# Horus

Java Framework
für
Servlet basierte Internet/Intranet Anwendungen

Seminararbeit

Institut für Informationsverarbeitung und Computergestützte Neue Medien
Technische Universität Graz

CHRISTIAN MAYRHUBER <Christian.Mayrhuber@gmx.net>

SS 2000

# Inhaltsverzeichnis

# 1 Einleitung

Internet und Intranet sind Schlagworte, hinter deren technischer Realisierung grosse Dynamik in wie kaum einem anderen Bereich steckt. Diese Dynamik ergibt sich einzig und allein durch Kundenwünsche, bietet sich doch für die Unternehmen die Möglichkeit den Informationsfluß zwischen den Mitarbeitern zu erhöhen und gleichzeitig die *corporate identity* vom Mitarbeiter bis hin zu potentiellen Kunden zu transportieren. Daher existiert keine Lösung, die für jedes Unternehmen passt, es gibt nur speziell auf die Bedürfnisse einer Firma oder Abteilung zugeschnittene Lösungen, welche aber durchaus gemeinsame Grundbausteine besitzen.

Ein Framework kann die Entwicklung derartiger Softwarebausteine beschleunigen, indem Aufgaben, die bei jeder Anfrage eines Webbrowsers an die Software anfallen vom Framework übernommen werden und damit die Entwicklungszeit verkürzt wird.

Diese Arbeit beschreibt den Aufbau des Frameworks *Horus* und dessen Desingentscheidungen. Es gliedert sich in folgende Bereiche:

- *Kapitel 2* gibt einen Überblick über die im World Wide Web verwendeten Technologien und beschreibt warum Java und XML die Basis für Horus sind.
- *Kapitel 3* stellt verwendete Softwareentwurfsmuster vor.
- *Kapitel 4* gibt einen Designüberblick von Horus in der aktuellen Version.

# 2 Webtechnologie

Dieses Kapitel widmet sich dem Überblick über die im World Wide Web verwendeten und für das Framework relevanten Technologien und warum Java und XML die Basis von Horus bilden.

## 2.1 Hypertext Markup Language

Die *Hypertext Markup Language (HTML)* wurde 1989 von Tim Berners-Lee am CERN in Genf entwickelt, um den dort arbeitenden Wissenschaftlern den Zugang zu Informationen zu erleichtern[36].

HTML ist eine sogenannte Auszeichnungssprache (Markup Language). Sie hat die Aufgabe, die logischen Bestandteile eines Dokuments zu beschreiben. Als Auszeichnungssprache enthält HTML daher Befehle zum Markieren typischer Elemente eines Dokuments, wie Überschriften, Textabsätze, Listen, Tabellen oder Grafikreferenzen.

Das Beschreibungsschema von HTML geht von einer hierarchischen Gliederung aus: HTML beschreibt Dokumente. Dokumente haben globale Eigenschaften, wie zum Beispiel einen Titel oder eine Hintergrundfarbe. Der eigentliche Inhalt besteht aus Elementen, zum Beispiel einer Überschrift 1. Ordnung. Einige dieser Elemente haben wiederum Unterelemente[21].

## 2.2 Hypertext Transfer Protocol

Das *Hypertext Transfer Protocol (HTTP)* ist genau wie HTML 1989 von Tim Berners-Lee und seinen Mitstreitern am CERN entwickelt worden. Es dient zur Übertragung von Daten und es besteht keine dauernde Verbindung zwischen dem WWW Browser (Client) und dem WWW Server (Server). Die Kommunikation zwischen Client und Server erfolgt vielmehr über ein simples Anfrage/Antwort-Schema.

Über das HTTP Protokoll können durch die Angabe eines *Uniform Resource Locators (URL)* die Ressourcen eines Servers angesprochen werden. Ressourcen können statische Dateien, Servlets oder aktive Inhalte wie CGI Skripte sein [17].

## 2.3 Common Gateway Interface

Das *Common Gateway Interface (CGI)* wurde von der NCSA (National Center for Supercomputing Applications) entwickelt und erlaubt es einem WWW-Browser über einen WWW-Server Programme auszuführen. Solche Programme (oder Skripte) können beispielsweise Formulareingaben aus HTML-Dateien verarbeiten, auf dem Server-Rechner Daten speichern und dort gespeicherte Daten auslesen. Auf diese Weise werden WWW-Seiten zu Oberflächen für Änwendungen", beispielsweise für elektronische Warenbestellung oder zum Abfragen von Datenbanken[21].

## 2.4 Java Servlets

Die Entwicklung von *Java* begann in den 80ern bei Sun Microsystems unter dem Namen Oak. Das Ziel war eine in erster Linie plattformunabhängige Programmiersprache. So kommt es, daß Java Programme in einen Bytecode kompiliert werden, der in Java Virtual Machines (JVM) lauffähig ist. Diese JVM wird auf den meisten Plattformen als Programm zur Verfügung gestellt.

