# Elektrotechnik und Informatik

DIPLOMARBEIT

## Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen


vorgelegt von:

Hans Jochen Wüst
Matr.-Nr.: 609058

betreut von:

Prof. Dr.-Ing. habil. Hans Wojtkowiak
Dr.-Ing. Bernd Klose
(Universität Siegen)

Dipl.-Math. Andreas Schreiber
Dipl.-Inform. Markus Litz
(Deutsches Zentrum für Luft- und Raumfahrt - Köln)



Tag der Ausgabe: 14.12.2007 Tag der Abgabe: 13.06.2008

# Abstract

This document presents the results of the diploma thesis „Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen“. In cooperation with the German Aerospace Center an open source tool will be realized to graphically test and evaluate the communication between WebDAV applications. First of all the necessary basics are discussed. Following this the developed solution is introduced and the concepts of the various subsystems are explained. Based on these results the design and implementation of the subsystems are described. A case of application shows the practical use of the developed concepts. Finally problems having occurred in the course of the development will be examined, and an outlook for possible developments and optimizing of the prototype will be given.

Im Rahmen dieser Ausarbeitung werden die Ergebnisse der Diplomarbeit „Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen“ präsentiert. In Kooperation mit dem Deutschen Zentrum für Luft- und Raumfahrt wird ein Open-Source-Werkzeug realisiert, mit dessen Hilfe die Kommunikation zwischen WebDAV-Anwendungen grafikunterstützt getestet und ausgewertet werden kann. Zunächst werden die benötigten Grundlagen erörtert. Im Anschluss daran wird das entwickelte Lösungskonzept vorgestellt und die Entwürfe der verschiedenen Subsysteme werden dargelegt. Das darauf aufbauende Design und die Umsetzung der Entwürfe werden beschrieben. Die Praxistauglichkeit der erarbeiteten Konzepte wird anhand eines Anwendungsfalls belegt. Abschließend werden im Laufe der Entwicklung aufgetretene Probleme betrachtet und ein Ausblick auf mögliche Weiterentwicklungen und Optimierungen des Prototypen skizziert.

# Inhaltsverzeichnis

# 1 Einleitung

Die rechnerbasierte Verarbeitung und Bereitstellung von Daten verlangt aufgrund von stetig wachsenden Anforderungen und immer größer werdenden Datenmengen nach neuen Lösungskonzepten. Komplexe Projektstrukturen, stetig größer werdende Datenmengen und auch räumlich verteilte Arbeitsgruppen erfordern effektive Werkzeuge zur Unterstützung und Optimierung der Datenverwaltung [AGG03].

Das HTTP-Protokoll ist ein weit verbreitetes und stabiles Protokoll, das es erlaubt, Informationen vielen Nutzern ohne großen technischen Aufwand zur Verfügung zu stellen. Allerdings sind die Benutzer damit nicht direkt in der Lage auch Daten zu bearbeiten und zu verwalten. An dieser Stelle setzt das WebDAV-Protokoll als Erweiterung des HTTP-Protokolls an und stellt die nötigen Funktionalitäten zur Verfügung.

Da sich das WebDAV-Protokoll derzeit noch in der „Etablierungsphase“ befindet, ist die Zahl der Anwendungen mit Unterstützung des WebDAV-Protokolls begrenzt, und auch entsprechende Entwicklungswerkzeuge sind noch nicht für alle Anwendungsfälle vorhanden. So existieren zwar Produkte für den Test von WebDAV-Server-Anwendungen, und auch allgemeine Netzwerkanalyse-Programme sind verfügbar, jedoch kein Werkzeug, das die Möglichkeit der Fehlersuche sowohl für Client- als auch Server-Anwendungen in Hinblick auf das WebDAV-Protokoll bietet. Ziel der vorliegenden Arbeit ist es, die Basis für solch ein Werkzeug zu schaffen. Dieses soll dem Entwickler von WebDAV-Applikationen die Fehlersuche und den Test von Server- und Client-Anwendungen erleichtern.

Die vorliegende Arbeit entstand im Rahmen einer Kooperation zwischen dem Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) und der Universität Siegen. Betreut wurde die Arbeit auf universitärer Seite durch

die Fachgruppe Technische Informatik, geleitet von Prof. Dr.-Ing. habil. Hans Wojtkowiak. Ansprechpartner war Dr.-Ing. Bernd Klose. Ansprechpartner in der Einrichtung „SC – Verteilte Systeme und Komponentensoftware“ des Deutschen Zentrums für Luft- und Raumfahrt waren Abteilungsleiter Dipl.-Math. Andreas Schreiber und Dipl.-Inform. Markus Litz.

## 1.1 Motivation

Das deutsche Zentrum für Luft- und Raumfahrt ist eine Forschungseinrichtung der Bundesrepublik Deutschland mit den Schwerpunkten Luftfahrt, Raumfahrt, Energie und Verkehr. Das Spektrum der Forschungen reicht von der Grundlagenforschung bis hin zu fertigen Produkten und Anwendungen. Hierbei werden sowohl nationale, internationale als auch industrielle Kooperationen umgesetzt.

Wissenschaftliche Einrichtungen müssen in der Regel eine große Menge von Daten verwalten. Hierbei sind üblicherweise sowohl die Menge der Daten als auch die vielen verschiedenen Formate und Prozesszugehörigkeiten der Daten problematisch. Zu einem Experiment gehören zum Beispiel Dokumentationen der verschiedenen Arbeitsschritte, Eingabe- oder Messdaten, Modelle sowie Ergebnisdaten. Zusätzlich können die Daten noch in verschiedenen Versionen vorliegen. Für das Management dieser Daten wird im DLR auf Clientseite die Anwendung „DataFinder“ auf Grundlage des WebDAV-Protokolls eingesetzt. Als Server kann dafür jede Server-Software eingesetzt werden, welche die WebDAV-Spezifikation [RFC4918] hinreichend implementiert. Neben kommerziellen Servern (z. B. „Tamino“ der Software AG oder „SharePoint“ von Microsoft) werden zunehmend Open Source Produkte eingesetzt. Im DLR wird daher an dem Open Source WebDAV-Server „Catacomb“ mitentwickelt. Für das Entwickeln des Servers und des Clients sind genaue und automatisierte Tests sowie eine einfache Möglichkeit zur Fehlersuche eine wichtige Hilfe.

## 1.2 Aufgabenstellung

Im Rahmen der Diplomarbeit sollen Konzept und Design für ein Open Source Werkzeug realisiert werden, mit dessen Hilfe WebDAV-Anwendungen grafikunterstützt getestet und ausgewertet werden können. Dabei soll das Werkzeug als Proxy zwischen Client und Server fungieren, um eine Analyse der Kommunikation zwischen Server und Client zu ermöglichen. Die Realisierbarkeit soll durch eine prototypische Implementierung des Werkzeugs belegt werden. Lasten- und Pflichtenheft sind obligatorischer Bestandteil der Arbeit (siehe Anhang).

Weiterhin soll eine Marktstudie (siehe Anhang) durchgeführt werden, in deren Rahmen zu ermitteln ist, welche Werkzeuge bereits auf dem Markt angeboten werden und ob eines dieser Werkzeuge für die geplanten Erweiterungen als Basis genutzt werden kann oder ob eine Eigenentwicklung nötig wird. Falls Letzteres der Fall sein sollte, muss noch eine Entscheidung über die zu verwendende Programmiersprache gefällt werden.

