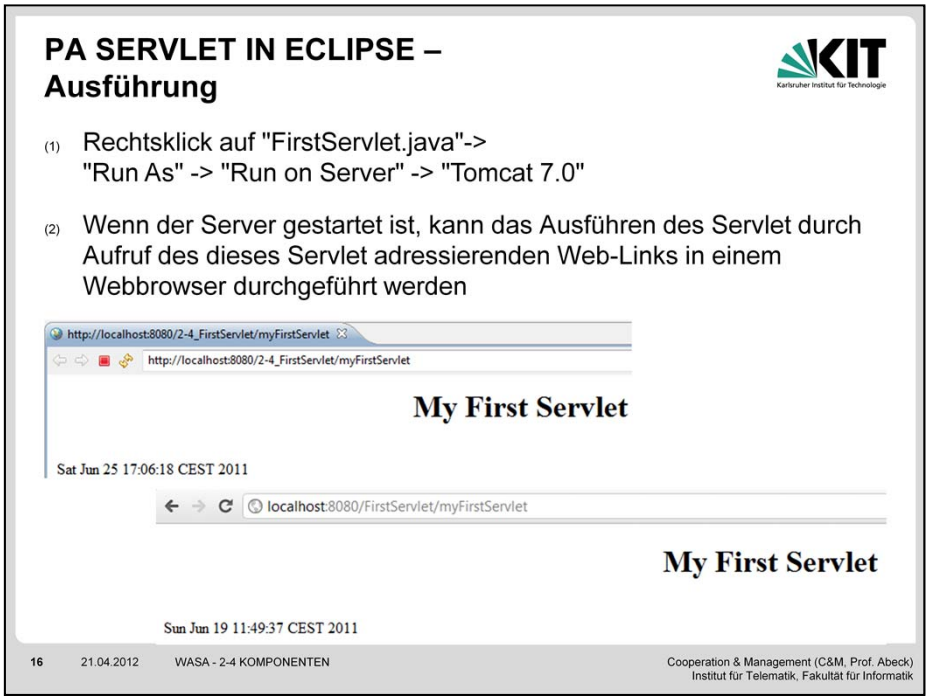
[ECLIPSE-DYN-WEB-PROJ] Eclipse Guide to Web Application Development http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.wst.webtools.doc.user/topics/twcresta.html
[SERVLET-SPEC] Java Servlet Specification. http://www.oracle.com/technetwork/java/javaee/servlet/index.html




Ein Java-Skelett des Servlets kann von Eclipse mit Hilfe von "File"->"New"->"Other" automatisch erstellt werden. Um ein einfaches Servlet zu erstellen, welches als Ausgabe nur "My First Servlet" und das aktuelle Datum enthält, reichen ein paar Zeilen Code aus. Man sieht, dass das Servlet die abstrakte Klasse "HttpServlet" implementiert. Außerdem werden "doGet()" und "doPost()" üblicherweise überschrieben. Die Eingabe von HTML-Code in den Ausgabestream ist offensichtlich etwas unschön. Daher wird meistens die Servlet-Technologie im Verbund mit JSP (Java Server Pages) eingesetzt.

(1) Eine ausführliche Beschreibung des Servlet-Quellcodes ist bereits in PA SERVLET OHNE ECLIPSE erfolgt, weshalb an dieser Stelle darauf verzichtet wird.

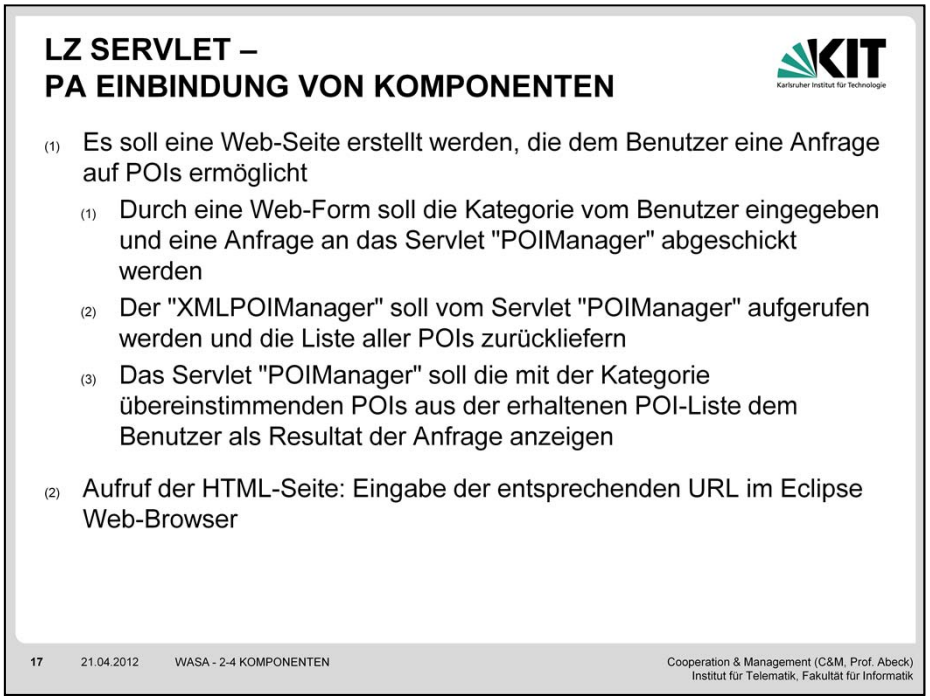
(@WebServlet, 5.) Die Möglichkeit, ein Servlet zu deklarieren, wird auch durch Annotationen angeboten. Die Deklaration des Servlets in der Konfigurationsdatei "web.xml" wird dadurch überflüssig. Die Annotation und die Konfigurationsdatei können aber zur selben Zeit existieren. Der Effekt von nicht übereinstimmenden Deklarationen des Servlet besteht darin, dass dasselbe Servlet von mehreren URLs gleichzeitig aufrufbar ist. Wenn in der "web.xml"-Datei eine andere Klasse für dieselbe URL wie in der Annotation deklariert ist, erhält das im Alphabet zuletzt stehende Servlet die URL. Die "web.xml"-Datei wird von Tomcat immer als erste geladen.




Die Ausführung aus Eclipse oder das manuelle Verteilen und Ausführen haben das selbe Resultat.

(1) Eclipse bietet die einfache Möglichkeit, eine Web-Ressource (wie z.B. das Servlet) direkt auszuführen. Dies erübrigt nicht nur das Eintippen der Adresse. Falls mehrere Server definiert wurden, lässt sich auch auswählen, auf welchem Server die Web-Ressource ausgeführt werden soll. Das gezielte Debuggen ist auch möglich, wenn die Web-Ressource mit dem Server im Debug-Modus gestartet wird.

(2) Der Server, der in Eclipse läuft, ist nicht auf die IDE eingeschränkt. Es kann daher zu Konflikten kommen, wenn Tomcat oder ein anderer Service (wie z.B. Skype o.ä.) den Port 8080 und 8009 besetzen. Diese sollten gestoppt werden. Das Service-Management in Windows kann durch Eingabe von "services.msc" im "Start"-Menü oder über den Task Manager (STRG+ALT+DEL) im Reiter "Services" genutzt werden.

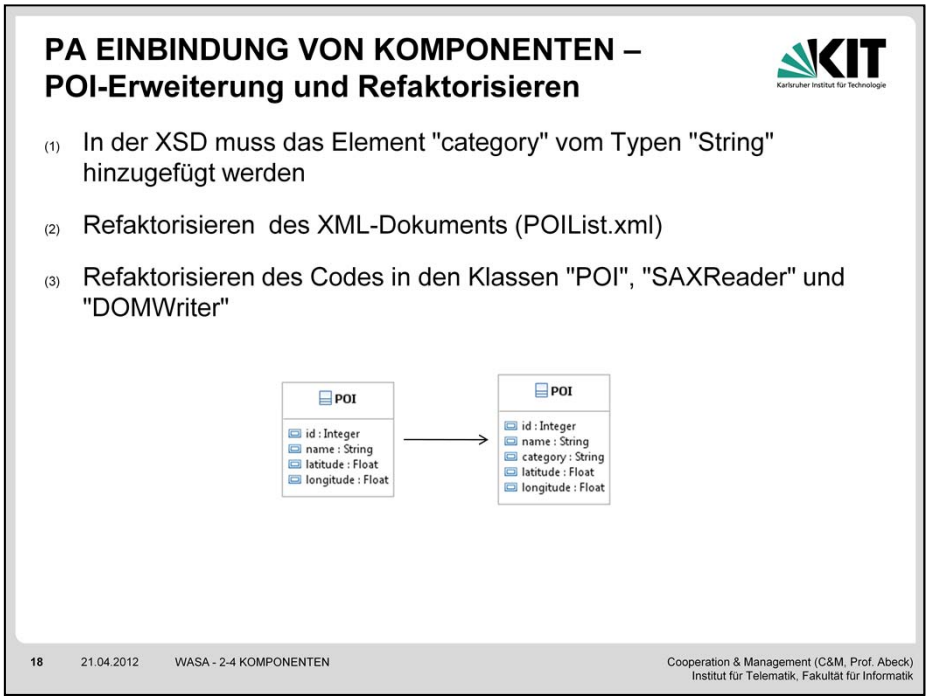



In dieser Praktischen Aufgabe (PA) wird die Entwicklung von Servlets am Beispiel des bereits aus den vorhergehenden Kurseinheiten bekannten POI-Management (PointOfInterest) aufgezeigt.

(1) Die nachfolgenden Schritte beschreiben die Funktionsweise und die wesentlichen Bestandteile der zu entwickelnden Anwendung.
(1.1) Das Servlet "POIManager" erhält die Anfrage nach einem POI.
(1.2) Ein wesentlicher Teil der Geschäftslogik der zu entwickelnden Web-Anwendung stellt die in der Kurseinheit INFORMATION entwickelte Komponente "XMLPOIManager" dar, durch die die Liste aller POIs bereitgestellt wird.
(1.3) Die Auswahl der aufgrund der in der Anfrage angegebenen Kategorie gewünschten POIs wird vom Servlet erbracht und stellt die eigentliche Intelligenz dieser Anwendung dar.