Das *Java Servlets* Application Programming Interface (API) wurde von Sun als Schnittstelle definiert um serverseitigen Java Anwendungen eine einfache und flexible Grundlage zu geben. Der Vorteil von Servlets gegenüber CGI besteht darin, daß die Servlets eine objektorientierte Schnittstelle zum Webserver definieren und da vorkompiliert, bereits als Bytecode vorliegen in der Regel schneller arbeiten als die meisten zu interpretiertend CGI Skripte. In den Servlets steht ausserdem das gesamte Application Programming Interface (API)zur Verfügung, womit sich einfach Datenbanken, Namensdienste und andere Server ansprechen lassen. Das Java API ist bereits sehr umfangreich und wird laufend weiterentwickelt[4] [7] [23].

## 2.5 Extensible Markup Language

Die *Extensible Markup Language (XML)* ist eine Teilmenge von SGML, die vollständig in diesem Dokument beschrieben wird. Das Ziel ist es, die Möglichkeit zu bieten, generic SGML in der Weise über das Web auszuliefern, zu empfangen und zu verarbeiten, wie es jetzt mit HTML möglich ist. XML wurde entworfen, um eine einfache Implementierung und Zusammenarbeit sowohl mit SGML als auch mit HTML zu gewährleisten[20].

XML enthält im Gegensatz zu HTML, keinerlei Befehle um ein Dokument zu formatieren, zum Beispiel ist es nicht möglich festzulegen, das ein Text fett gedruckt erscheint. XML beschreibt vielmehr die logische Struktur eines Dokuments wie Name, Adresse, Land, usw. Diese logische Struktur kann aber mittels eines Style Sheets auf ein konkretes Aussehen, wie etwa in ein HTML oder PDF Dokument transformiert werden [27] [24].

XML trennt also die Struktur vom Aussehen, was ein enormer Vorteil gegenüber HTML ist. Mit XML steht also erstmals ein internationaler Standard für den Datenaustausch zur Verfügung.

## 2.6 Schlußfolgerung

Kombiniert man die Plattformunabhängigkeit von Java mit der Möglichkeit, auf Daten in dem standardisierten und flexiblen XML Format zugreifen zu können, so ist diese Kombination geradezu dafür prädestiniert, als Middleware zu fungieren.

Da die meisten Unternehmen im Internet präsent sein wollen, um dort Informationen über das Unternehmen anzubieten, ist der Zugriff auf unternehmensinterne Daten unverläßlich. Die Kombination Java und XML erleichtert dies beträchtlich, da nicht für jede proprietäre Anwendung eine eigene Middleware geschrieben werden muß, der Datenaustausch erfolgt über XML und die plattformunabhängige Anwendungslogik in Java bleibt dieselbe.

# 3 Design Patterns

*Design Patterns* sind Entwurfsmuster wiederverwendbarer Software. Dieses Kapitel beschreibt die Grundzüge der Entwurfsmuster welche im Horus Framework eingesetzt werden. Design Patterns wurden 1994 erstmals im Buch *Design Patterns, Elements of Reuseable Software* beschrieben und damit eine gemeinsame Sprachregelung eingeführt[29]. Dieses Kapitel setzt eine gewisse Vertrautheit mit der Objektorientierten Programmierung voraus. Eine gute Einführung in die Objektorientierte Programmierung mit Java findet sich auf der Sun Homepage, im Java Tutorial [4].

Für die Design Patterns werden im folgenden die englischen Orginalbezeichnungen verwendet, damit die einheitliche Bezeichnung der Patterns beibehalten wird.

Die Anwendbarkeit eines Patterns beschreibt wofür ein Pattern eingesetzt werden kann.

## 3.1 Creational Patterns

*Creational Patterns* abstrahieren den Instantierungsprozess, sie machen ein System unabhängig davon wie Objekte erzeugt werden.

### 3.1.1 Abstract Factory

Stellt eine Schnittstelle zur Verfügung, sodaß Familien von verwandten oder abhängigen Objekten erzeugt werden können, ohne ihre konkreten Klassen anzugeben. Es müssen also zum Kompilationszeitpunkt nicht alle Klassen bekannt sein die im Programm verwendet werden. Vielmehr eröffnen Factories die Möglichkeit unbekannte Programmteile dynamisch nachzuladen, sofern die Schnittstellen eingehalten werden.

**Anwendbarkeit**

- Ein System soll von der Art und Weise wie die Produkte erzeugt, zusammengefügt und repräsentiert werden, unabhängig sein.
- Ein System soll mit einer von mehreren Produktfamilien konfiguriert werden.
- Eine Familie von verwandten Produkt Objekten ist entworfen um miteinander zu funktionieren und sie müssen sicherstellen, daß sie auch so verwendet werden.
- Sie möchten eine Klassenbibliothek mit Produkten zur Verfügung stellen und nur deren Schnittstellen, nicht aber deren Implementierung offen legen.