## 1.3 Gliederung

Kapitel 2 befasst sich mit den verwendeten Technologien und Prinzipien. Dieses Kapitel soll dem Leser das nötige theoretische Basiswissen vermitteln, um sich mit den restlichen Teilen der Ausarbeitung auseinandersetzen zu können. Kapitel 3 erläutert die Ideen und Überlegungen, die der Umsetzung zugrunde liegen. Danach werden in Kapitel 4 das Design und die Implementierung der Anwendung betrachtet. Besonderes Augenmerk wird dabei auf die Realisierung der in Kapitel 3 vorgestellten Konzepte gelegt. Kapitel 5 stellt die Ergebnisse der validierten Arbeit vor. Weiterhin wird ein Anwendungsszenario der Software präsentiert. Kapitel 6 bildet den Abschluss der Ausarbeitung und fasst alle Ergebnisse zusammen. Des Weiteren werden bei der Erstellung der Anwendung gesammelte Erfahrungen und mögliche zukünftige Entwicklungen aufgezeigt.

# 2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen, die für das Verständnis der Arbeit und die Nachvollziehbarkeit der Entscheidungen nötig sind, erörtert. Hierbei wird auf die jeweiligen Technologien und Konzepte in nötiger Detailtiefe eingegangen. Für weiterführende Informationen sei der Leser auf die angegebene Literatur verwiesen.

## 2.1 Software Engineering

Als Konsequenz aus der sogenannten „Softwarekrise“ der 1960er Jahre [SOM01] wurden Techniken und Methoden entwickelt, die es dem Ersteller von Software erlauben, die Kontrolle über das Produkt auch bei komplexen und umfangreichen Aufgaben zu behalten. So ist der damals geprägte Begriff „Software Engineering“ mittlerweile nicht nur als technische Disziplin zu verstehen, sondern Software Engineering umfasst zum Beispiel auch Projektverwaltung oder die Entwicklung von Methoden und Werkzeugen zur Unterstützung bei der Durchführung von Softwareprojekten. Weiterhin deckt Software Engineering alle Phasen des Zustandes einer Software ab, beginnend mit der Erstellung der Anforderungen bis hin zur Wartung des fertigen Produktes. Die verschiedenen Konzepte des Software Engineerings befinden sich in einer stetigen Weiterentwicklung, und die Einführung moderner Paradigmen, wie etwa das der Objektorientierung, ermöglichen ein intuitives Vorgehen und stellen den Menschen und seine Wahrnehmung in den Mittelpunkt der Betrachtung. Die Verfahren der objektorientierten Analyse und des objektorientierten Designs erlauben in Verbindung mit der Unified Modeling Language (UML) heutzutage nahtlose Übergänge zwischen den verschiedenen Sichten und Phasen der Modelle. Zudem erleich-

tert die objektorientierte Entwicklung die Wiederverwendung von bereits erstellten Komponenten.

Die Methoden des Software Engineerings beinhalten Modellbildungen, Notationen und Regeln, Anleitungen und Vorschläge. Sie ermöglichen dem Softwareentwickler ein systematisches und strukturiertes Vorgehen bei der Erstellung und Betreuung einer Software. Allerdings gibt es keine allgemeingültige, ideale Kombination dieser Methoden, sondern der Entwickler muss für das jeweilige Projekt die passenden Methoden und Hilfsmittel wählen. Dies kann je nach Projektumfeld zu sehr unterschiedlichen Zusammenstellungen von Methoden führen.

Im Folgenden wird näher auf die im Rahmen dieser Arbeit verwendeten Vorgehensmodelle und Methoden zur Qualitätssicherung eingegangen.

### 2.1.1 Vorgehensmodelle

Ein Vorgehensmodell betrachtet ein Projekt aus einem bestimmten Blickwinkel und unterteilt es in verschiedene Abschnitte. Dies hat zur Folge, dass mit einem Vorgehensmodell nie alle Details eines Projektes erfasst werden können. Die Verwendung eines solchen Modells bietet dem Nutzer jedoch den Vorteil, die komplexen Zusammenhänge eines großen Projektes zu strukturieren und somit beherrschbar zu gestalten. Im Folgenden werden die für dieses Projekt wichtigen Prozessmodelle kurz erläutert.

**Wasserfall-Modell**
Das Wasserfall-Modell ist ein sehr bekanntes Vorgehensmodell und wird alternativ auch als Softwarelebenszyklus bezeichnet [ROY70]. Alle wichtigen Aktivitäten bei der Softwareentwicklung sind hierbei in einer festen Reihenfolge und ohne Wiederholungen kaskadiert: Analyse und Spezifikation, Entwurf, Implementierung, Test, Betrieb und Wartung. Das Hauptproblem dieses Modells ist die starre Aufteilung in die verschiedenen Phasen. Dies

hat zur Folge, dass sehr früh die genauen Spezifikationen fest stehen müssen und die Anpassung von Anforderungen zu einem späteren Zeitpunkt nur schwierig durchzuführen ist.

**Evolutionäre-Entwicklung**
Bei diesem Vorgehensmodell wird aufgrund von groben Vorgaben ein Prototyp der Software erstellt. Dieser wird dem Kunden zur Verfügung gestellt und von ihm kritisiert. Auf Basis dieser Kritik werden die Anforderungen verfeinert und ein weiterer Prototyp erstellt. Das Modell lässt sich in zwei Kategorien unterteilen (siehe Bild 2.1):

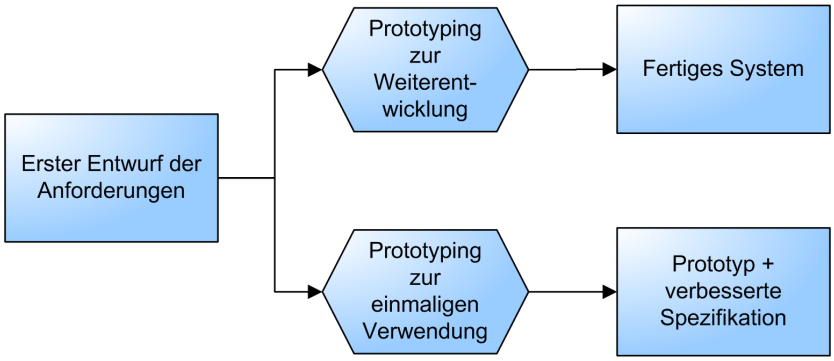


Bild 2.1: Arten von Prototyping nach [SOM01]

1. Evolutionäre Prototypen: Hierbei werden die erstellten Prototypen aufgrund der genaueren Anforderungen bei jedem Schritt verbessert. Endergebnis dieses Prozesses ist das fertige System.

2. Explorative-Prototypen: Die Explorativen-Prototypen dienen dazu die Anforderungen des Kunden zu spezifizieren und werden nach der Benutzung nicht weiter verwendet. Diese Prototypen werden auch als Wegwerf-Prototypen bezeichnet.

Wenn man die erstellten Prototypen als Produkte betrachtet, kann man diese auch anhand der Art des Produktes charakterisieren [KLO07]:

- Fachliche Prototypen: Hierbei werden Teile der fachlichen Anforderung realisiert und einer Überprüfung unterzogen.

- Labormuster: Mit Labormustern können Architekturen und Techniken auf Programmierebene validiert werden.
- Demonstrationsprototyp: Diese Art von Prototyp dient dem Test von nicht funktionaler Logik und des Look & Feels einer Anwendung.
- Pilotsystem: Hierbei wird ein Basissystem realisiert und inkrementell zum fertigen Produkt ausgebaut.