(2) Durch die Eingabe des entsprechenden Uniform Resource Locator (URL) ruft der Benutzer die Anwendung auf und kann sie über den Web-Browser bedienen.

| | |
|---|---|
| PA | Praktische Aufgabe |
| POI | PointOfInterest |
| URL | Uniform Resource Locator |



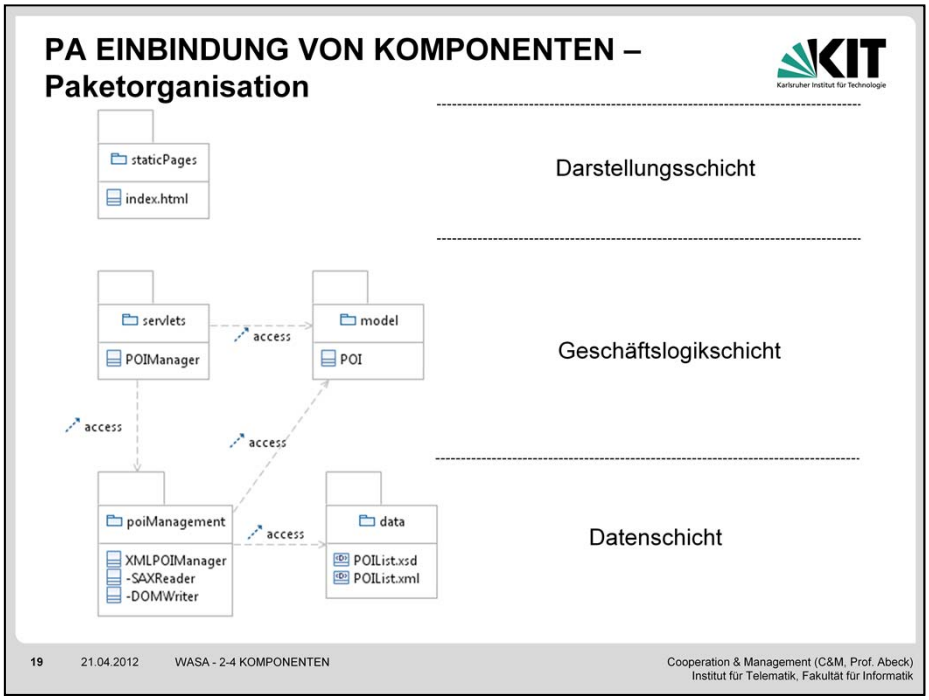

In Situationen, in denen eine strukturelle Änderung in der Anwendung erforderlich ist, müssen alle davon betroffenen Artefakte angepasst (refaktorisiert, umgestaltet) werden, um unerwartete Laufzeitfehler zu vermeiden. Eine Erweiterung der POI-Struktur ohne entsprechende Anpassung würde zu den folgenden Effekten führen:
(i) Der "SAXReader" liest im Fall der Änderung des POI-Artefakts (XSD- und XML-Dokumente) nur die bekannten Elemente und überspringt die ihm nicht bekannten Elemente (hier "category").
(ii) Der "POIManager" aber, der eine Anfrage auf die Kategorie eines POI macht, wird zur Laufzeit auf ein "null"-Objekt "category" vom Typ "String" stoßen und abstürzen (es sei denn, man testet sorgfältig auf nicht initialisierte Variablen).

(1) Eclipse erkennt, wenn die XSD-Datei verändert wurde und zeigt einen Fehler beim XML-Dokument "POIList.xml" an.

(2) Da die Schema-Definition verändert wurde, muss auch die POI-Liste angepasst werden. Mit Hilfe der XML-Editoren können die fehlenden Elemente einfach eingefügt werden.

(3) Da die Veränderung im Basisdatentyp geschieht, müssen alle von POI abhängigen Klassen verändert werden.
Automatische Vermeidung solcher Situationen ist durch den Mechanismus der Reflexion in Java möglich. In der Programmierung bedeutet Reflexion (engl. reflection) oder Introspektion, dass ein Programm seine eigene Struktur kennt und diese, wenn nötig, modifizieren kann.
Näheres zu Reflexion in Java findet man unter [JAVA-REFL]. Reflexion ist ein sehr fortgeschrittener Teil der Java-Sprache, der hier nicht weiter betrachtet.

[JAVA-REFL] Oracle, Java Tutorial on Reflection, http://download.oracle.com/javase/tutorial/reflect/index.html




[Ke09:181]
Pakete gruppieren Elemente und definieren Namensräume (engl. namespaces), in denen sich diese Elemente befinden.

(staticPages) Das "staticPages"-Paket enthält statische HTML-Seiten wie die Suchmaske "index.html".

(servlets) Das "servlets"-Paket enthält das "POIManager"-Servlet.

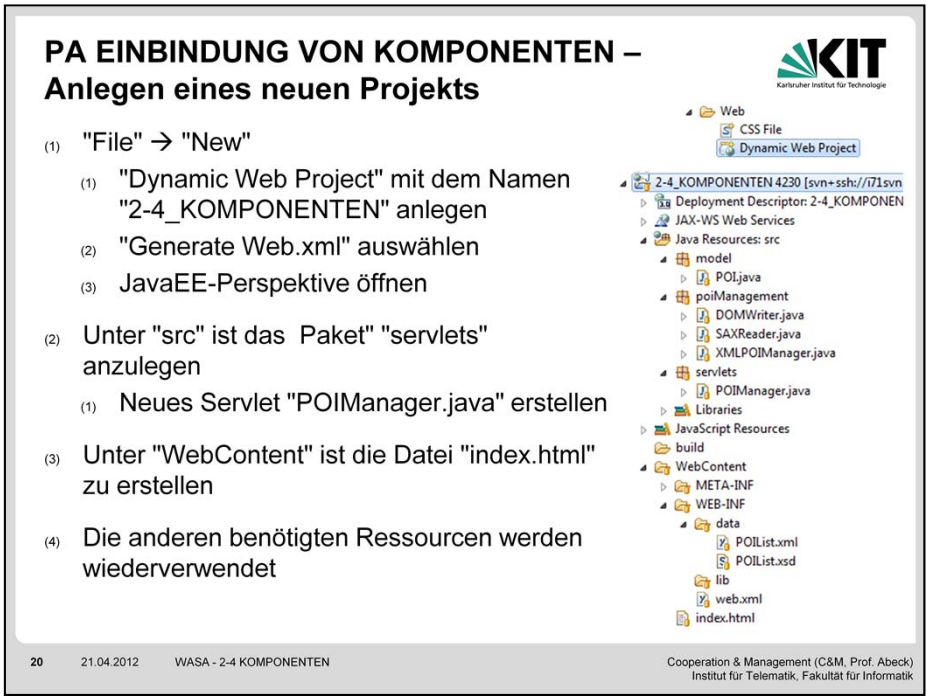
(model) Das "model"-Paket enthält die domänenspezifischen Klassen, hier die POI-Klasse.
Die in den Paketen "model" und "servlets" beinhalteten Artefakte bilden die Applikationsschicht.

(poiManagement, data) Die in diesen Paketen enthaltenen Klassen sind für die Daten verantwortlich und bilden die Datenschicht der Web-Anwendung.

(access) Ein Paket-Zugang (engl. package access) ist eine Beziehung, die alle Namen öffentlicher Elemente eines Pakets zum importierenden Paket als privat hinzufügt [:181].

(import) Ein Paket-Import (engl. package import) ist eine Beziehung, die alle Namen öffentlicher Elemente eines Pakets zum importierenden Paket als öffentlich hinzufügt [:181].
Hinweis: Java enthält nativ keine Unterstützung für die Beziehung, die mit dem Stereotypen <<import>> gekennzeichnet ist. Alle importierten Elementnamen sind nur im jeweils importierenden Paket verfügbar, was der <<access>>-Beziehung in der UML entspricht. Eine <<import>>-Beziehung muss daher in Java durch mehrere <<access>>-Beziehungen ersetzt werden [:182].

[Ke09] Christoph Kecher: UML2 Das umfassende Handbuch, 3. Auflage, Galileo Computing, 2009.




Zur besseren Übersicht im Workspace ist für jede Praktische Aufgabe ein neues Projekt mit der Namenskonvention "<Kurseinheitennummer>_<PA-Kurzbezeichnung>" anzulegen.

Der Unterschied zwischen einem statischen und einem dynamischen Web-Projekt besteht darin, dass im statischen Web-Projekt keine JSP-Seiten und Servlets erlaubt sind. Ein statisches Web-Projekt kann in ein dynamisches Web-Projekt durch "Convert to a Dynamic Web Project" vom "Project"-Menü konvertiert werden [ECLIPSE-DYN-WEB-PROJ].

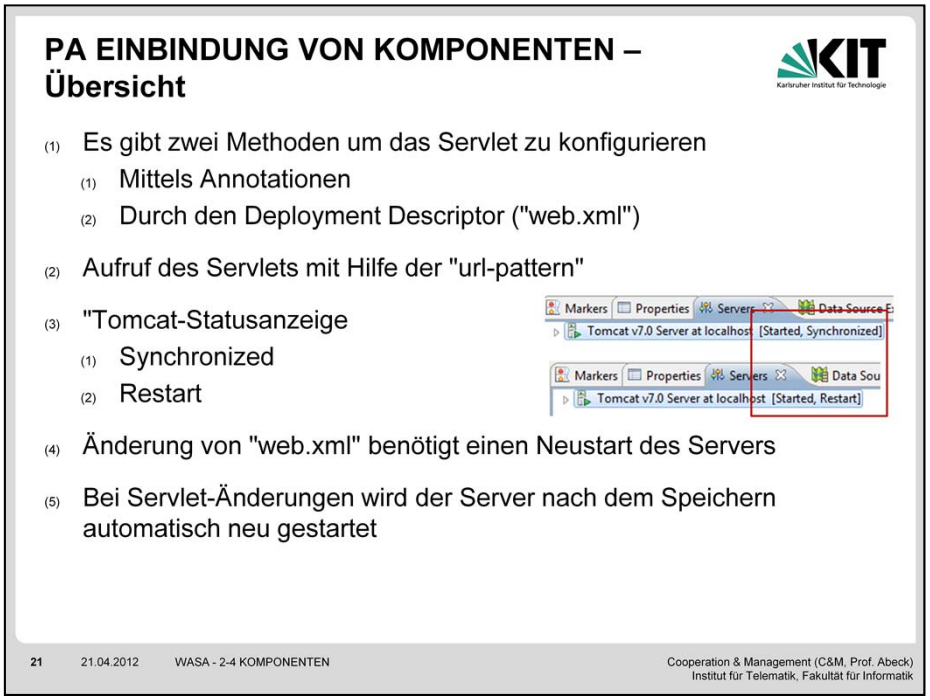
(1.1) Hier ist im Verzeichnis "Web" das "Dynamic Web Project" zu wählen. Es öffnet sich ein Wizard zur Erstellung des Projektes. Nach der Eingabe des Namen für das Projekt wird als "Target Runtime" der "Apache Tomcat 7.0 Server" ausgewählt. Anschließend kann der Wizard mit den Default-Einstellungen beendet werden.
(1.2) Die Möglichkeit einer automatischen Generierung der Konfigurationsdatei wird auch von Eclipse bereitgestellt.
(1.3) Die JavaEE-Perspektive beinhaltet die am meisten benutzten Sichten bei der Erstellung von Web-Anwendungen.