## 3.2 Structural Patterns

*Structural Patterns* sind Entwurfsmuster, die beschreiben wie Objekte zusammengefügt werden um grössere Strukturen zu formen.

### 3.2.1 Adapter

Ein Adapter konvertiert die Schnittstelle einer Klasse in eine Schnittstelle die von einer Software erwartet wird. Adapter lassen Klassen, die sonst wegen unterschiedlicher Schnittstellen nicht zusammenarbeiten könnten, miteinander funktionieren.

**Anwendbarkeit**

- Sie möchten eine existierende Klasse benutzen, aber ihre Schnittstelle ist nicht die, die sie benötigen.
- Sie wollen eine wiederverwendbare Klasse entwerfen, welche mit Klassen mit inkompatiblen Schnittstellen zusammenarbeiten muss und dies nicht vorhergesehen werden konnte.
- Sie müssen einige abgeleitete Klassen benutzen, aber es ist unpraktisch ihre Schnittstellen zu verwenden, in dem jede von ihnen abgeleitet wird. Ein Objekt Adapter kann die Schnittstelle der Basisklasse adaptieren.

### 3.2.2 Bridge

Eine Bridge entkoppelt eine Schnittstelle von ihrer Implementierung. So kann die Implementierung erweitert werden, ohne dass sich dies auf die Schnittstelle auswirkt, oder umgekehrt.

**Anwendbarkeit**

- Sie wollen eine dauernde Bindung zwischen einer Abstraktion und ihrer Implementierung verhindern. Dies ist beispielsweise der Fall, wenn die Implementierung während der Laufzeit ausgewählt werden soll.
- Die Abstraktion und ihre Implementierung müssen beide durch ableiten erweiterbar sein. Das Bridge Pattern läßt sie die unterschiedlichen Abstraktionen und unabhängig von einander ableiten.
- Änderungen in der Implementierung verursachen keine Neukompilation ihrer Software.
- Sie wollen die Implementierung einer Abstraktion komplett von der benutzenden Software verbergen.
- Sie wollen eine Implementierung mit vielen Objekten teilen (etwa über Referenzzählung) und diese Tatsache soll verborgen werden.

## 3.3 Behavioral Patterns

*Behavioral Patterns* beschäftigen sich mit Algorithmen und der Zuweisung von Verantwortlichkeiten von Objekten. Behavioural Patterns beschreiben nicht nur Muster von Klassen oder Objekten, sondern auch die Kommunikation zwischen ihnen.

### 3.3.1 Observer

Definiert eine ein-zu-mehreren Abhängigkeit zwischen Objekten, sodaß falls ein Objekt seinen Zustand ändert alle anderen darüber benachrichtigt werden. Dies hat den Vorteil, dass die Objekte, die benachrichtigt werden wollen, sich erst zur Laufzeit beim Observer anmelden müssen.

#### Anwendbarkeit

- Wenn eine Abstraktionn zwei Gesichtspunkte hat, einer abhängig vom anderen. Die Kapselung dieser Gesichtspunkte in zwei separate Objekte lässt sie beide verändern und unabhängig voneinander verwenden.
- Wenn eine Änderung eines Objekts eine Änderung anderer Objekte bewirkt und sie nicht wissen wie viele Objekte geändert werden müssen.
- Wenn ein Objekt andere Objekte benachrichtigen soll und nichts über diese Objekte wissen muss.

### 3.3.2 Strategy

Definiert eine Schnittstelle für eine Familie von Algorithmen, kapselt jeden Algorithmus und ermöglicht deren Austausch. *Strategy* macht den Algorithmus unabhängig von der Software die ihn benutzt austauschbar.

#### Anwendbarkeit

- Viele Klassen unterscheiden sich nur in ihrem Verhalten. Strategy ermöglicht das Verhalten einer Klasse einzustellen.

- Sie benötigen unterschiedliche Variationen eines Algorithmus. Wenn diese Variationen als Klassenhierarchie implementiert sind, kann Strategy benutzt werden.
- Ein Algorithmus benutzt Daten über die nur der Algorithmus verfügen muss.
- Eine Klasse hat mehrere Verhalten und diese treten als mehrfache Bedingungsabfragen (if Abfragen) in ihren Operationen auf. Anstatt die vielen Bedingungsabfragen durchzuführen können ähnliche Bedingungszweige in eine eigene Strategy Klasse ausgelagert werden.