**Entwicklung unter Wiederverwendung**
Bei diesem Modell wird davon ausgegangen, dass bereits eine große Anzahl von fertigen Komponenten zur Verfügung steht. So ist der Haupteinsatzzweck dieses Modells die Suche und der Einsatz von passenden Komponenten für das zu entwickelnde System. Hierbei werden die Anforderungen teilweise an die Eigenschaften vorhandener Komponenten angepasst. Dieser Prozess ermöglicht nicht immer die vollständige Abdeckung der Kundenwünsche, bietet jedoch eine Verringerung der Kosten und Risiken durch die Verwendung bereits fertiger Komponenten.

Die hier aufgeführten Vorgehensmodelle besitzen je nach Projekt und Aufgabe Vor- und Nachteile. Allen drei Modellen gemeinsam ist das Fehlen einer übergreifenden Wiederholung der Prozessschritte. Als Konsequenz daraus wurden zwei Ansätze entwickelt, die speziell Wiederholungen berücksichtigen und verschiedene der Entwicklungsmodelle unterstützen:

**Inkrementelle Entwicklung**
Bei der inkrementellen Entwicklung wird versucht die Vorteile der evolutionären Entwicklung und die des Wasserfall-Modells zu verbinden. Hierbei erfolgt zu Beginn lediglich eine grobe Spezifikation und danach eine Aufgliederung des Problems in Teilsysteme. Diese werden dann nach Priorität geordnet umgesetzt. Die Realisierung der Teilsysteme kann dabei je nach Reifegrad der Spezifikationen evolutionär oder dem Wasserfall-Modell entsprechend erfolgen. Ein fertig gestelltes Teilsystem wird dem Kunden zur Verfügung gestellt, und die daraus resultierenden Erfahrungen können für weitere Teilsysteme und die Verbesserung der Spezifikationen genutzt werden.

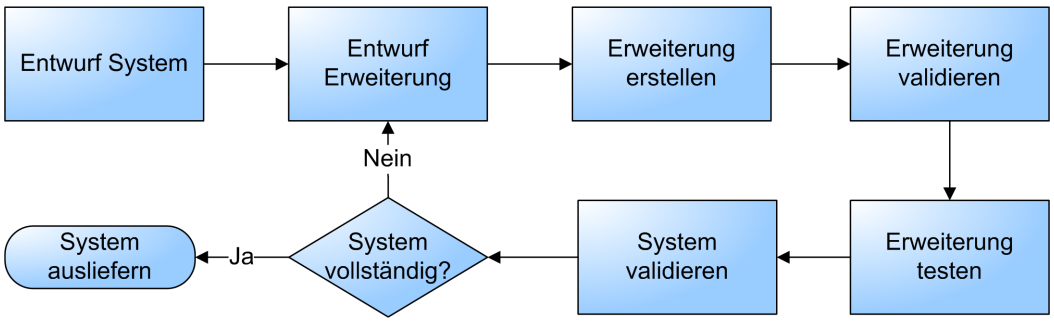


Bild 2.2: Inkrementelle Softwareentwicklung, angelehnt an [SOM01]

Dieses Vorgehen wird in Bild 2.2 veranschaulicht. Eine Weiterentwicklung dieses Ansatzes ist die „Extremprogrammierung“ [BEC00].

**Spiralförmige Entwicklung**
Das Spiralmodell wurde 1988 von Boehm vorgeschlagen und ist ein sehr verbreitetes Vorgehensmodell [BOE88]. Hierbei wird die Entwicklung in vier Segmente unterteilt:

1. Zieldefinition
2. Risikoabschätzung
3. Entwicklung und Validierung
4. Planung

Die einzelnen Phasen des Softwareprozesses repräsentieren jeweils eine Windung der Spirale und bauen aufeinander auf. Eine Phase besteht jeweils aus den oben aufgeführten vier Segmenten. Weiterhin kann für jede Phase ein passendes Entwicklungsmodell gewählt werden. Als einziges Vorgehensmodell beinhaltet das Spiralmodell eine Risikoabschätzung. Dies kann sich insbesondere in Verbindung mit einem entsprechenden Projektmanagement als Vorteil erweisen.

Allgemein sei noch angemerkt, dass bei den Ansätzen der evolutionären Entwicklung die Systemspezifikationen erst mit dem Fortschreiten des Projektes verfeinert und vervollständigt werden. Dies verhindert die Verwendung dieser Modelle bei Verträgen, die die Systemspezifikationen bereits zu Beginn vollständig enthalten müssen [SOM01].

Es existieren noch weitere Vorgehensmodelle und im Folgenden werden zwei davon kurz betrachtet und ein Zusammenhang zu den bisher diskutierten Modellen hergestellt [MUE99] [BUH04]:

**V-Modell 97**
Das V-Modell wurde im Auftrag der Bundeswehr der Bundesrepublik Deutschland entwickelt, und später wurden auch alle Behörden zur Verwendung dieses Modells verpflichtet. Das V-Modell ist eine komplexe Weiterentwicklung des Wasserfall-Modells und besteht aus vier verschiedenen Submodellen: System-/Softwareerstellung (SE), Projektmanagement (PM), Qualitätssicherung (QS) und Konfigurationsmanagement (KM). Die Submodelle sind stark miteinander verwoben, und deren Prozesse können für jedes Projekt individuell angepasst werden. Dieses Modell spezifiziert zudem Entwicklungs- und Methodenstandards sowie Anforderungen an die verwendeten Werkzeuge.

**Rational Unified Process (RUP)**
Dieses Vorgehensmodell wurde von der Firma Rational entwickelt und ist durch zwei zueinander orthogonale Einteilungen strukturiert. Die statische Einteilung wird dabei als „Methodenachse“ bezeichnet und setzt sich aus verschiedenen Prozessbereichen zusammen. Es existieren vier Hauptprozessbereiche: Erfassung der Anforderungen, Analyse & Design, Implementierung und Test. Weiterhin gibt es drei unterstützende Bereiche: Management, Verteilung und Umgebung. Die zweite Einteilung nimmt eine zeitliche Aufteilung des Projektes vor. Hierbei werden vier Phasen unterschieden: Zieldefinition, Entwurfsphase, Realisierung und Übergangsphase. Jede Phase wird dabei in mehrere Iterationen aufgeteilt, und in jeder Phase werden Methoden aus unterschiedlichen Prozessbereichen angewendet, ähnlich einem vereinfachten Wasserfall-Modell. Dieses Vorgehensmodell kann an die Anforderungen des jeweiligen Projektes angepasst werden. In Bild 2.3 ist dieses Vorgehensmodell schematisch dargestellt.

Beide Vorgehensmodelle unterstützen objektorientierte Softwareentwicklung und iteratives Vorgehen. Zudem ist in beiden Modellen die Nutzung von UML-Techniken vorgesehen beziehungsweise bei RUP ein integraler Bestandteil.