(2.1) Im "servlets"-Paket liegt das Servlet, das die Anwendungslogik erweitert und eine Schnittstelle für den Benutzer bereitstellt.

(3) Direkt unter dem "WebContent"-Verzeichnis werden Dateien abgelegt, die der Benutzer aufrufen können soll, wie z.B. die Suchmaske, die durch die statische Web-Seite "index.html" realisiert werden soll. Werden Dateien in den Ordner "META-INF" oder "WEB-INF" abgelegt, sind sie für den Benutzer nicht mehr zugänglich.

(4) Bei der Wiederverwendung muss sichergestellt werden, dass die POI-Kategorie korrekt von den importierten Ressourcen bearbeitet werden kann.

[ECLIPSE-DYN-WEB-PROJ] Eclipse Guide to Web Application Development
http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.wst.webtools.doc.user/topics/twcresta.html



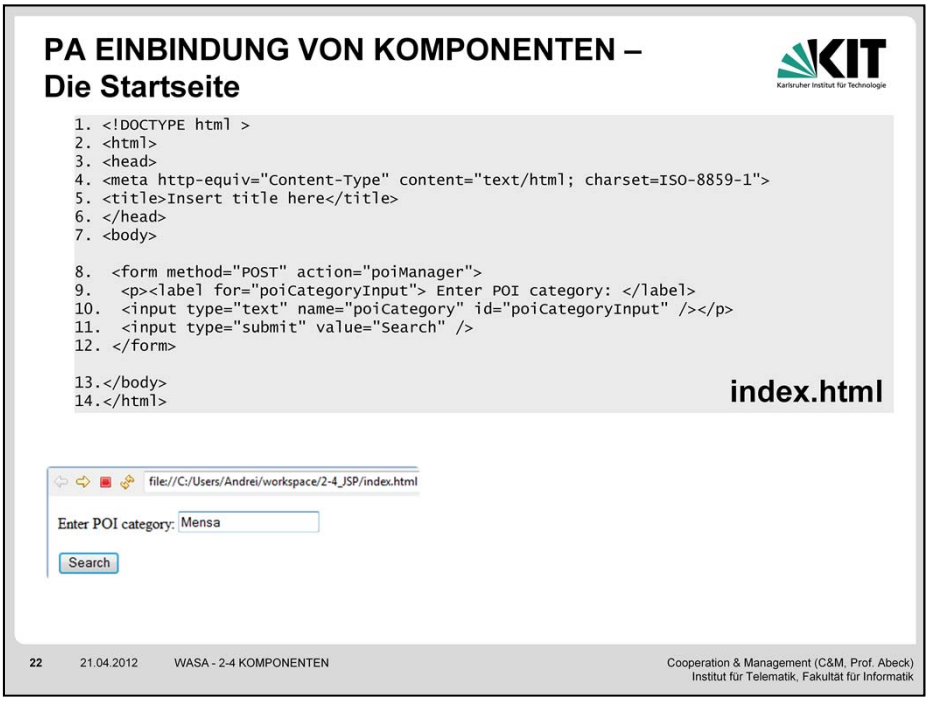

Diese Seite dient zur schnellen Übersicht über die Arbeitsweise mit Tomcat und Servlets.

(1.1) Das (ab der Version Java 1.5 unterstützte) Annotationssystem ist eine sehr elegante und einfache Weise, die Servlets für die Auslieferung (engl. deployment) zu konfigurieren. Annotationen in Servlets überschreiben die Servlet-Definitionen aus "web.xml", weil sie später von Tomcat geladen werden.
(1.2) Die Konfiguration kann weiterhin durch den Deployment-Deskriptor "web.xml" realisiert werden.

(2) Die in der Annotation oder im Deployment-Deskriptor angegebene URL kann benutzt werden, um das Servlet aufzurufen. Hierdurch wird die in "index.html" beschriebene Web-Form ausgeführt.

(3) Tomcat kann Veränderungen an den Dateien erkennen.
(3.1) Der Status "Synchronized" heißt, dass alle Dateien auf dem Dateisystem seit dem letzten Deployement nicht verändert wurden.
(3.2)Der Status "Restart" heißt, dass die Dateien verändert wurden und Tomcat einen Neustart benötigt, um die neuesten Änderungen wirksam zu machen.

(4, 5) Durch Doppelklick auf den Server in der Server-Sicht ist es möglich, die Einstellungen für das automatische Veröffentlichen (engl. publishing ) von Tomcat zu konfigurieren.



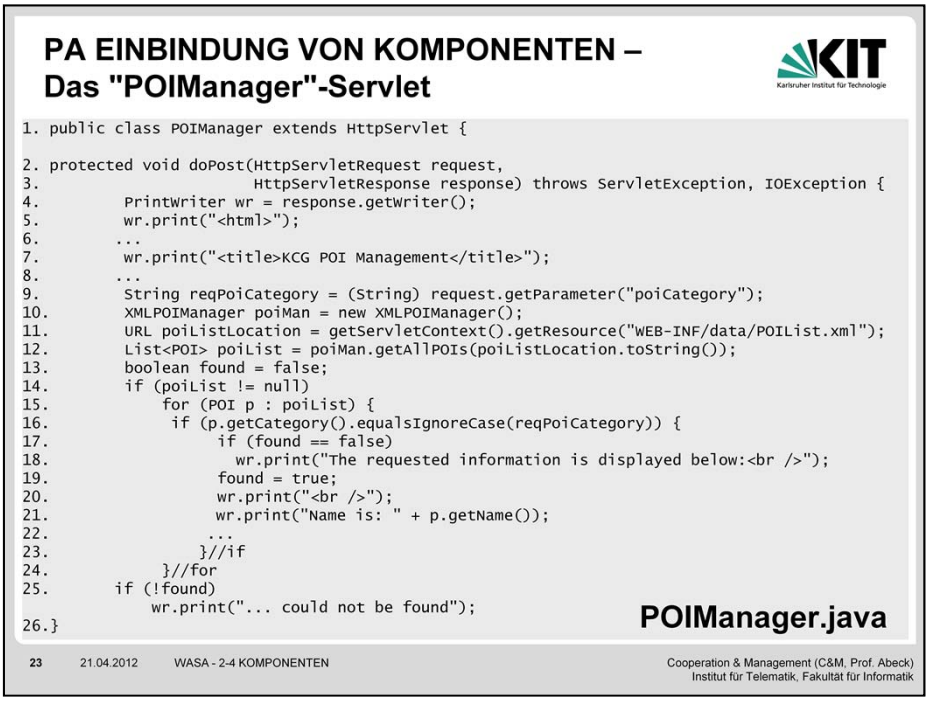

Die "index.html"-Datei realisiert die Suchmaske durch die ein Benutzer Anfragen an das Servlet leisten kann.

(1. – 7.) Standard HTML.

(8.) Deklaration der Form mit POST als HTTP-Transferoption und Weiterleitung auf das POIManager-Servlet. Das "method"-Attribut spezifiziert die Übertragungsmethode. Das "action"-Attribut hat den Wert des URL-Patterns dass in der Annotation oder in der "web.xml"-Datei deklariert wurde.

(10.) Diese HTML-Zeile generiert ein Eingabefeld für die POI-Kategorie. Der Wert des "name"-Attributs wird benutzt um die Daten zu identifizieren. Im Servlet kann man die Daten aus der POST-Anfrage entnehmen.

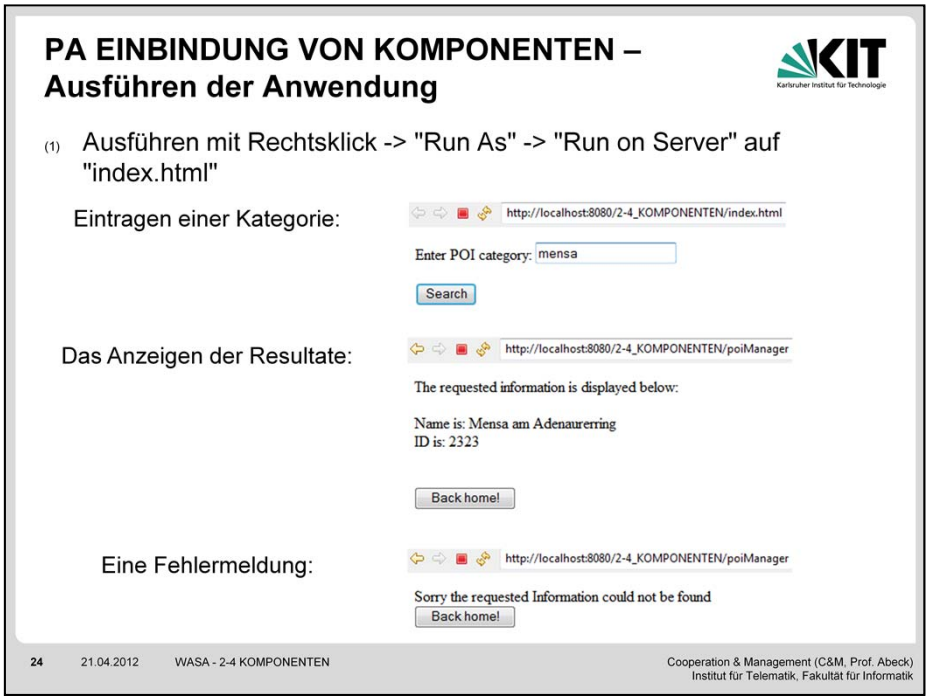
(11.) Diese HTML-Zeile generiert den Submit-Knopf der den Browser veranlasst eine POST-Anfrage and das Servlet, das im Attribut "action" des "form"-Elements adressiert wird, zu schicken.




Das Servlet antwortet auf HTTP-POST-Anfragen und liefert eine Web-Seite zurück mit den angefragten Informationen.
Der hier präsentierte Code ist vereinfacht um die Funktion zu veranschaulichen. Man muss auf Variablen mit Wert "null" achten, wie z.B.: wenn im Request kein Parameter namens "poiCategory" existiert wird "null" zurückgegeben. In der, zur Verfügung gestellten, Quellcodedatei zur PA sind Fehlerfindungsmechanismen eingebaut.