# 4 Designüberblick

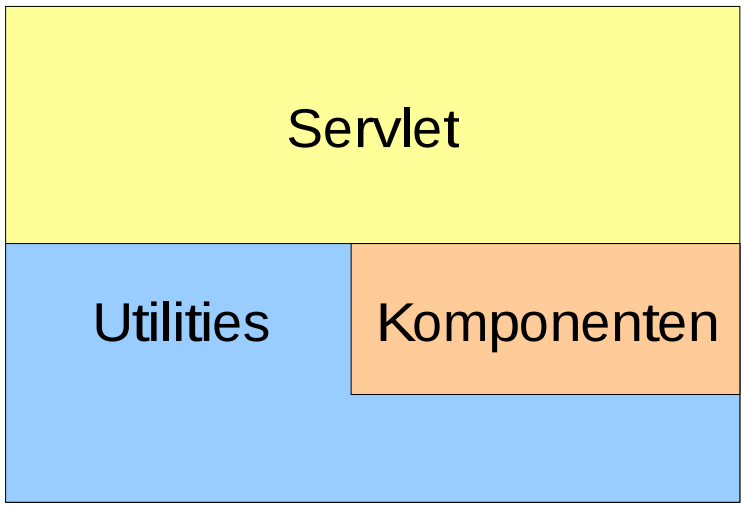


Abbildung 1: Übersicht über das Framework

Dieses Kapitel gibt einen Überblick über das Design des Frameworks und versucht zu erklären, warum ein konkretes *Design Pattern* in einem bestimmten Teil des Frameworks eingesetzt wurde. Horus gliedert sich in drei Teile, dem Servlet, den Komponenten und den Utilities, siehe Abbildung 1.

Das Servlet stellt den Gateway vom Webbrowser zu den Komponenten dar. Die Komponenten beantworten die Requests des Webbrowsers und benutzen die Utilities. Da die Utilities von den Komponenten und dem Servlet benutzt werden und das Servlet die Komponenten lädt, erfolgt die Darstellung von den Utilities zuerst und anschließend die des Servlets mit den Komponenten. Für das detaillierte Design möchte ich auf den Projektbericht [34] verweisen.

## 4.1 Utilities

### 4.1.1 Factory

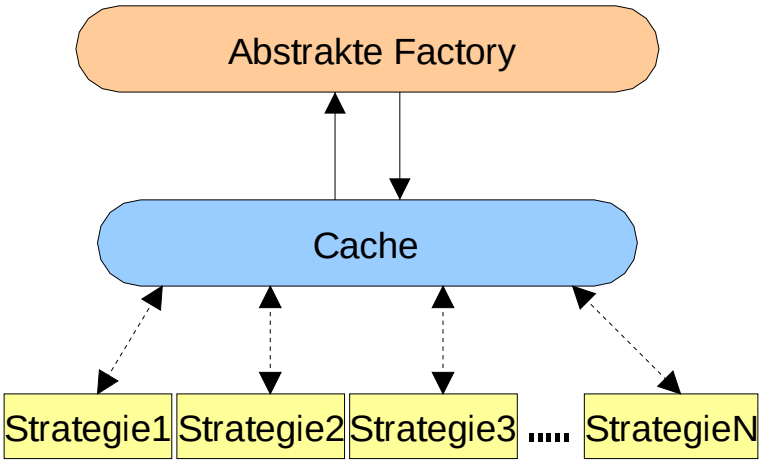


Abbildung 2: Kommunikationsfluß der Factory

Die Factory entspricht dem Design Pattern *Abstract Factory* [29] und ist in der Lage Objekte, welche durch eine Namenskonvention und einen Suchpfad definiert werden, zu laden und zu instantieren. Damit die Factory verwendet werden kann, muß die Namenskonvention in einer *Concrete Factory* implementiert werden.

Der Cache ist in der Lage beliebiege Java Objekte zu puffern welche über einen eindeutigen Schlüssel angesprochen werden können. Im Fall

der Factory ist der Schlüssel eine Zeichenkette die den voll qualifizierten Klassennamen also z.B.:`at.osiris.horus.util.Cache` enthält. Der Cache verfügt über eine Strategie, welche ihm bei seiner Instantierung mitgeteilt werden muss. Welche Objekte wie lange im Speicher gehalten werden wird über diese Strategie des Caches festgelegt, siehe Abbildung 2. Der Einfachheit halber wird in der ersten Version von Horus nur eine First In First Out (FiFO) Strategie implementiert. Das Design Pattern *Strategy* [29] beschreibt ein derartiges Muster allgemein. Im diesem speziellen Fall verfügt die Strategie über eine eigene Datenstruktur, über welche sicherstellen soll, daß effizient festgestellt wann ein Objekt aus dem Cache gelöscht wird um einem anderen Platz zu machen.