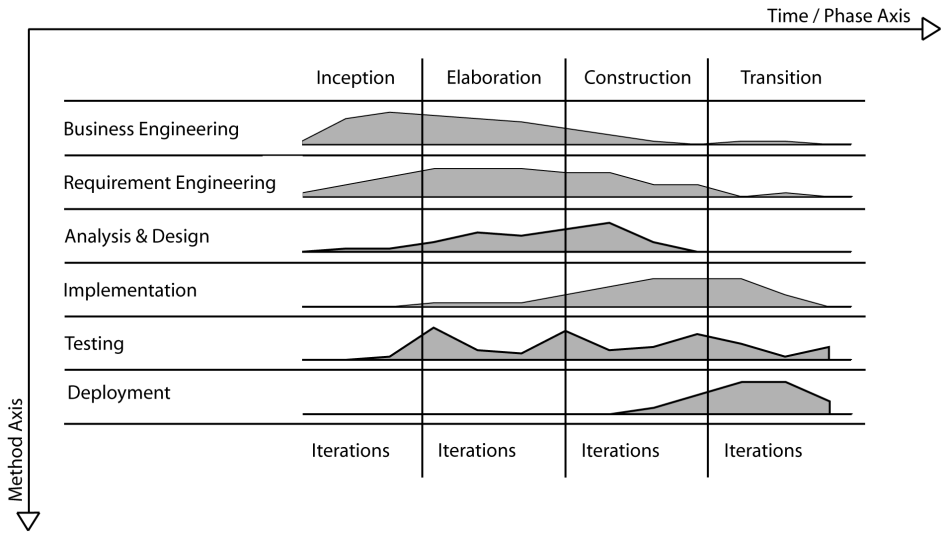


Bild 2.3: Rational Unified Process (RUP)

Weiterhin definieren beide Modelle im Zuge des Projektmanagements Vorgehen zur Risikoabschätzung und Risikominimierung. Qualitätssicherung und Änderungsmanagement werden beim V-Modell 97 als eigenes Submodell in den Prozess integriert. Wiederverwendung erlaubt dieses Vorgehensmodell durch die Integration von Fertigprodukten. Beim RUP werden formale Regeln und Workflows definiert, die die Qualität und die Konsistenz der Prozessergebnisse sicherstellen sollen. Die Verwendung von komponentenbasierten Architekturen ermöglicht Wiederverwendung bei Nutzung des Rational Unified Process.

### 2.1.2 Qualitätssicherung

> *„Unter Softwarequalität versteht man die Gesamtheit der Merkmale und Merkmalswerte eines Softwareproduktes, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.“* [BAL98]

Dieses Zitat bezieht sich auf die Produktqualität, die Prozessqualität ist jedoch ebenso wichtig. Denn es ist davon auszugehen, dass mittels qualitativ hochwertiger Prozesse auch hochwertige Produkte erstellt werden können.

Sowohl auf die Produktqualität, als auch auf die Prozessqualität soll im Folgenden näher eingegangen werden.

Zunächst aber werden Merkmale, die gute Software auszeichnen, definiert. Neben der offensichtlichen Funktionalität spielen das Laufzeitverhalten, die Struktur der Quellen und die Dokumentation eine ebenso große Rolle für die Qualität. Welche Kriterien im Einzelnen an die Software angelegt werden, hängt zudem noch von dem gewünschten Einsatzgebiet der Software ab. Die wichtigsten Qualitätsmerkmale sind [ISO9126]:

- **Funktionalität**
  Das Merkmal „Funktionalität“ spiegelt wieder, ob und in welcher Qualität eine Software den geforderten Funktionsumfang erfüllt.
- **Änderbarkeit**
  Es ist mit hoher Wahrscheinlichkeit davon auszugehen, dass Software während ihres Lebenszyklus an geänderte Anforderungen angepasst oder erweitert werden muss. Deshalb sollte gute Software von Beginn an so realisiert werden, dass Modifikationen ohne großen Aufwand durchgeführt werden können.
- **Zuverlässigkeit**
  Die Software sollte keine Daten verfälschen oder durch Fehlfunktionen Schäden verursachen.
- **Effizienz**
  Die Software sollte möglichst effizient arbeiten, also keine Systemressourcen verschwenden.
- **Benutzbarkeit**
  Dieses Merkmal trifft hauptsächlich auf die Benutzerschnittstelle zu. So sollte eine Software sich an bestimmte Standards der Benutzerführung halten und eine ausreichende Hilfestellung für Benutzer bieten.
- **Übertragbarkeit**
  Dieses Merkmal gibt wieder, ob die Software auch in anderen Systemumgebungen (Hard- und Software) verwendet werden kann.

Diese Merkmale können nochmals je nach ihrer Bedeutung für die Software und ihren Einsatzzweck gewichtet werden.

Um die Einhaltung dieser Merkmale sicherzustellen, gibt es verschiedene Methoden. Diese können wiederum kombiniert werden. Wie bereits im vorherigen Kapitel erwähnt, ist die iterative Softwareentwicklung eine dieser Methoden.

**Testgetriebene-Entwicklung**
Ziel dieser Methode ist es bereits vor der Erstellung des eigentlichen Programmcodes Tests zu definieren. Diese werden genutzt, um bei der Entwicklung des Programms regelmäßig und teilweise automatisiert die Funktion des Programms zu überprüfen. Eine konkrete Umsetzung dieser Technik sind zum Beispiel Modultests. Diese werden auch als Unit-Tests bezeichnet.

**Reviews**
Ein Review ist ein Treffen einer Gruppe von Entwicklern. Diese analysieren gemeinsam zum Beispiel Teile des Quellcodes eines Programms oder die Architektur eines Systems. Durch dieses Vorgehen können Fehler entdeckt werden, und ein einheitlicher Stil bei der Entwicklung wird gefördert. Ein zusätzlicher Nutzen ist die dabei stattfindende Verteilung von Wissen auf alle Teilnehmer. Als Ergebnis eines solchen Treffens entsteht, per Hand oder mit Werkzeugunterstützung, ein Dokument, in dem die Kritikpunkte zusammengefasst sind [PRE87].

**Standards**
Die Anwendung von Standards kann in vielen Bereichen des Software Engineerings helfen konkrete Qualitätskriterien umzusetzen. Die Standards lassen sich wiederum in zwei Gruppen aufteilen: Produkt- und Prozessstandards. Produktstandards betreffen zum Beispiel den Stil der Codierung und Dokumentation oder eine einheitliche Gestaltung von Schnittstellen. Prozessstandards garantieren einen einheitlichen Ablauf von Tätigkeiten, wie etwa dem Erstellen einer Software-Version. Standards stellen geeignete Praktiken und Richtlinien zur Verfügung, die eine Stetigkeit im Lebenszyklus der Software schaffen. Dadurch fällt es anderen Entwicklern leichter, sich in Projekte wieder oder neu einzuarbeiten [SOM01].

## 2.2 Open Source

> *„Ich denke, dass jede allgemein nützliche Information frei sein sollte. Mit „frei“ beziehe ich mich nicht auf den Preis, sondern auf die Freiheit, Informationen zu kopieren und für die eigenen Zwecke anpassen zu können. Wenn Informationen allgemein nützlich sind, wird die Menschheit durch ihre Verbreitung reicher, ganz egal, wer sie weitergibt und wer sie erhält.“*
>
> [GRA04, Richard Stallman, ca. 1990]

Die Begriffe „Open Source“ und „freie Software“ sind eng miteinander verwandt, und oft sind die genauen Definitionen und Unterschiede den meisten Anwendern und Entwicklern nicht bekannt. Deswegen soll an dieser Stelle eine genaue Definition der beiden Begriffe vorgenommen werden. In diesem Zusammenhang sind zwei Organisationen von besonderer Bedeutung. Zum einen ist dies die „Free Software Foundation“ (FSF, [FSF08]), eine Stiftung zur Förderung der freien Software. Zum anderen existiert die „Open Source Initiative“ (OSI, [OSI08]), eine Organisation zur Förderung von Open Source Software. Einen umfassenden Überblick zu diesem Thema gewährt Volker Grassmuck in seinem Buch „Freie Software – Zwischen Privat- und Gemeineigentum“ ([GRA04]).