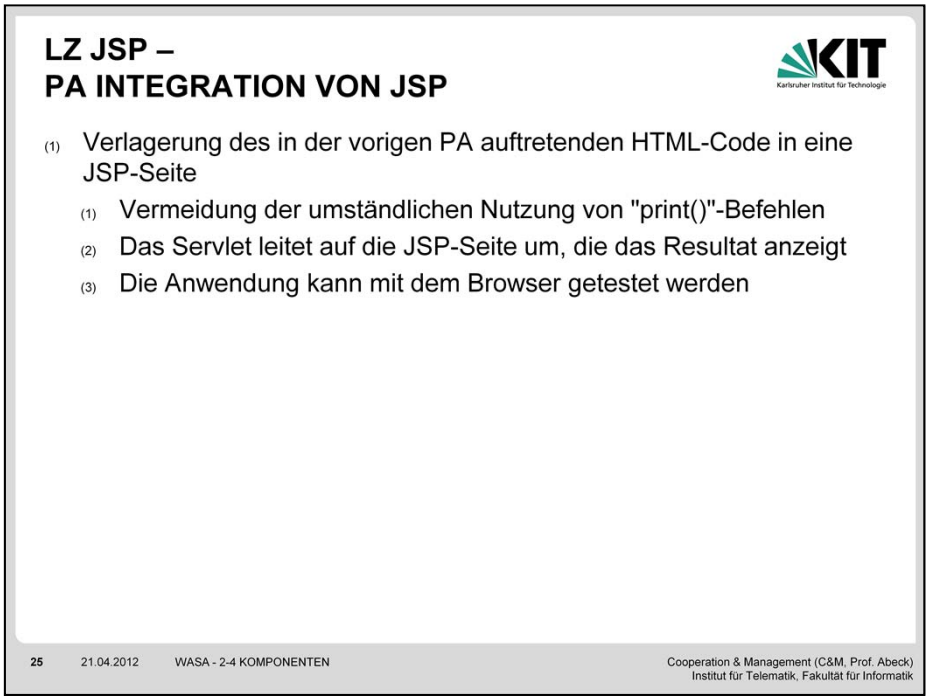
(2. ...doPost) Die Methode, die die POST-Anfrage entgegennimmt.
(4. PrintWriter) Der "PrintWriter" wird benutzt, um in die HTML-Seite zu schreiben, die dem Benutzer als Antwort zurückgeliefert wird.
(5. wr.print) Das "html"-Tag wird als String in den "PrintWriter" geschrieben. Das gleiche Vorgehen wird bei den anderen HTML-Tags verfolgt.
(9. String reqPoiCategory) Der Parameter "poiCategory", der von der HTML-Form in der POST-Anfrage geschickt wird, kann durch seinen Namen, der ihm in der HTML-Form verliehen wurden, aus dem "request"-Objekt gelesen werden. In diesem Fall handelt es sich um die Kategorie, nach der die Suche stattfinden wird.
(10.) Eine Instanz der "XMLPOIManager"-Klasse wird erstellt, um das Abrufen der POI-Informationen aus der XML-Dateien delegieren zu können.
(11.) Mit Hilfe des Servlet-Kontext kann man mit der "getResource()"-Methode auf alle Ressourcen unter der Wurzel der Anwendungsstruktur zugreifen. Würde man zum Ressourcenladen die "getResource()"-Methode über den Classloader mit "this.getClass().getResource()" zugreifen könnte man nicht auf den "WEB-INF"-Ordner zugreifen. Mehr kann man unter [TOMCAT-CLASS-LOADER] finden. Die Motivation für die "getResource()"-Methode ist, dass der Pfad der zu den Ressourcen führt, nicht Hartkodiert sein muss, da er je nach Deployement anders sein kann.
(12.) Nach dem der Pfad zur Ressource, mit der "getResource()"-Methode ermittelt wurde kann die Arbeit an den XMLPOIManager delegiert werden. Als Resultat wird eine POI-Liste zurückgeliefert.
(13.) Die boolsche Variable "found" wird benutzt um zu unterscheiden ob die Information erfolgreich ausgegeben werden kann.
(14.) Beispiel eines "if"-Statements zur Überprüfung des Objekts um sicherzustellen, dass das Program nicht Abstürzt, so wird eine NullPointerException vermeidet, wenn das Objekt auf keine Liste zeigt, i.e. nicht initialisiert ist.
(15.) Mit Hilfe einer Java for-each-Schleife wird durch die Liste iteriert.
(16.) Vergleich zwischen der Kategorie des, in der Schleife, aktuellen POIs, mit der, vom Benutzer in der HTML-Form gesendeten Kategorie. Auf Groß- und Kleinschreibung wird nicht geachtet.
(17.-18.) Um eine benutzerfreundliche Ausgabe zu generieren wird einmalig, beim ersten passenden POI, eine Informationszeile ausgegben. Man beachte, dass um einen Zeilenumbruch zu generieren das HTML-Tag "<br />" benutzt werden muss, die "\n"-Sequenz wird nicht als Zeilenumbruch ausgegeben.
(19.-22.) Bei finden eines passenden POIs wird dieser, als HTML formatiert, durch die "print()"-Methode des PrintWriter ausgegeben.
(25.) Wenn kein passendes POI gefunden wurde, dann hat die "found"-Variable den Wert "false", weshalb in diesem Fall eine Fehlernachricht ausgegeben wird.

[TOMCAT-CLASS-LOADER] Der Tomcat Class-Loader, http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html




(1) Durch die angegebene Befehlsfolge lässt sich die Web-Anwendung aus Eclipse starten.

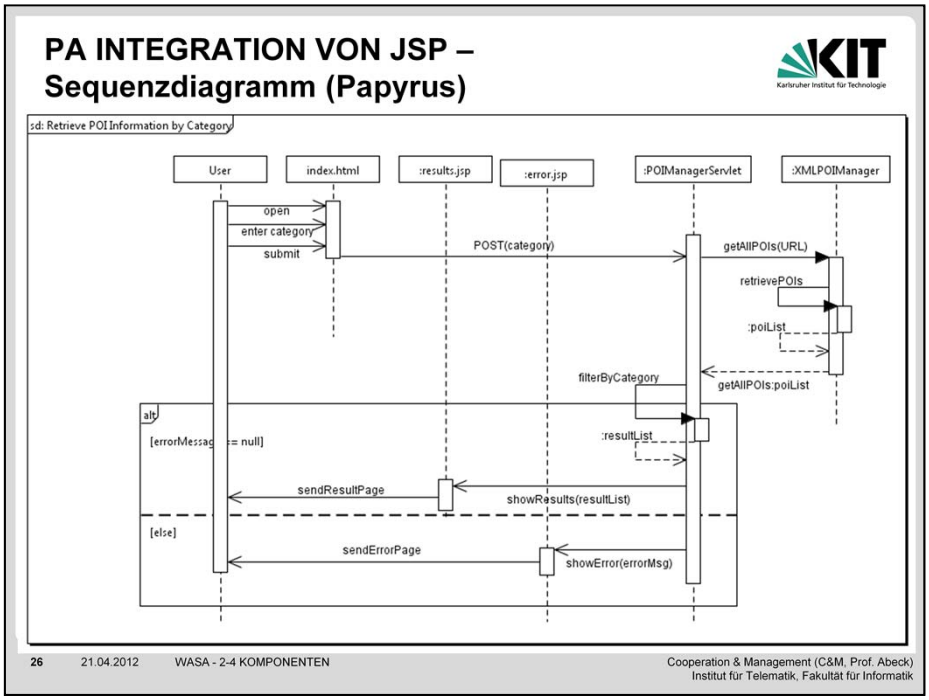
Die Bildschirmausschnitte zeigen beispielhaft den korrekten bzw. fehlerhaften Ablauf der Anwendung auf.




(1) (1.1) Die Formatierung der Web-Seite wird schnell kompliziert, wenn sie in Servlets als "print()"-Statements realisiert wird. Die JSP-Technologie verschafft hier Abhilfe.

(1.2) Das Zusammenspiel zwischen Servlets und JSPs wird nachfolgend in einem Sequenzdiagramm aufgezeigt.

(1.3) Der Aufruf erfolgt analog zu der Servlet-Lösung.




Die Abfolge der Interaktionen zwischen den Artefakten und dem Benutzer wird anhand eines Sequenzdiagramms gezeigt. Die UML2-Tools sind leider nicht genügend ausgereift, um gute Sequenzdiagramme zu erstellen. Deshalb wird in diesem Beispiel ein Diagramm benutzt, dass in Eclipse mit dem Papyrus-Diagramm-Editor-Plug-In erstellt wurde.

(User) Der Benutzer ruft die Startseite "index.html" des "POI-Management"-Services des KITCampusGuide auf.

(index.html) Dem Benutzer wird die Möglichkeit angeboten, nach POIs einer bestimmten Kategorie zu suchen. Mit Hilfe eines Web-Formulars (engl. web form) wird der vom Benutzer eingegebene Suchbegriff an das "POIManager"-Servlet weitergeleitet.

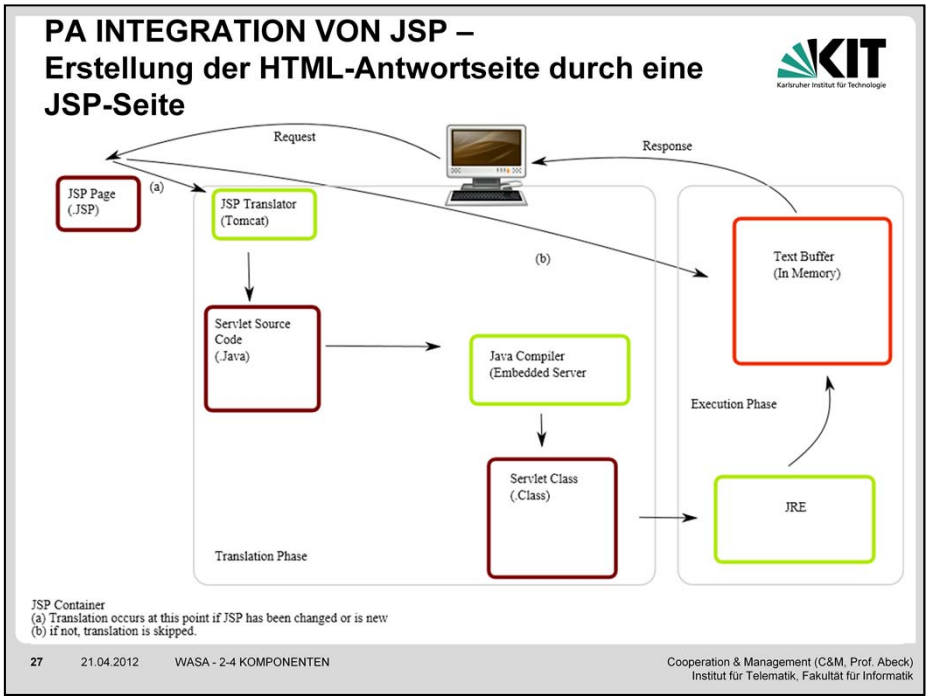
(POIManager) Das Servlet erhält die POST-Anfrage und macht einen Aufruf an den "XMLPOIManager", um alle POIs zu erhalten. Nachdem Erhalt der POI-Daten wird eine Filterung der POIs gemäß der Kategorie durchgeführt. Die Informationen der POIs mit passender Kategorie werden in der Session gespeichert und mit einem "redirect" auf die "results.jsp"-Seite gelenkt. Für den Fall, dass ein Fehler auftritt, wird die Variable "errorMessage" in die Session gespeichert und mit einem "redirect" auf die "results.jsp"-Seite gelenkt.