### 4.1.2 XML Utilities

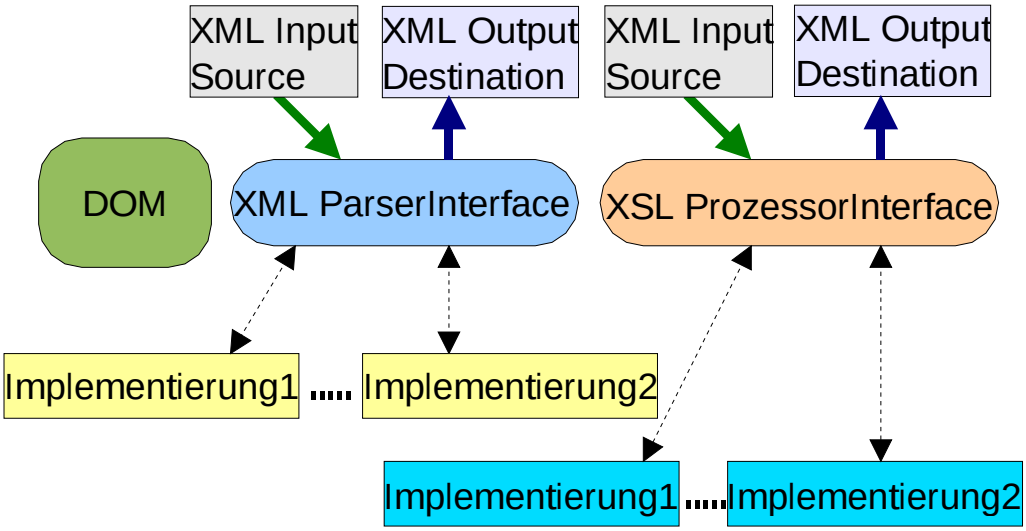


Abbildung 3: Aufbau der XML Utilities

Die XML Utilities definieren Schnittstellen welche eine spezielle Implementierung eines Parsers und eines XSL Prozessors von der im Framework verwendeten trennen. XML Parser und Prozessor erhalten ihre Datenquelle von einem `XMLInputSource` Objekt und schreiben ihre Ausgabe an das Ziel, einem `XMLOutputDestination` Objekt. Die beiden Ein/Ausgabe Klassen sind in der Lage DOM [26] Bäume, Streams und Dateien zu verarbeiten. Das Framework arbeitet intern nur mit

den Schnittstellen `XMLParserInterface` und `XMLProcessorInterface` wobei die tatsächliche Implementierung offengelassen wird. Dies entspricht dem Design Pattern *bridge* [29]. Der Vollständigkeit halber sei erwähnt, daß Xerces [14] und Xalan [16] der Apache Organisation [1] verwendet werden.

In den XML Utilities existiert eine Klasse DOM mit nur statischen Methoden welche es erlaubt es einen DOM Baum aus einer XML Datei zu lesen, nach Knoten in diesem zu suchen und Attribute eines Knotens zu lesen, sowie einen DOM Baum in eine XML Datei zu schreiben. Grund hierfür ist, daß die Konfigurationen einen Document Object Model (DOM) [26] Baum enthalten, welcher einfach nach den Konfigurationsdaten durchsucht werden soll. Siehe Abbildung 3.

### 4.1.3 Konfigurationsmanager

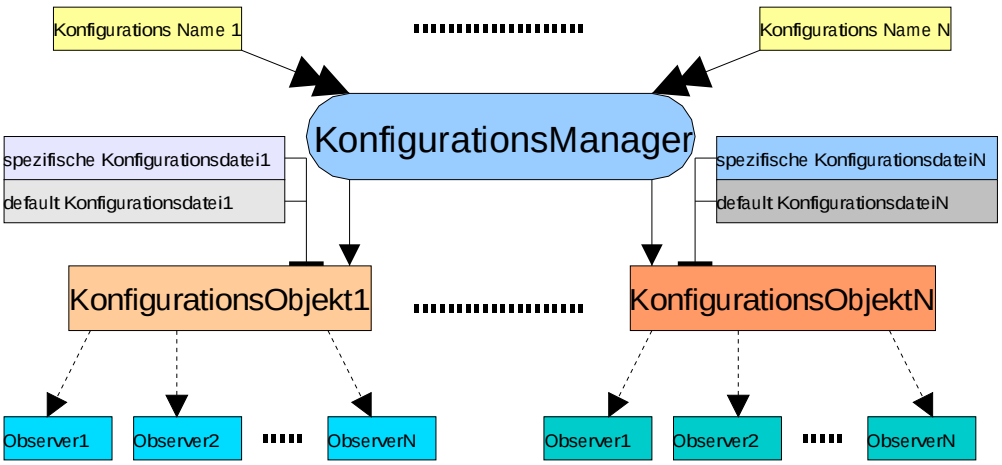


Abbildung 4: Laden von Konfigurationen

Der Konfigurationsmanager stellt Konfigurationen über einen Namen bereit. Für den Konfigurationsmanager ist nur der Name der Konfiguration bekannt und die Postition der Konfigurationsdatei mit den Standardwerten und der Konfigurationsdatei mit den spezifischen Konfigurationsdaten. Mit dieser Information wird ein Konfigurations Objekt instantiert,

das dann diese beiden Dateien lädt und auf eine Änderung überwacht. Sollte eine Konfigurationsdatei verändert werden und es ist ein *Observer* [29] registriert so wird diese Datei erneut geparst und dem Observer als Ergebnis mitgegeben, siehe Abbildung 4 Nicht nur die Teile des Frameworks lassen sich mit dem Konfigurationsmanager verwalten, sondern auch die Daten die zum Aufbau einer Anwendung notwendig sind, wie wenn beispielsweise für ein Interface eine passende Klasse benötigt wird, die über eine Factory geladen werden soll.