### 2.2.1 Definition

In diesem Kapitel werden insgesamt drei Definitionen gegeben. Zum einen die Definition des Begriffs „freie Software“, die des Begriffs „Open Source“ und die Definition des Begriffs „Copyleft“. Zum anderen soll der Unterschied zwischen freier Software und Open Source verdeutlicht werden.

„Freie Software“ wurde von Richard Stallmann im Zuge des GNU-Projektes folgendermaßen definiert:

- *Freiheit 0: Das Programm darf zu jedem Zweck ausgeführt werden.*
- *Freiheit 1: Das Programm darf studiert und verändert werden.*
- *Freiheit 2: Das Programm darf verbreitet werden.*
- *Freiheit 3: Das Programm darf verbessert und verbreitet werden, um damit einen Nutzen für die Gemeinschaft zu erzeugen.*

[GFS08, Definition freier Software nach GNU]

Die Definition von „Open Source“ nach der Open Source Initiative fällt etwas umfangreicher aus. Lizenzen, die die Definition erfüllen, dürfen den geschützten Titel „Open Source™“ tragen. Im Folgenden sind die zehn Kriterien frei übersetzt wiedergegeben:

1. *Die Software darf beliebig kopiert, verbreitet und genutzt werden.*
2. *Die Software liegt in einer für den Menschen lesbaren und verständlichen Form vor.*
3. *Die Software darf verändert werden und in der veränderten Form weitergegeben werden, auch unter den gleichen Lizenzbestimmungen wie das Basisprodukt.*
4. *Die Lizenz darf die Weitergabe von verändertem Quellcode nur einschränken, wenn „Patch-Dateien“ erlaubt sind. [. . . ]*
5. *Es dürfen keine Personen oder Gruppen diskriminiert werden.*
6. *Das Anwendungsgebiet darf nicht eingeschränkt werden.*
7. *Die Lizenz muss in sich abgeschlossen sein.*
8. *Die Lizenz muss produktneutral sein.*
9. *Die Lizenz ist nur auf den Geltungsbereich der lizenzierten Software anwendbar.*
10. *Die Lizenz muss technologieneutral sein.*

[OSD08, Definition freier Software nach OSI (OSD), verkürzt]

Die Definition des Begriffs „Open Source“ basiert auf der Definition von „freier Software“. Jedoch ist der Begriff der Freiheit in der ursprünglichen Definition von „freier Software“ wesentlich schärfer formuliert. Da außerdem frei oftmals mit kostenlos verwechselt wurde, ist der Begriff auch zwecks besserer Vermarktung eingeführt worden. Das sollte helfen diese Art von Software auch bei kommerziellen Entwicklern zu etablieren.

Die Definition des Begriffs „Copyleft“ ist für das Verständnis der weiter unten aufgeführten Lizenzen wichtig. Deshalb folgen nun einige Erläuterungen hierzu. Der Begriff „Copyleft“ entstammt dem Umfeld des Urheberrechts. Im Gegensatz zum „Copyright“, das kopieren, verändern und weiterverbreiten von geschütztem Material ausdrücklich verbietet, erlaubt das Copyleft dieses explizit. Weiterhin fordert das Copyleft bei Modifizierung von ursprünglich mit Copyleft versehenem Material von dem veränderten Werk die gleichen Rechte ein wie von dem Ursprungswerk. Damit soll verhindert werden, dass ursprünglich freie Werke als Basis für unfreie Werke genutzt werden können. Um diese Forderung durchzusetzen, nutzt das Copyleft interessanterweise das Copyright. Ausführlichere und weitergehende Informationen zum Copyleft können [GCP08] entnommen werden.

### 2.2.2 Lizenzmodelle

Im Folgenden werden die wichtigsten Open Source Lizenzmodelle (weitere unter [OSL08]) aufgeführt und kurz erläutert. Diese kurze Übersicht erhebt allerdings keinen Anspruch auf Vollständigkeit.

- **GPL** – GNU General Public License
  Die GNU General Public License ist wohl die bekannteste Open Source Lizenz und wurde von der Free Software Foundation [FSF08] formuliert. Das Werk des Autors wird durch diese Lizenz geschützt, und darauf aufbauende Werke müssen ebenfalls zwingend unter GPL veröffentlicht werden. Weiterhin muss der Quellcode einer unter GPL stehenden Applikation veröffentlicht werden. Diese Bedingungen führen

| Lizenz | GPL | LGPL | BSD | Apache |
|---|---|---|---|---|
| Mischung mit kommerziellem Code | Nein | Ja | Ja | Ja |
| bei Mischung muss Quellcode beiliegen | Ja | Ja | Nein | Nein |
| Copyleft | Ja | Ja | Nein | Nein |
| GPL-kompatibel (v3) | Ja | Ja | Nein | Ja |
| OSI-kompatibel | Ja | Ja | Ja | Ja |

Tabelle 2.1: Open Source Lizenzen, Vgl. [NUE00] und [GWO06]

dazu, dass diese Art der Lizenz bei kommerziellen Softwareentwicklern unbeliebt ist. Weitere Informationen können [GPL3] entnommen werden.

- **LGPL** – GNU Lesser General Public License
  Die GNU Lesser General Public License ermöglicht durch einen Passus auch kommerziellen Entwicklern die Verwendung dieser Lizenz. Anders als bei der GPL ist es bei der LGPL erlaubt, eine Bibliothek, die unter LGPL steht, für ein kommerzielles Produkt zu nutzen. Der hierbei entstandene Quellcode muss nicht veröffentlicht werden und nicht unter LGPL stehen. Siehe auch [LGPL3].

- **BSD** – Berkley Software Distribution
  Die BSD-Lizenz ist ebenfalls für kommerzielle Nutzer geeignet und überzeugt durch Kürze und Einfachheit. Sie erlaubt die Verwendung, Weitergabe und Modifikation des Produktes. Lediglich die Lizenz selbst, Vermerke im Quellcode und entsprechende Dokumentation müssen vollständig vorhanden sein. Die BSD-Lizenz bildet unter anderem die Basis für die Apache Lizenz. Weitere Informationen unter [BSD08].

- **Apache** – Apache License, Version 2.0
  Die Apache Lizenz ist ebenfalls eine einfache und kurze Lizenz. Die Verwendung, Verteilung und Modifikation von Software unter Apache Lizenz ist frei. Der Quellcode eines Produktes muss nicht im Paket enthalten sein und Produkte, die auf Bibliotheken unter Apache Lizenz basieren, müssen nicht zwingend ebenfalls unter Apache Lizenz

veröffentlicht werden. Der genaue Wortlaut ist unter [APA07] einsehbar. Weitere Informationen zu den Unterschieden zwischen der Apache Lizenz und der GPL sind in [ASF08] verfügbar.

Alle Lizenzen besitzen einen Abschnitt, der einen Haftungssauschluss enthält. Einen kurzen Überblick gibt Tabelle 2.1.