(XMLPOIManager) Die "XMLPOIManager"-Komponente wurde in den vorherigen PAs besprochen und wird hier als "Black-Box"-Komponente betrachtet ,die eine Liste aller aktuell im Datenbestand vorliegenden POIs liefert.

(error.jsp) Eine JSP-Seite, die die Fehlernachricht nach dem Auftreten eines Bearbeitungsfehlers anzeigt.

(results.jsp) JSP-Seite, die dem Benutzer die Resultate darstellt.

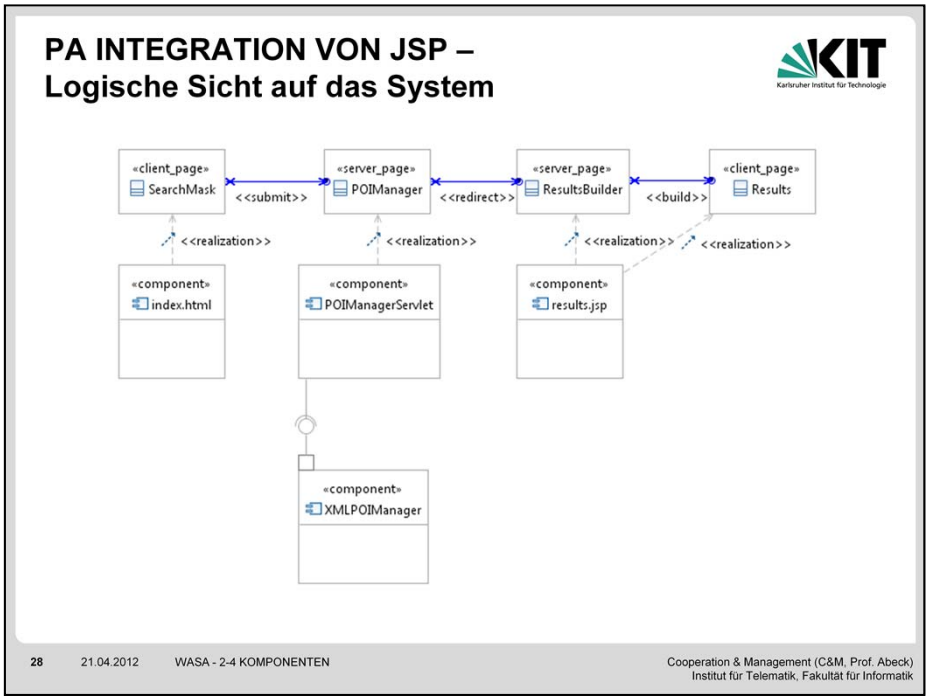
(alt – Alternative Folge) Eine alternative Abfolge, die in einem Sequenzdiagramm mit Hilfe eines kombinierten Fragments dargestellt wird.
(alt[errorMessage==null) Wenn die Variable "errorMessage" den Wert "null" hat, wird das "POIManagerServlet" den Fluss an die JSP-Seite "results.jsp" weiterleiten, um dem Benutzer die Resultate anzuzeigen. Als Parameter wird die gefilterte POI-Liste mitgegeben.
(alt[else]) Für den Fall, dass die Variable "errorMessage" mit irgend einer Fehlermeldung initialisiert ist, wird der Fluss an die JSP-Seite "error.jsp" weitergeleitet ,die dem Benutzer eine Fehlermeldung anzeigt.




(a) Dieser Ausführungsweg wird genommen, wenn eine Übersetzung der JSP-Seite (weil neu oder geändert) erforderlich ist.
(a.1) Die JSP-Seite wird von Tomcat in ein Java-Servlet übersetzt. Das aus einer JSP generierte Servlet ist identisch mit einem normalen Servlet. Der große Nutzen ist, dass dem Programmierer der Aufwand erspart wird, die umständlichen und schlecht wartbaren "print"-Befehle zu schreiben.
(a.2) Nach der Übersetzung wird der Weg eines normalen Servlets eingeschlagen, indem zunächst eine Übersetzung in eine ".class"-Datei und anschließend die Ausführung in der JRE erfolgt .

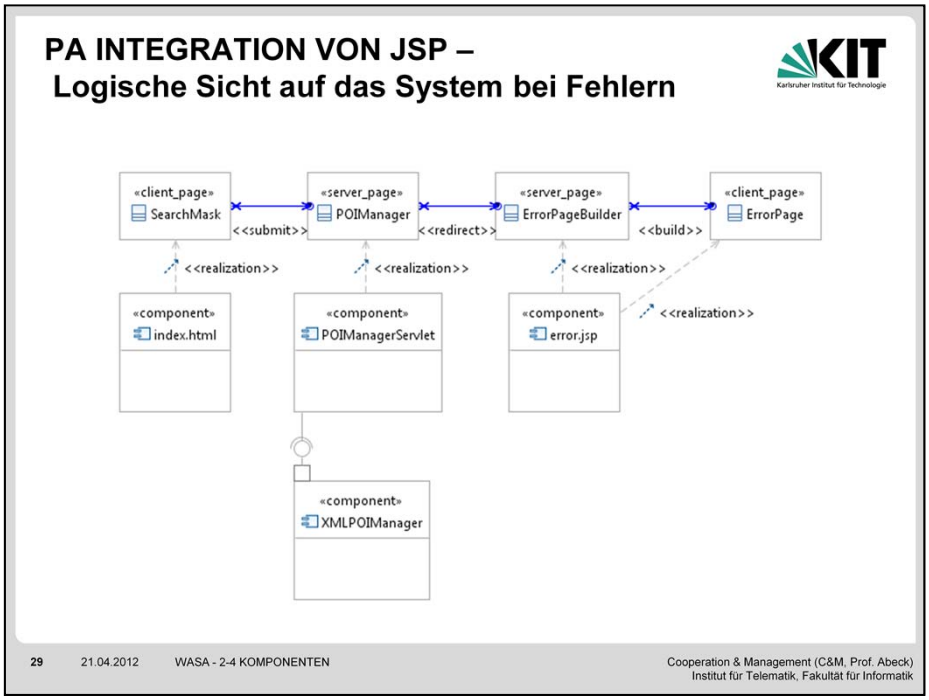
(b) Wenn die JSP-Seite schon einmal übersetzt wurde und z.B. im Cache liegt, wird sie direkt verwendet und nicht neu übersetzt. Dies ist ein wichtiges Merkmal, da man sich in einer realen Anwendung nicht leisten kann, bei jeder Anfrage den gesamten Weg von der Quelldatei zum Resultat zu durchlaufen.

Quelle: Wikimedia commons, http://upload.wikimedia.org/wikipedia/commons/0/03/JSPLife.svg




(submit) Mit dem "submit"-Stereotypen einer gerichteten Assoziation kann man darstellen, dass eine Seite eine Anfrage an eine andere Seite stellt. Hier wird das Senden der POI-Kategorie von der HTML-Form zu dem Servlet modelliert.
(build) Mit dem "build"-Stereotypen einer gerichteten Assoziation kann man darstellen, dass eine Seite eine andere Seite aufbaut. Diese Assoziation kann nur von einer Server-Page zu einer Client-Page gezogen werden, da nur Server-Pages dynamische Elemente besitzen. Hier wird die Client-Page "Results" von der Server-Page "POIManager" benötigt.
(realization) Mit einer "realization"-Beziehung lässt sich darstellen, welche Komponente für welche der Seiten verantwortlich ist und sie erzeugt.
In den UML2-Tools wird die "realization" nicht unterstützt, weshalb ein "realization"-Stereotyp auf Basis einer Dependency erstellt wurde.
(redirect) Mit dem "redirect"-Stereotypen kann man eine Umleitungsbeziehung darstellen. Der POIManager leitet auf die "ResultsBuilder"-Server-Page um, die von der "results.jsp"-Komponente realisiert wird.
(client-page, SearchMask) Diese Seite wird dem Benutzer angezeigt und wird mit dem Stereotypen <<client_page>> gekennzeichnet.
(component, index.html) Die Datei index.html ist eine statische HTML-Seite, die für die Generierung der "SearchMask"-Client-Page. verantwortlich ist.
(component, POIManagerServlet) Diese Komponente ist ein Servlet, gekennzeichnet durch den "Servlet"-Suffix.
(component, results.jsp) Diese Komponente leistet das Anzeigen der Resultate indem sie eine Server-Page realisiert die durch die Liste der POIs iteriert und für den Benutzer eine HTML-Seite, die Client-Page "Results" aufbaut.
(client-page, Response) Dies ist die von der Server-Page "POIManager" generierte Antwort auf die Anfrage des Benutzers, die Information als HTML darstellt .
(server_page, POIManager) Die Server-Page "POIManager" enthält den dynamischen Teil, die Ausführbare Logik und erstellt die Antwort der Anfrage in Form der Client-Page "Results".
(server_page, POIManager) Die Server-Page "POIManager" enthält den dynamischen Teil, die ausführbare Logik und erstellt die Antwort der Anfrage in Form der Client-Page "Results".
(XMLPOIManager) Die Komponente XMLPOIManager bietet eine Schnittstelle für den Zugriff auf POI-Daten. Dieses Verhalten wird mit der "Ball-Socket"-Notation dargestellt.