### 4.1.4 Logger

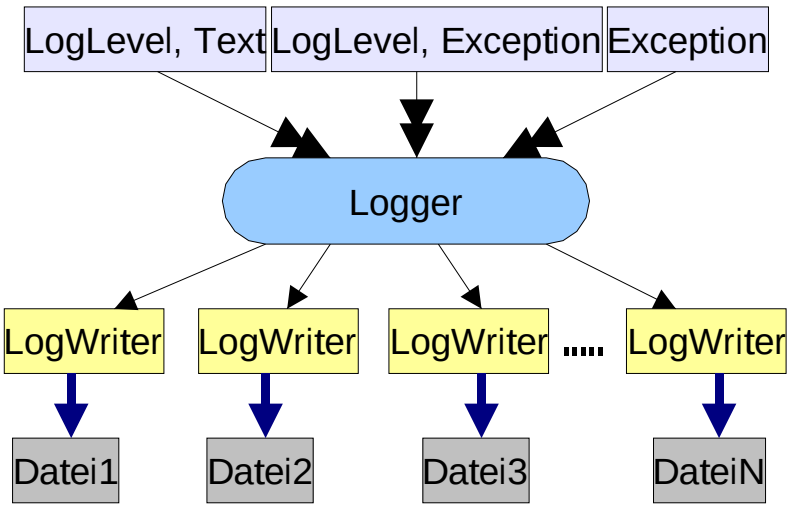


Abbildung 5: Aufbau des Loggers

Der Logger ist dafür zuständig, dass Nachrichten mit einer bestimmten Logging Stufe in jene Datei geschrieben wird, welche dafür vorgesehen ist. Außerdem ist der benötigt der Logger eine Konfiguration in der in XML Notation festgelegt ist wohin welcher Loglevel geschrieben wird, sowie das Format der Nachricht. Es werden nur die LogLevels geschrieben die in der Konfigurationsdatei einem Ausgabeziel zugeordnet sind, alle anderen werden ignoriert. Der Logger schreibt die Loginformation nicht selbst in jede LogDatei sondern bedient sich einem `LogWriter` pro LogDatei, siehe Abbildung 5. Werden Exceptions geloggt so wird automa-

tisch der StackTrace geschrieben. Ist dennoch die Nachricht der Exception erwünscht so muß die ihre `toString` Methode aufgerufen und diese Nachricht geloggt werden.

## 4.2 Servlet und Komponenten

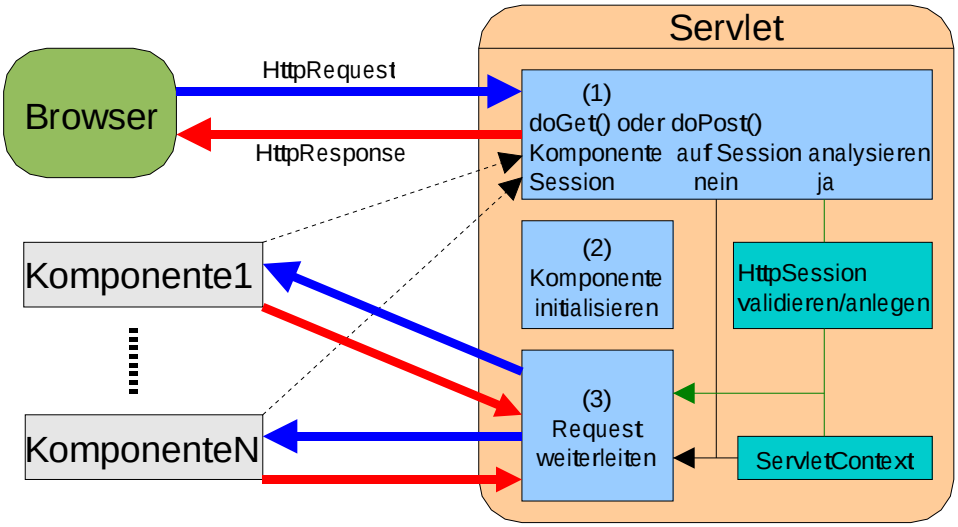


Abbildung 6: Servlet und Komponenten

Das Servlet nimmt Requests eines Browsers, wie in Abbildung 6 dargestellt, entgegen und leitet ihn an die Komponente weiter die ihn bearbeitet und die Antwort zum Browser zurückschickt.