### 2.2.3 SourceForge.net

Das Webportal „SourceForge.net“ (engl. Quellcodeschmiede), im Weiteren als SF bezeichnet, ist ein Anwendungsdienstleister für Open-Source-Projekte. Bei der Realisierung von Open-Source-Projekten werden die Software-Entwickler durch eine reichhaltige Auswahl an Diensten unterstützt. Die Nutzung dieser Dienste ist kostenlos [SFN08]. Betrieben wird das Portal von der Firma SourceForge. Im Folgenden werden die wichtigsten der Dienste aufgeführt:

- Softwarekonfigurationsmanagement (SCM): CVS oder Subversion
- Tasks: Verwaltung und Planung von Aufgaben
- Wiki: Information und Dokumentation
- Forum: Diskussion und Dokumentation
- Mailinglisten: Information und Diskussion
- Bug-Tracker: Erfassung von Fehlern und Änderungswünschen
- Webspace: Speicherplatz für die Homepage des Projektes

Die von SF zur Verfügung gestellte Infrastruktur ermöglicht den Entwicklern sich auf ihre eigentliche Tätigkeit, die Realisierung von Software, zu konzentrieren. Weiterhin ist SF eine Kommunikationsplattform für Entwickler, aber auch Anwender können und sollen in Projekte integriert werden. Dadurch soll die Qualität und Verfügbarkeit einer Software über einen langen Zeitraum verbessert werden.

## 2.3 Architektur- und Entwurfsmuster

> *„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass Sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen“*
>
> Christopher Alexander in [ALE77]

Architektur- und Entwurfsmuster (engl. design patterns) sind bewährte Lösungsvorschläge und Schemata für bestimmte Architektur- und Entwurfsprobleme. Die Bezeichnung „Entwurfsmuster“ stammt ursprünglich aus dem Bereich der Architektur. Diesem Bereich ist auch obiges Zitat entnommen. Der Begriff wurde dann in Anlehnung an die ursprüngliche Verwendung in der Softwareentwicklung eingeführt [QUI99]. Ein Muster bietet dem Entwickler die Möglichkeit, mit einem erprobten Konzept jeweils eine bestimmte Klasse von Problemen in der Entwicklung zu lösen. Weiterhin können durch die Verwendung von Mustern abstraktere Strukturen sichtbar werden, und innerhalb einer Entwicklergruppe wird eine gemeinsame Begriffsbasis geschaffen. Diese Basis ist als Grundlage für Diskussionen unter Entwicklern und die Einarbeitung neuer Entwickler von Vorteil. Im Allgemeinen sind Muster zudem unabhängig von der verwendeten Programmiersprache [BRD04].

Muster können, wie die Überschrift auch schon andeutet, nochmals in verschiedene Kategorien eingeteilt werde. So gibt es nicht mehr nur die „klassischen“ Entwurfsmuster, sondern etwa auch Muster für die Analyse, Architektur, Kommunikation und weitere. Die Klassifizierung hängt dabei von der Granularität und Abstraktionsebene des jeweiligen Musters ab. Architekturmuster beinhalten zum Beispiel komplette Software-Architekturen in ihren Lösungsvorschlägen.

Eine ganz eigene Gruppe von Mustern stellen die Antimuster (engl. anti patterns) dar. Diese Muster haben das Gegenteilige zum Inhalt, denn sie dokumentieren fehlerhafte Lösungen. Sie können als Beispiele betrachtet werden, wie man ein Problem *nicht* lösen sollte. Antimuster sollen daher als

Negativbeispiel ebenso zur Verbesserung von Softwarequalität beitragen wie die eigentlichen Muster [FRE04].

Im Folgenden wird auf vier Muster näher eingegangen, die jeweils in dem Standardwerk für Entwurfsmuster „Design Patterns - Elements of Reusable Object-Oriented Software“ der „Gang of Four“ (GOF) – auch als „Viererbande“ bezeichnet – beschrieben sind [GAM96]. Diese wurden im Rahmen der Arbeit verwendet und sollen kurz diskutiert werden.

### 2.3.1 Einzelstück-Muster

Das Einzelstück-Muster (engl. singleton pattern) ist ein GOF Entwurfsmuster, genauer ein Erzeugermuster, und wird zum Beispiel bei der Realisierung von Protokollklassen (Logger) verwendet. Das Singleton-Entwurfsmuster stellt sicher, dass von einer Klasse genau nur eine Instanz erzeugt werden kann. Dies wird durch eine dedizierte Methode zum Zugriff auf diese Klasse erzwungen.

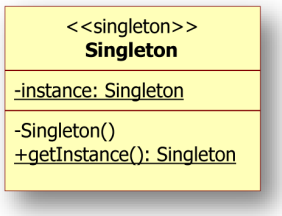


Bild 2.4: Das Einzelstück-Muster

In Bild 2.4 ist die UML-Notation einer solchen Klasse dargestellt. Wie man dem Bild entnehmen kann, besitzt diese Klasse keinen öffentlichen Konstruktor, sondern die Klasse ist selbst für die Instanziierung verantwortlich. Sowohl die Instanz als auch die Zugriffsmethode auf diese Instanz sind statisch und damit auch global zugänglich.

### 2.3.2 Beobachter-Muster

Das Beobachter-Muster (engl. observer pattern) ist ebenfalls ein GOF-Entwurfsmuster, in diesem Fall ein Verhaltensmuster. Dieses Muster wird auch als „Veröffentlicher/Abonnenten"-Muster bezeichnet (engl. publisher/-subscriber pattern) und dient der Weitergabe einer Änderung des Zustandes von einem Objekt an viele Objekte. Zur Umsetzung dieser Aufgabe wird der Ansatz der losen Kopplung verwendet. Dieses Konzept sorgt für eine möglichst geringe Abhängigkeit der Komponenten voneinander. Dadurch lassen sich diese Objekte leichter trennen und flexibler kombinieren. In der Praxis wird dies durch die Nutzung von Schnittstellen (engl. interface) realisiert.

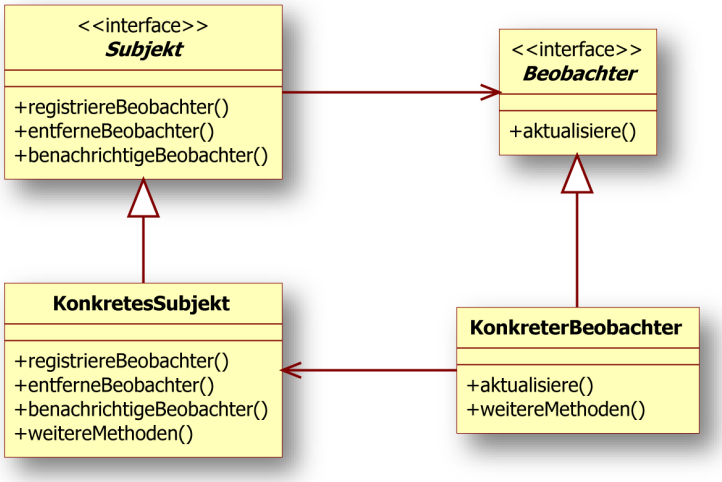


Bild 2.5: Das Beobachter-Entwurfsmuster

In Bild 2.5 ist die UML-Notation dieses Musters dargestellt. Das Subjekt hat einen Zustand, der für andere Objekte von Interesse ist. Das Subjekt besitzt Methoden zum An-, Abmelden und Benachrichtigen von Beobachtern. Damit die Beobachter sich registrieren können, müssen sie die entsprechende Beobachter-Schnittstelle verwirklichen. Diese Schnittstelle beinhaltet normalerweise nur die Methode „aktualisiere()". Wenn sich der Zustand des Subjekts geändert hat, wird diese Methode vom Subjekt bei allen angemeldeten Beobachtern aufgerufen.