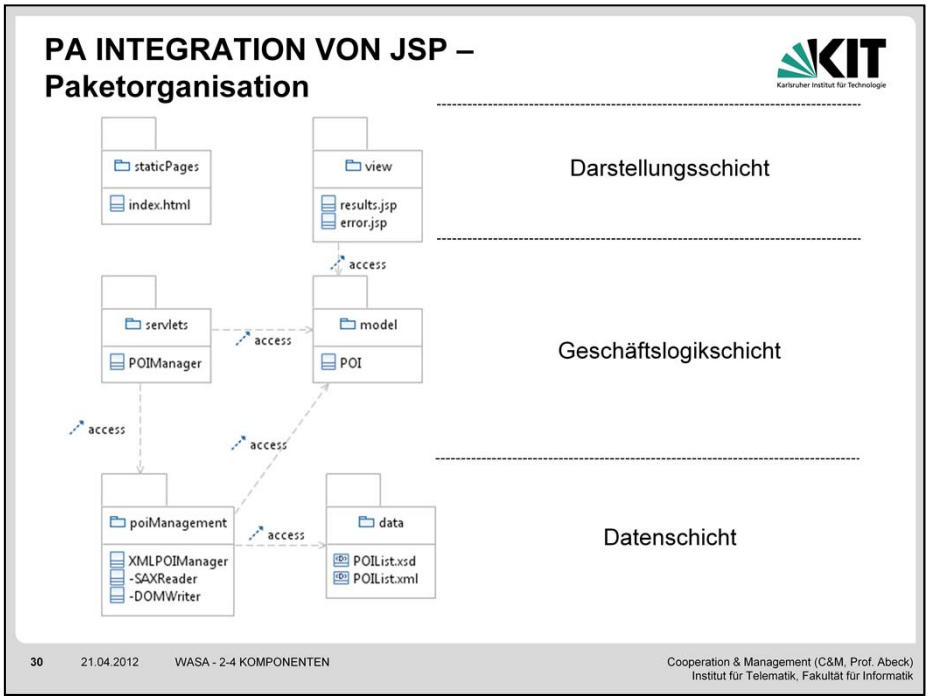
[Ke09] Christoph Kecher: UML2 Das umfassende Handbuch, 3. Auflage, Galileo Computing, 2009.
[Co00] Conallen Jim: Building Web Applications with UML, Addison-Wesley Longman, 2000.




In einem Fehlerfall wird das Servlet nicht auf die "results.jsp"-Seite, sondern auf eine Fehlermeldungs-Seite umleiten. So ist auch das Diagramm übersichtlicher und der Programmierer kann die Komplexität des Codes der "results.jsp"-Seite reduzieren.

(redirect) Mit dem "redirect"-Stereotyp lässt sich eine Umleitungsbeziehung darstellen. Der "POIManager" leitet auf die "ErrorPageBuilder"-Server-Page um, die von der "error.jsp"-Komponente realisiert wird.

(component, error.jsp) Diese Komponente leistet das Anzeigen der Fehlermeldung.




Pakete gruppieren Elemente und definieren Namensräume (engl. namespaces), in denen sich diese Elemente befinden.

(import) Ein Paket-Import (engl. package import) ist eine Beziehung, die alle Namen öffentlicher Elemente eines Pakets zum importierenden Paket als öffentlich hinzufügt [Ke09:181].

(access) Ein Paket-Access (engl. package access) ist eine Beziehung, die alle Namen öffentlicher Elemente eines Pakets zum importierenden Paket als privat hinzufügt [Ke09:181].
Hinweis: Java enthält nativ keine Unterstützung für die Beziehung, die mit dem Stereotypen <<import>> gekennzeichnet ist. Alle importierten Elementnamen sind nur im jeweils importierenden Paket verfügbar, was der <<access>>-Beziehung in der UML entspricht. Eine <<import>>-Beziehung muss daher in Java durch mehrere <<access>>-Beziehungen ersetzt werden [Ke09:182].

(staticPages) Das "staticPages"-Paket enthält statische HTML-Seiten, wie beispielsweise die Suchmaske "index.html".

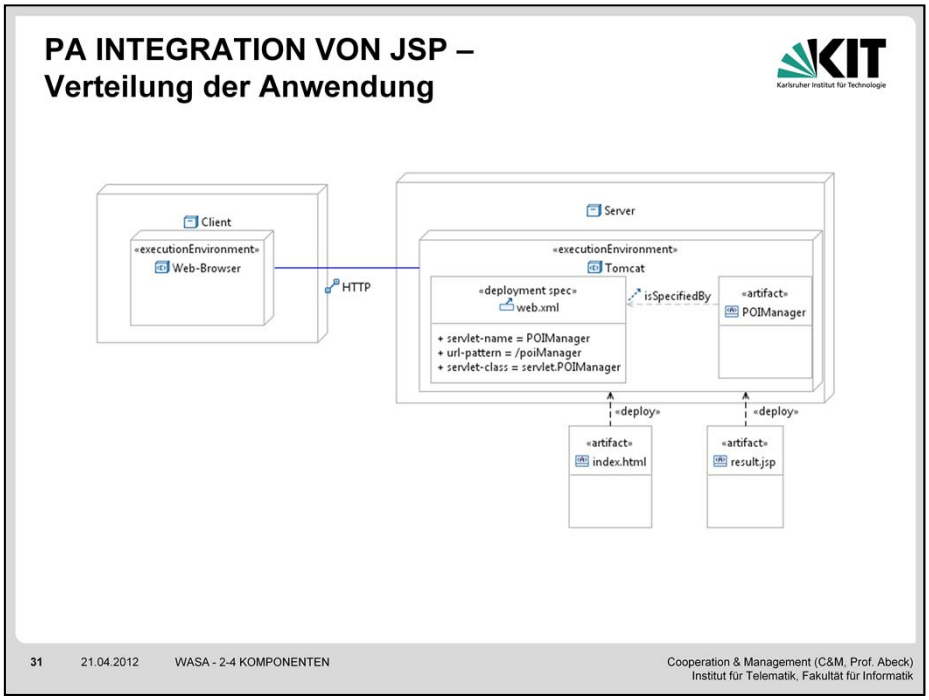
(servlets) Das "servlets"-Paket enthält das "POIManager"-Servlet.

(model) Das "model"-Paket enthält die domänenspezifischen Klassen, hier die "POI"-Klasse.
Die in den Paketen "model" und "servlets" beinhalteten Artefakte bilden die Geschäftslogikschicht.

(poiManagement, data) Die in diesen beiden Paketen enthaltenen Klassen sind für die Daten verantwortlich und bilden die Datenschicht der Web-Anwendung.

(view) Das "view"-Paket enthält die Servlets, die die Anzeige der nach Kategorie gefilterten POI-Liste übernimmt. Im Normalfall wird "results.jsp" die Anzeige übernehmen, im Fehlerfall "error.jsp".

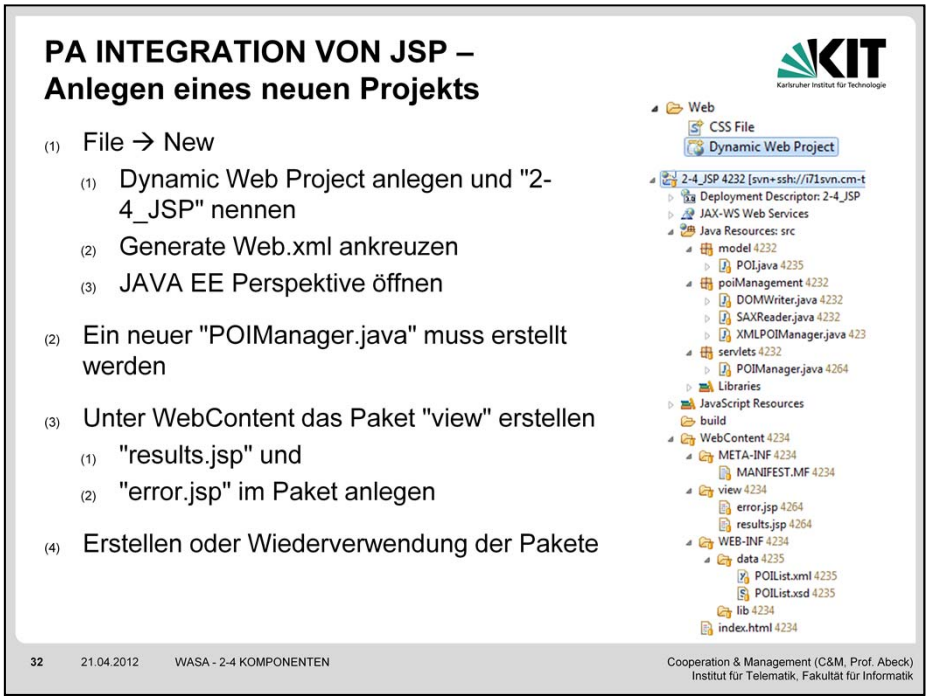
[Ke09] Christoph Kecher: UML2 Das umfassende Handbuch, 3. Auflage, Galileo Computing, 2009.




Standardmäßig werden Java-Komponenten zusammen mit Tomcat in einer JVM ausgeführt [TOMCAT-UG]. Dies bedeutet, dass auch die "poiManagement"-Komponente, der XMLPOIManager und die Helferklassen zusammen mit dem "POIManager"-Servlet ausgeführt wird. Der Übersichtlichkeit halber wird die "poiMangement"-Komponente nicht dargestellt.

(Client) Der Client-Knoten mit einem Web-Browser als Ausführungsumgebung. Die Ressourcen werden mit Hilfe des Web-Browsers aufgerufen.
(Tomcat) Tomcat ist die Ausführungsumgebung der Servlets. Die Ausführungsumgebung wird mit Hilfe des Stereotyps <<executionEnvironment>> dargestellt [Ke09:161].
(HTTP) Ein Kommunikationspfad(Communication Path) der durch das HTTP-Protokoll die Kommunikation des Web-Browsers mit dem Web-Container Tomcat ermöglicht.
(web.xml) Die Spezifikation für das Deployement der Artefakte wie z.B. das POIManager-Servlet. Die Konfigurationsdaten des POIManager-Servlets werden, bei der Implementierung, in der Deployement-Spezifikation web.xml eingetragen.
(isSpecifiedBy) Leider wird die graphische Notation der "Specification link" nicht durch den Editor unterstützt und als Workaround wurde eine Abhängigkeit mit dem Stereotypen <<isSpecifiedBy>> benutzt. Dies ist eine Beziehung zwischen der <<deployement spec>> und dem zu deployten Artefakt, das POIManager Servlet.
(POIManager) Der POIManager ist das Servlet, dass in Tomcat ausgeführt wird und die gewünschte POI-Funktionalität bereitstellt. Die Artefakte werden mit Hilfe des Stereotyps <<artifact>> dargestellt [Ke09:161].
(index.html) Dies ist das Frontend der Web-Anwendung und läuft auch auf dem Tomcat-Server. Die Darstellung unterscheidet sich von der des POIManager um eine andere Beziehung im UML-Deployement-Diagramm zu veranschaulichen.
(results.jsp) Das Zentrale Artefakt dieser PA ist die "results.jsp"-Datei. Sie wird wie Servlets oder HTML-Seiten in der Tomcat-Umgebung ausgeführt.
(deploy) Diese Beziehung sagt aus, dass das Artefakt am Ende des Pfeils in der Ausführungsumgebung an der Spitze laufen wird.