**1** Bei einem Request des Web Browsers [17] wird die Methode doGet() oder doPost() des Servlets aufgerufen und in die beiden Objekte `HttpRequest` und `HttpResponse` gekapselt. Aus dem HttpRequest analysiert das Servlet den Parameter *command*, sieht mit dessen Wert in der Konfiguration nach und lädt über die ComponentFactory die Komponente.

Falls die Komponente eine Session benötigt, muß sie das `SessionComponentInterface` implementieren. Dies wird vom Servlet abgefragt. Falls keine gültige HttpSession existiert, dann wird ein neues Session Objekt über Cookies angelegt.

**2** In einer URL [12] lassen sich Parameter angeben, welche als Name/Wert Paare definiert sind [21], z.B.:`command=xsl&file=test.xml`. Der Parameter command wird vom Servlet benötigt und definiert die Komponente, der Parameter file betrifft eine Methode der Komponente xsl, er definiert, daß von der Komponente xsl die Methode `setFile(String filename)` aufgerufen wird. Dies funktioniert mit beliebigen `set` Methoden.

**3** Nach dem Laden und der Initialisierung der Komponente wird ihre `serve` Methode aufgerufen. Dieser Methode wird je nach dem ob die Komponente eine Session benötigt, oder nicht, ein HttpSessionEvent oder ein HttpEvent übergeben. Der HttpEvent kapselt einen HttpRequest, die HttpResponse und den HttpContext. Der HttpSessionEvent kapselt zusätzlich eine HttpSession.

Die `serve` Methode bearbeitet die Anfrage und liefert die Antwort über das HttpResponse Objekt an den Browser zurück.

### 4.2.1 Komponenten

Für die Komponenten sind zwei Schnittstellen definiert, das `ComponentInterface` und das `SessionComponentInterface`. Beide Schnittstellen definieren eine `init` Methode und eine `serve` Methode. `init` besitzt einen Parameter vom Typ Object über den etwaige Initialisierungsargumente übergeben werden können. Die Komponente muß auch ordnungsgemäß funktionieren ohne dass `init` aufgerufen wurde.

Beim ComponentInterface besitzt die `serve` Methode einen Parameter vom Typ HttpEvent, beim SessionComponentInterface einen HttpSessionEvent.

Damit auch Komponenten auf den Logger und den Konfigurationsmanager zugreifen können, werden beide im ServletContext als Attribute mit den Namen Log und ConfigManager abgelegt.

# A Begriffsdefinitionen

**HTML** Ist die HyperText Markup Language, die Standard Auszeichnungssprache des WWW.

**HTTP** Bedeutet HyperText Transfer Protocol und ist das Anwendungsprotokoll das die Webbrowser mit den HTTP Servern sprechen.

**Web Publishing** Stellt Daten über HTTP Server in Form von HTML im Internet oder Intranet Personen zur Verfügung.

**XML** Ist die Extensible Markup Language, eine vom WWW-Konsortium standardisierte Auszeichnungssprache die es erlaubt mittels DTD's eigene Auszeichnungssprachen wie HTML zu definieren.

**DTD** Document Type Definition, beschreibt den Aufbau eines XML Dokuments und wird zur Überprüfung auf Konformität eingesetzt.

**XSL** Extensible Stylsheet Language dient dazu Stilvorlagen für XML Dokumente zu erzeugen.

**XSLT** XSL Transformation, ist das Vokabular von XSL zur Transformation eines XML Dokuments in einen anderen Dokumenttyp.

**Konfigurationsmanager** Verwaltet Konfigurationsdaten für Softwarekomponenten und ganze Anwendungen im XML Format.

**Logger** Schreibt Fehler- und Statusmeldungen in Logdateien.

**Servlet** Bezeichnung eines mini Servers in Java der das Servlet Interface implementiert. Servlets nehmen unterschiedliche Netzwerkanfragen an, bearbeiten diese und liefern die Antwort zurück.

**RDBMS** Relationales DatenBank Management System, ist eine Bezeichnung für Relationale Datenbanken mit den dazugehörigen Verwaltungstools.

**Middleware** Mit Middleware beizeichnet man Software die Anwendungen welche ansonsten nicht miteinander Daten austauschen können den Datenaustausch ermöglichen.

**JDK** Java Development Kit, die Entwicklungsumgebung für Java

**JSDK** Java Servlet Development Kit, die Entwicklungsumgebung für Servlets

**UR** User Requirement, eine Anforderung die der Kunde an das Produkt stellt.

**TR** Technical Requirement, eine Anforderung die das Produkt an die Technik stellt.

**SR** Software Requirement, eine Anforderung die an die betreffende Software gestellt wird, hat ein UR mit der selben Nummer.