### 2.3.3 Dekorierer-Muster

Ein weiteres Muster der „Gang of Four“ ist das Dekorierer-Muster. Es erlaubt das dynamische Hinzufügen von Zuständigkeiten zu einem Objekt und gehört zur Kategorie der Strukturmuster (engl. structural pattern). Um die Funktionalität einer Klasse zu erweitern, kann auch Unterklassenbildung verwendet werden, jedoch bietet das Dekorierer-Muster den Vorteil, das zur Laufzeit gesteuert werden kann, wann und wie die Funktionalität erweitert wird. Dabei wird das Basisobjekt von einem anderen Objekt, dem Dekorierer, umschlossen. Methoden können durch den Dekorierer entweder in ihrer Funktionalität verändert werden oder an das Basisobjekt delegiert werden. Dekorierer können geschachtelt werden und erlauben so die unbeschränkte Erweiterung der Funktionen eines Objektes.

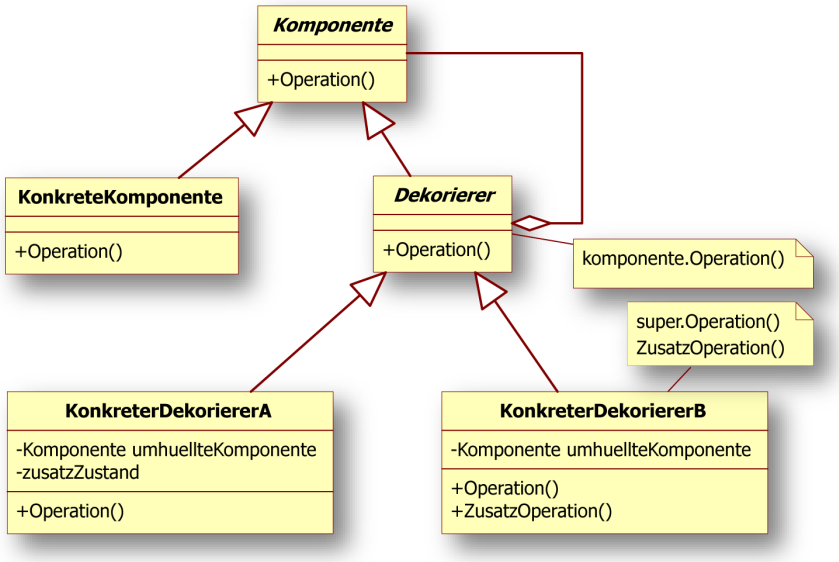


Bild 2.6: Das Dekorierer-Muster

Die UML-Notation veranschaulicht das oben beschriebene Muster in Bild 2.6. Hierbei fällt auf, dass sowohl die Komponenten als auch die Dekorierer die gleiche Schnittstelle besitzen. Dadurch ist die Verwendung von Dekorierern für den Nutzer transparent. Die Aufrufe von Methoden werden an

die Komponente weitergeleitet, zuvor und/oder danach können aber auch weitere Methoden des Dekorierers ausgeführt werden. Dekorierer können sowohl die Funktionalität als auch den Zustand einer Komponente erweitern. Die Dekorierer speichern dabei den Bezug zur Komponente in einer Instanzvariablen.

### 2.3.4 Model-View-Controller-Muster

Das Model-View-Controller-Muster („Modell-Präsentation-Steuerung“) ist ein zusammengesetztes Muster. Es gehört auch nicht zu den Entwurfsmustern, sondern in die Gruppe der Architekturmuster. Es besteht aus drei Komponenten, auch in Bild 2.7 dargestellt:

- **Modell** (Model)
  Das Modell speichert die zu verarbeitenden Daten und Zustände. Je nach Realisierung und Ausprägung kann das Modell auch die Logik und Kernfunktionalität bereitstellen. Das Modell ist unabhängig von den zwei übrigen Teilen und alleine funktionsfähig.

- **Präsentation** (View)
  Die Präsentation stellt die Daten des Modells dar. Bei Änderungen der Daten benachrichtigt das Modell die Präsentation. Diese aktualisiert daraufhin die Darstellung der Daten. Die Präsentation ist auf Modell und Steuerung angewiesen.

- **Steuerung** (Controller)
  Die Steuerung kontrolliert die Präsentation, nimmt die Benutzeraktionen entgegen und verarbeitet diese. Allerdings ändert die Steuerung selbst keine Daten, sondern entscheidet, welches Modell seine Daten aktualisieren muss. Weiterhin kann die Steuerung die in der Präsentation vorhandenen Manipulationsmöglichkeiten für die Daten des Modells einschränken.

Präsentation und Steuerung bilden normalerweise ein Paar. Ein Modell kann von mehreren solcher Paare genutzt werden. Wie oben bereits angemerkt, kombiniert dieses Architekturmuster mehrere Entwurfsmuster.

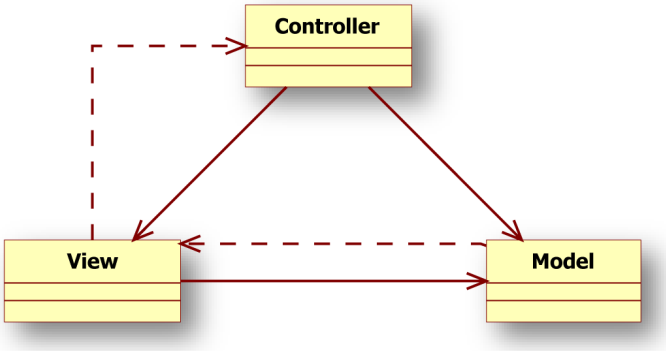


Bild 2.7: Das Model-View-Controller-Muster

Wenn das Modell seine Daten aktualisiert oder seinen Zustand ändert, werden diese Änderungen mittels des Beobachter-Musters gemeldet. Die Präsentation, üblicherweise eine grafische Benutzeroberfläche, realisiert das – in dieser Arbeit nicht näher betrachtete – Kompositum-Muster. In Kurzform ist dies zum Beispiel eine Zusammensetzung von mehreren grafischen Elementen (Eingabefelder, Menüs, etc.) zu einem Formular. Ein ebenfalls nicht in dieser Ausarbeitung betrachtetes Muster nutzt die Steuerung in Verbindung mit der Präsentation. Das hier verwendete Entwurfsmuster wird als Strategie-Muster bezeichnet. Kurz gesagt stellt die Steuerung für die Präsentation eine Strategie für ihr Verhalten bereit. Wird eine andere Steuerung verwendet, resultiert dies in einem anderen Verhalten der Präsentation.

Ziel der Kombination all dieser Muster ist es, die drei Komponenten dieses Architekturmusters so unabhängig wie möglich voneinander zu halten. Dies ermöglicht eine einfache Erweiterung und erleichterte Wiederverwendung der Applikation oder ihrer Module.