[TOMCAT-UG] Tomcat - A Minimalistic User's Guide, http://www.jajakarta.org/tomcat/tomcat3.2-4.0/tomcat-3.2.3/doc/uguide/tomcat_ug.html
[Ke09] Christoph Kecher: UML2 Das umfassende Handbuch, 3. Auflage, Galileo Computing, 2009.




Um eine bessere Übersicht im Workspace zu haben, ist für jede Praktische Aufgabe ein neues Projekt mit der Namenskonvention "<Kurseinheitennummer>_<PA-Kurzbezeichnung>" anzulegen.

[ECLIPSE-DYN-WEB-PROJ]
Der Unterschied zwischen einem statischem Web-Projekt und einem dynamischen besteht darin, dass im statischen Web-Projekt keine JSP-Seiten und Servlets erlaubt sind. Ein statisches Web-Projekt kann in ein dynamisches Web-Projekt, durch "Convert to a Dynamic Web Project" vom "Project"-Menü, konvertiert werden.

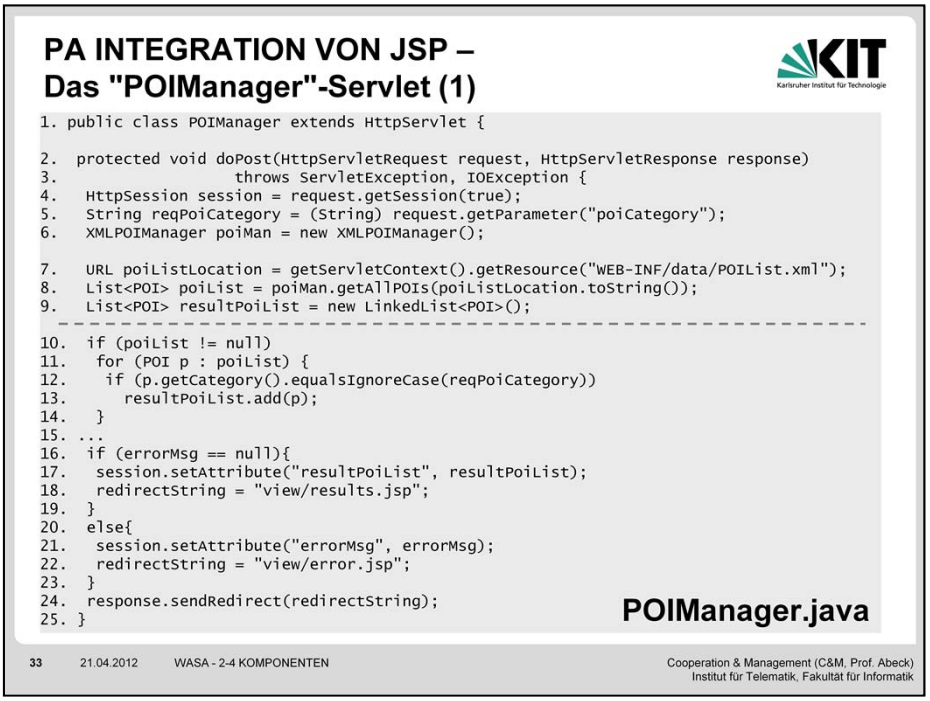
(1) In Eclipse wird im "Project-Explorer" mit "Rechtsklick"->"New"->"Project" den Dialog "New Project" geöffnet. Hier wird im Verzeichnis "Web" das "Dynamic Web Project" gewählt. Es öffnet sich nun ein Wizard zur Erstellung des Projektes. Nachdem der Name für das Projekt eingegeben wurde, wird als "Target Runtime" der "Apache Tomcat 7.0 Server" ausgewählt. Anschließend kann der Wizard mit den Default-Einstellungen beendet werden.
(1.2) Die Möglichkeit einer automatischen Generierung der Konfigurationsdatei wird auch von Eclipse bereitgestellt.
(1.3) Die Java-EE Perspektive beinhaltet die am meisten benutzten Sichten bei der Erstellung von Web-Anwendungen.

(2) Der POIManager unterscheidet sich von der PA "EINBINDUNG VON KOMPONENTEN", da die Anzeige, also das Erstellen der Resultatseite, von einer JSP-Seite übernommen wird.

(3) Da die Seiten mit dem Resultat der Anfrage dem Benutzer angezeigt werden sollen´, werden die JSP-Seiten unter "WebContent" abgelegt.
(3.1) Die "results.jsp"-Seite übernimmt im Normalfall die Anzeige der Resultate.
(3.2) Die "error.jsp"-Seite übernimmt im Fehlerfall die Anzeige der Fehlermeldung.

(4) Bei der Wiederverwendung muss man sicherstellen, dass die POI-Kategorie korrekt von den importierten Ressourcen bearbeitet werden kann.

[ECLIPSE-DYN-WEB-PROJ] Eclipse Guide to Web Application Development http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.wst.webtools.doc.user/topics/twcresta.html

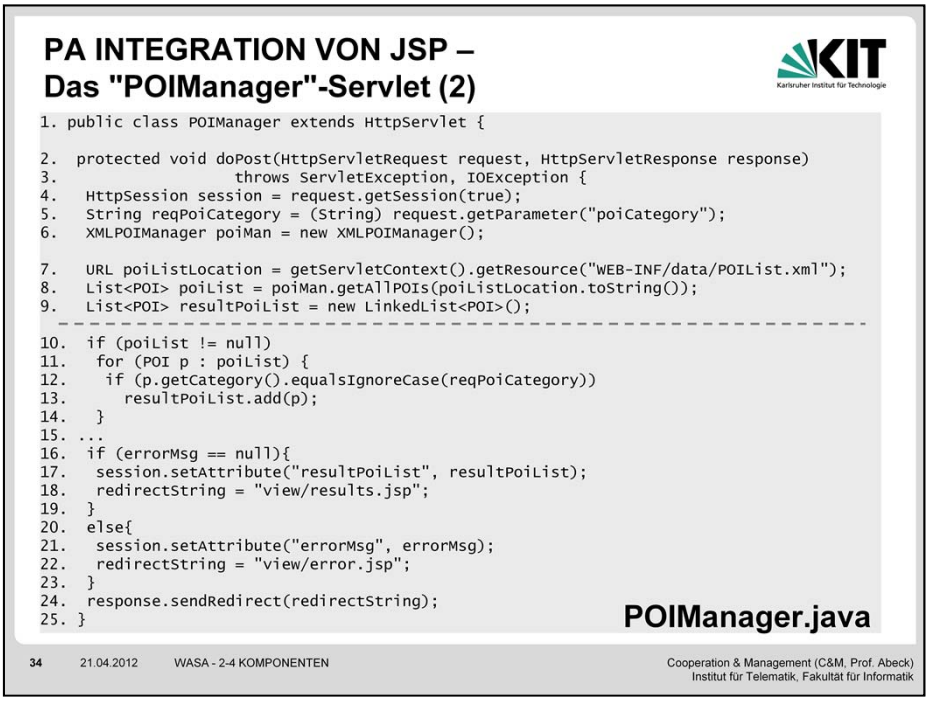



Das Servlet antwortet auf HTTP-POST und liefert eine Web-Seite zurück mit den angefragten Informationen. Der hier präsentierte Code ist vereinfacht um die Funktion zu veranschaulichen. Man muss auf Variablen mit Wert "null" achten; beispielsweise wird "null" zurückgegeben, falls im Request kein Parameter namens "poiCategory" existiert. In der zur Verfügung gestellten Quellcodedatei zur PA sind Fehlerfindungsmechanismen eingebaut.

(3. ... doPost) Die Methode ,die die POST-Anfrage entgegennimmt.
(4. ... session) Das Session-Objekt kann man sich als Wörterbuch vorstellen in denen Daten mit einem Schlüssel-Wert Paar, persistent, gespeichert werden. Das Session-Objekt wird daher benutzt um die Kommunikation zwischen dem Servlet und der JSP-Seite zu ermöglichen, da das Session-Objekt für alle Servlets und JSP-Seiten erreichbar ist. Es gibt ein Session-Objekt pro Benutzer und es wird nach einer bestimmten Zeit von Inaktivität gelöscht.
(5. ... reqPOICategory) Der Parameter "poiCategory" der von der HTML-Form in der POST-Anfrage geschickt wird kann durch seinen Namen, der ihm in der HTML-Form verliehen wurden, aus dem "request"-Objekt geholt werden. Dies ist nun die Kategorie nach der die Suche stattfinden wird.
(6. XMLPOIManager ...) Eine Instanz der XMLPOIManager-Klasse wird erstellt, um das Abrufen der POI-Informationen aus der XML-Dateien delegieren zu können.

(7. URL ...) Mit Hilfe des Servlet-Kontext kann man mit der "getResource()"-Methode auf alle Ressourcen unter der Wurzel der Anwendungsstruktur zugreifen. Würde man zum Ressourcenladen die "getResource()"-Methode über den Classloader mit "this.getClass().getResource()" zugreifen könnte man nicht auf den "WEB-INF"-Ordner zugreifen. Mehr kann man unter [TOMCAT-CLASS-LOADER] finden. Die Motivation für die "getResource()"-Methode ist, dass der Pfad der zu den Ressourcen führt, nicht Hartkodiert sein muss, da er je nach Deployement anders sein kann.
(8. List<POI> poiList ...) Nachdem der Pfad zur Ressource, mit der "getResource()"-Methode ermittelt wurde, kann die Arbeit an den XMLPOIManager delegiert werden. Als Resultat wird eine POI-Liste zurückgeliefert.
(9. List<POI> resultPoiList ...) Die JSP-Seite übernimmt die Ausgabe. Deshalb braucht man ein Objekt, dass die Resultate hält und in das Session-Objekt gespeichert wird, um es dann in der JSP-Seite auszulesen. Das "resultPoiList"-Objekt wird mit den POIs, die zu der angeforderten Kategorie passen, bevölkert.
Das selbe Resultat kann erzielt werden wenn aus der, vom XMLPOIManager zurückgelieferten, Liste, die POIs die nicht die passende Kategorie haben, entfernt werden. Dann bräuchte man kein extra Objekt. Es ist aber übersichtlicher und effizienter wie es hier beschrieben wurde.

[TOMCAT-CLASS-LOADER] http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html
[KR10] James F. Kurose, Keith W. Ross: Computer Networking – A Top-down Approach, 5th Edition, Pearson, 2010.