**SD** Software Design, das Software Design zum entsprechenden SR.

**Anwendungs Servlet** Ein Anwendungs Servlet bietet Zugang zu Softwarekomponenten auf einem Rechner im Netzwerk.

**Design** Ein Design beschreibt das Funktionieren und das Zusammenspiel der einzelnen Teile einer Software.

**Design Patterns** Sind Software Entwurfsmuster, welche im Buch Design Patterns erstmals niedergeschrieben wurden[29].

# Literatur

[1] Homepage apache webserver.
URL: http://www.apache.org.

[2] Homepage enterprise java beans.
URL: http://java.sun.com/products/ejb/.

[3] Homepage jakarta project.
URL: http://jakarta.apache.org.

[4] Homepage java.
URL: http://java.sun.com.

[5] Homepage java beans technologie.
URL: http://java.sun.com/beans/.

[6] Homepage java server pages.
URL: http://java.sun.com/products/jsp.

[7] Homepage java servlet api.
URL: http://java.sun.com/products/servlet/.

[8] Homepage java websever.
URL: http://www.sun.com/software/jwebserver.

[9] Homepage microsoft internet information services (iis).
URL: http://www.microsoft.com/windows2000/guide/server/features/web.asp?RLD=71.

[10] Homepage simple api for java (sax).
URL: http://www.megginson.com/SAX/Java/.

[11] Homepage sun, java and xml.
URL: http://java.sun.com/xml/.

[12] Homepage world wide web consortium.
URL: http://www.w3c.org.

[13] Homepage xml on apache.org.
URL: http://xml.apache.org.

[14] Homepage xml parser xerces.
URL: http://xml.apache.org/xerces.

[15] Homepage xml tutorial.
URL: http://java.sun.com/xml/tutorial_intro.html.

[16] Homepage xsl processor xalan.
URL: http://xml.apache.org/xalan.

[17] Internet engeneering taskforce, rfc http protocol.
URL: ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[18] Xforms - the next generation of web forms.
URL: http://www.w3.org/MarkUp/Forms/.

[19] Xml zone, ibm developerworks.
URL: http://www.ibm.com/developer/xml/.

[20] Extensible markup lanugage (xml) 1.0, w3c recommendation, February 1998.
URL: http://www.w3.org/TR/REC-xml.

[21] Selfhtml, html-dateien selbst erstellen, August 1998.
URL: http://www.teamone.de/selfhtml.

[22] Html 4.01 specification, w3c recommendation, December 1999.
URL: http://www.w3.org/TR/html4.

[23] *Java Servlet Specification*, 2.2 final edition, December 1999.

[24] Xsl transformations (xslt) version 1.0, w3c recommendation, November 1999.
URL: http://www.w3.org/TR/xslt/.

[25] Commerce xml (cxml) spezifikation version 1.1, April 2000.
URL: http://www.cxml.org/home/.

[26] Document object model (dom) level 2 specification, May 2000. http://www.w3.org/TR/DOM-Level-2/.

[27] Extensible stylesheet language (xsl) version 1.0, w3c working draft, March 2000.
URL: http://www.w3.org/TR/xsl/.

[28] Danny Ayers, Hans Bergstein, Michael Bogovich, Jason Diamond, Matthew Ferris, March Fleury, Ari Halberstadt, Paul Hole, Piroz Mohseni, Andrew Patzer, Ron Phillips, Sing Li, Krishna Vedati, Mark Wilcox, and Stefan Zeiger. *Professional Java Server Programming*.
Wrox Press Ltd.
Arden House
1102 Warwick Road
Acocks Green
Birmingham B27 6BH. UK, 1999.

[29] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reuseable Object-Oriented Software*. Addison Wesley (Deutschland) Gmbh, 1999.

[30] Cay S. Hostman and Gary Cornell. *Core Java 2, Volume 1*. Core Java 2. Sun Microsystems Press, 901 San Antoni Road, Palo Alto, California, 94303 USA, 1999.

[31] Cay S. Hostman and Gary Cornell. *Core Java 2, Volume 2*. Core Java 2. Sun Microsystems Press, 901 San Antoni Road, Palo Alto, California, 94303 USA, 2000.

[32] Jason Hunter and William Crawford. *Java Servlet Programming*. O'Reilly & Associates Inc., 1998.

[33] Thomas Kriechbaum. Xml und Java. Eine Seminararbeit für das Institut für Informationsverarbeitung undComputergestützte Neue Medien, June 2000.

[34] Christian Mayrhuber. Horus - projektbericht. Eine Projektarbeit für das Institut für Informationsverarbeitung undComputergestützte Neue Medien, June 2000.

[35] Michael Morrison. *Java Beans - Technik, Konzepte, Beispiele*. Markt & Technik (Deutschland) GmbH, 1997.

[36] Peter Rossbach and Hendrik Schreiber. *Java Server und Servlets*. Addison Wesley (Deutschland) Gmbh, 1999.

# Abbildungsverzeichnis