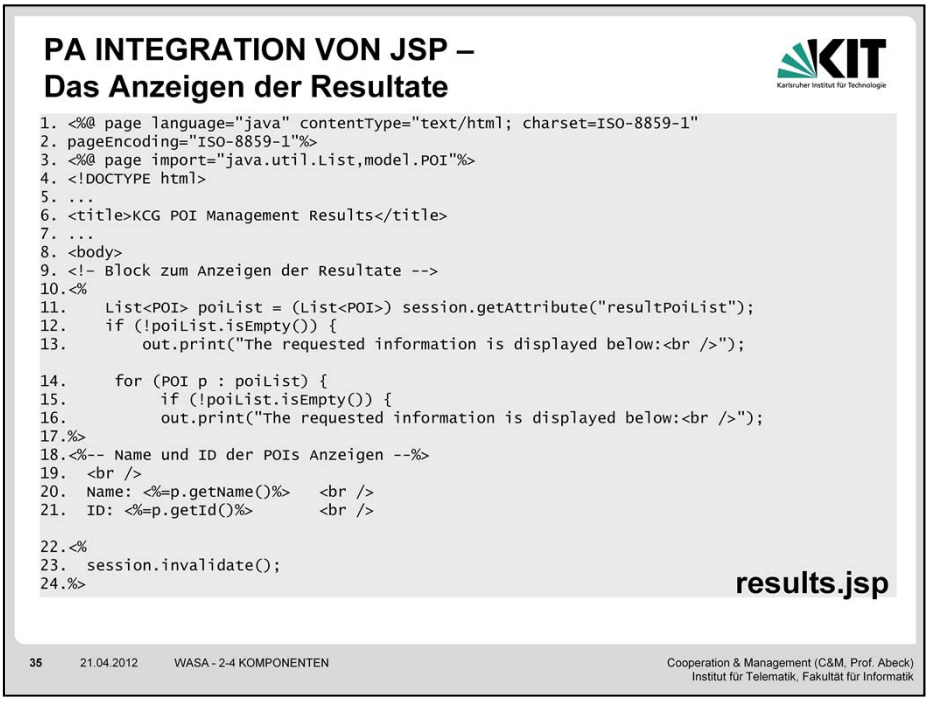

(10. if ...) Beispiel eines If-Statements zur Überprüfung des Objekts um sicherzustellen, dass das Program nicht Abstürzt, so wird eine "NullPointerException" vermeidet, wenn das Objekt auf keine Liste zeigt, i.e. nicht initialisiert ist.
(11. for ...) Mit Hilfe einer in Java vorhandenen"for-each"-Schleife wird durch die Liste iteriert.
(12. if ... equalsIgnoreCase ...) Vergleich zwischen der Kategorie des in der Schleife aktuellen POIs mit der vom Benutzer in der HTML-Form gesendeten Kategorie. Auf Groß- und Kleinschreibung wird nicht geachtet.
(13.result.poiList.add(p);) Passende POIs werden in die Resultatliste gespeichert.
(17. session.setAttribute ...) Die Session ist ein automatisch erstelltes Objekt, das als Ablage dient für Daten dient. Es wird ein Session-Objekt pro Benutzer erstellt. Anfragen vom selben Benutzer. Der Benutzer wird meistens durch den Cookie-Mechanismus identifiziert [Kr10:136]. Mit der "setAttribute"-Methode des "session"-Objekts wird die Resultatliste in die Session gespeichert und ist damit in der JSP-Seite aufrufbar. Der erste Parameter ist der Schlüssel, der Name, mit dem die Resultatliste in das Session-Objekt gespeichert wird. Der zweite Parameter ist das "resultPOIList"-Objekt an sich.

(16. - 19. if (errorMsg ...) Wenn ein kein Fehler aufgetreten ist, dann wird die POI-Liste mit den Resultaten in das Session-Objekt gespeichert und das "redircetString"-Objekt wird auf die "results.jsp"-Seite verweisen.

(20. - 23., else{) Wenn ein Fehler aufgetreten ist dann wird die Fehlernachricht in das Session-Objekt gespeichert und das "redircetString"-Objekt wird auf die "error.jsp"-Seite verweisen.

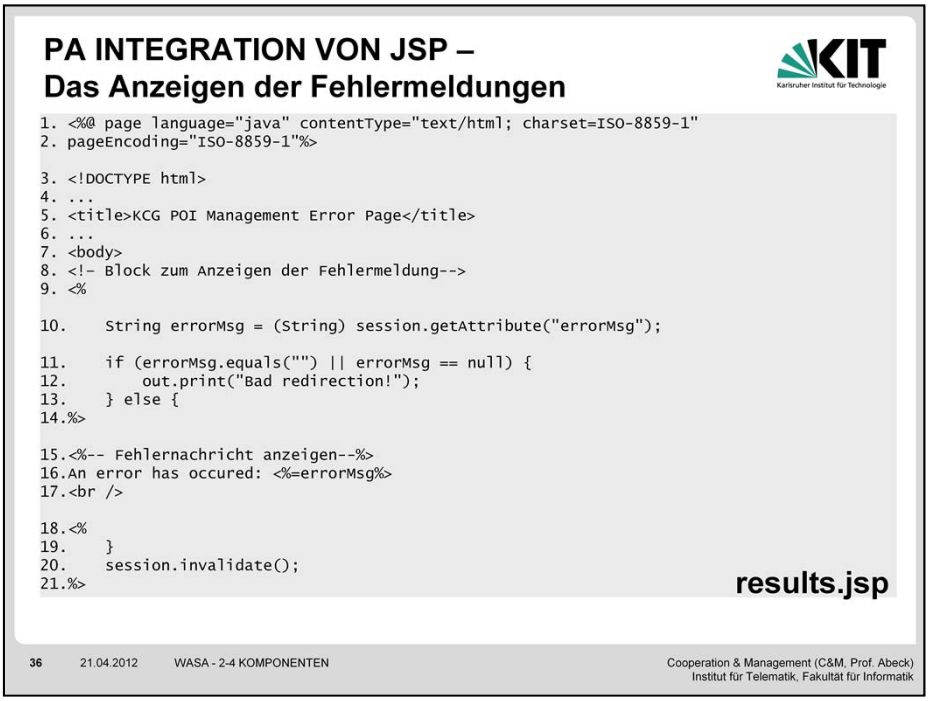
(24. response.sendRedirect ...) Mit der "sendRedirect"-Methode des "response"-Objekts wird der normale Informationsfluss durchbrochen. Anstatt dem Benutzer eine vom POIManager-Servlet generierte HTML-Seite anzuzeigen, wird erst eine der JSP-Seiten aufgerufen, die das Endresultat liefert. Die Umleitung (engl. redirect) wird erst nach dem Terminieren des Kontrollflusses des Servlets aktiv.

[TOMCAT-CLASS-LOADER] http://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html
[KR10] James F. Kurose, Keith W. Ross: Computer Networking – A Top-down Approach, 5th Edition, Pearson, 2010.




Der hier präsentierte Code ist vereinfacht um die Funktion zu veranschaulichen.
In der, zur Verfügung gestellten, Quellcodedatei zur PA sind Fehlerfindungsmechanismen eingebaut.

(1.) Die Standard-JSP-Direktive.
(3.) Die Import-Direktive, die die Liste aus java.util und den POI aus model importiert. Diese Direktive funktioniert wie die Java Import-Anweisung nur kann hier auch eine durch Komma separierte Liste von Imports geschrieben werden.
(4.-8.) HTML-Tags wie sie in einer normallen HTML-Seite auch anwesend wären.
(9.) Dies ist ein normales HTML-Kommentar und ist auch in einer JSP-Seite zugelassen.
(10.) Anfang eines JSP-Tags. Ab hier wird nur Java-Code erlaubt.
(11.) Die Liste von POIs wird hier aus der Session geholt. Der Name, hier "resultPOIList", mit dem die Liste in die Session gespeichert wurde muss bekannt sein. In der Session werden nur Objekte (java.lang.Object) gespeichert also muss durch einen expliziten Typ-Cast (downcast um genau zu sein) auf das korrekte Objekt gecastet werden. Bei falschen Casts werden Exceptions geworfen wie die "ClassCastException". In der, zur Verfügung gestellten, Quellcodedatei zur PA wird ein Fehlerfindungsmechanismus für diesen Fall gezeigt.
(12.) Eine Überprüfung der POI-Liste um zu bestimmen ob sie Elemente enthält.
(14.) Mit Hilfe einer von Java bereitgestellten "for-each"-Schleife wird durch die Liste iteriert.
(16.) Mit der "print()"-Methode des "out"-Objekts kann auf die HTML-Seite die von der JSP-Seite erstellt wird geschrieben werden, wie wenn es im statischen HTML enthalten wäre.
(17.) Ende des JSP-Tags aus Zeile 9. HTML darf wieder geschrieben werden.
(18.) Ein JSP-Kommentar das nicht in der gelieferten HTML-Seite enthalten sein wird.
(19.-21.) Die Daten des POI werden in HTML ausgegeben. Die kurzen Java-Sequenzen die in "<%=" und "%>" enthalten werden als String in die Seite geschrieben.
(23.) Das Session-Objekt wird invalidiert und die darin enthaltenen Attribute gelöscht, um eine erneute Anfrage vom selben Benutzer auswerten zu können.




(1.) Die Standard-JSP-Direktive.

(3.-7.) HTML-Tags wie sie in einer normallen HTML-Seite auch anwesend wären.
(8.) Dies ist ein normales HTML-Kommentar und ist auch in einer JSP-Seite zugelassen.
(9.) Anfang eines JSP-Tags. Ab hier wird nur Java-Code erlaubt.

(10.) Die Fehlernachricht "errorMsg" wird aus der Session geholt.
(12.) Wenn "errorMsg" nicht initialisiert oder leer ist dann ist wahrscheinlich eine Falsche Umleitung geschehen.
(13.) Wenn "errorMsg" eine Fehlernachricht enthält dann sollte sie auch dem Benutzer Angezeigt werden.
(14.) Ende des JSP-Tags aus Zeile 9. HTML darf wieder geschrieben werden.

(15.) Ein JSP-Kommentar das nicht in der gelieferten HTML-Seite enthalten sein wird.
(16.) Die Fehlernachricht wird angezeigt indem ein JSP-Snippet in dem HTML-Teil der JSP-Seite eingebettet wird.

(20.) Das Session-Objekt wird invalidiert und die darin enthaltenen Attribute gelöscht, um eine erneute Anfrage vom selben Benutzer auswerten zu können.


(1) Durch die angegebene Befehlsfolge lässt sich die Web-Anwendung aus Eclipse starten.

Die folgenden Bildschirmausschnitte zeigen beispielhaft den korrekten bzw. fehlerhaften Ablauf der Anwendung auf.