## 2.4 Netzwerkprotokolle

Ein Netzwerkprotokoll ist eine Vereinbarung von Regeln und Formaten, die es zwei durch ein Netzwerk verbundenen Kommunikationspartnern ermöglicht, Daten auszutauschen. Da ein einzelnes Protokoll für alle der dazu erforderlichen Aufgaben viel zu komplex wäre, wurden die Aufgaben auf verschiedene Protokolle aufgeteilt. Diese Protokolle sind wiederum in Schichten angeordnet. Es existieren zwei wichtige Modelle für die Anordnung und Aufteilung dieser Schichten, das OSI-Referenzmodell [ISO7498] und das DoD-Schichtenmodell [RFC1122]. Beiden Modellen gemeinsam ist, dass die jeweils höhere Schicht die Funktionen der darunter angeordneten Schicht nutzt. Die dabei entstehende Struktur wird auch als Protokollstapel bezeichnet. In Bild 2.8 ist die Aufteilung der verschiedenen Protokolle auf die Schichten der zwei Modelle dargestellt.

| Schicht | ISO/OSI-Referenzmodell | DoD-Schichtenmodell | Protokolle |
|---|---|---|---|
| 7 | Anwendung | Anwendung | HTTP, WebDAV, FTP |
| 6 | Darstellung | | |
| 5 | Sitzung | | |
| 4 | Transport | Transport | TCP, UDP |
| 3 | Vermittlung | Vermittlung | IP, ICMP, IGMP |
| 2 | Sicherung | Netzwerkzugang | Ethernet, Token Ring, ATM |
| 1 | Bitübertragung | | |

Bild 2.8: OSI-Referenzmodell und DoD-Schichtenmodell

Ein Netzwerkprotokoll legt im Allgemeinen den Aufbau eines Datenpakets, der kleinsten übertragbaren Informationseinheit, fest. Ein solches Paket besteht mindestens aus der Adresse des Absenders und der des Empfängers, dem Pakettyp und den eigentlichen Daten. Des Weiteren kann das Protokoll noch bestimmte Abfolgen von Paketen festlegen, die zum Beispiel zur Verbindungssteuerung genutzt werden.

In diesem Zusammenhang wird kurz auf ein wichtiges Konzept der Netzwerkprogrammierung eingegangen, das Client-Server-Modell. Das Client-Server-Modell ist das am häufigsten verwendete Konzept für die Erstel-

lung von netzwerkfähigen Applikationen. Dabei bietet ein als Server bezeichnetes Programm Dienste an, die von einem oder mehreren als Client bezeichneten Programmen genutzt werden. Ein Aufruf eines solchen Dienstes durch einen Client wird als Anfrage (engl. Request) bezeichnet, die Antwort des Servers auf diesen Aufruf als Antwort (engl. Response). Die Server unterteilt man noch in zwei Kategorien (siehe auch [STE94]):

- **Iterative Server**
  Diese Realisierung eines Servers nimmt eine Anfrage eines Clients an, verarbeitet diese und steht dann für erneute Anfragen zur Verfügung. Gleichzeitige Anfragen werden nacheinander verarbeitet.

- **Nebenläufige Server**
  Nebenläufige Server bearbeiten gleichzeitige Anfragen, indem sie für jede Anfrage einen eigenen Prozess erzeugen. Dieser arbeitet die Anfrage ab und beendet sich dann. (Je nach Implementierung und Betriebssystem können statt Prozessen auch Threads verwendet werden.)

In den folgenden Kapiteln werden jetzt die für diese Arbeit entscheidenden Protokolle vorgestellt und ihre Beziehungen zueinander dargestellt.

### 2.4.1 TCP/IP

Der Ursprung von TCP/IP liegt in den späten 1960ern. Damals wurde durch ein von den Vereinigten Staaten Amerikas finanziertes Forschungsprojekt über paketvermittelte Netzwerke der Grundstein für das heutige Internet gelegt. Deshalb wird TCP/IP im allgemeinen Sprachgebrauch auch häufig als Bezeichnung für den Protokollstapel beim Datenaustausch zwischen verschiedenen Rechnern verwendet. Diese Verwendung entspricht dem DoD-Schichtenmodell aus dem vorherigen Abschnitt. In diesem Kapitel sollen aber nur zwei Protokolle der mittleren Schichten dieses Modells

betrachtet werden, nämlich das Internet Protocol (IP) und das Transmission Control Protocol (TCP). Diese beiden Protokolle spielen im weiteren Verlauf eine wichtige Rolle.

**Internet Protocol – IP**

IP befindet sich auf der Vermittlungsschicht, sowohl im OSI-Referenzmodell als auch im DoD-Schichtenmodell. IP stellt als zentrale Basis für alle weiteren Protokolle einen vom Netzwerkzugang unabhängigen, unzuverlässigen, verbindungslosen Paketvermittlungsdienst bereit. Unzuverlässig bedeutet in diesem Zusammenhang, dass IP nicht garantiert, dass ein Paket auch seinen Empfänger erreicht. Verbindungslos besagt, dass IP auch keine Informationen über den Status der Verbindung speichert oder die Reihenfolge der Pakete beibehält. Weiterhin stellt IP weltweit eindeutige, logische Adressen bereit, die es ermöglichen Netzwerke zu strukturieren und Datenpakete gezielt zu vermitteln (Stichwort: Routing). Hier ein Beispiel für eine gültige IP-Adresse (IPv4): `Adresse: 123.12.1.112` Die detaillierte Definition des Internet Protocols findet sich in [RFC791].

**Transmission Control Protocol – TCP**

TCP ist eine Implementierung der Transportschicht. Auch in diesem Fall besteht kein Unterschied in der Einordnung zwischen dem OSI-Referenzmodell und dem DoD-Schichtenmodell. Im Gegensatz zu IP bietet TCP einen zuverlässigen und verbindungsorientierten Dienst zur Datenübertragung an. TCP kann Fehler in der Übertragung erkennen, stellt die Einhaltung der Reihenfolge der Pakete sicher und bietet Flusskontrolle. Damit Daten mit TCP übertragen werden können, müssen zwei Kommunikationspartner eine Verbindung aufbauen. Nach Beendigung des Datenaustausches wird diese Verbindung geschlossen. Die eindeutige Adressierung eines Kommunikationsendpunktes erfolgt bei TCP durch die Kombination einer IP-Adresse mit einer Portnummer. Die TCP-Adresse eines Webservers könnte zum Beispiel folgendermaßen aussehen: `Adresse: 123.12.1.112:80`. Die Portnummer kann im Bereich von 0 bis 65535 liegen. Der Bereich zwischen 0 und 1023 unterliegt Beschränkungen und wird von der Internet Assigned Numbers Authority (IANA)

verwaltet [IAN08]. Weitere Einzelheiten zu TCP sind unter [RFC793] verfügbar.

Andere Netzwerkprotokolle dieser Schichten sollen aus Gründen der Übersichtlichkeit nicht betrachtet werden (UDP, ICMP, etc.). Hier sei auf die einschlägige Fachliteratur verwiesen ([STE94], [COM95] und [TAN03]).

### 2.4.2 HTTP

Das Hypertext Transfer Protocol (HTTP) ist eines der wichtigsten und meist benutzten Protokolle der Anwendungsschicht der Referenzmodelle. Webserver nutzen HTTP, um die auf ihnen angebotenen Inhalte an die Clients, auch als Browser bezeichnet, auszuliefern. HTTP wurde 1989 von Tim Berners-Lee in Verbindung mit einem Protokoll zum eindeutigen Zugriff auf Ressourcen (siehe URI/URL) und einer Textauszeichnungssprache (siehe HTML) am CERN entworfen und bildet seitdem die Basis des World Wide Web.

Das Hypertext Transfer Protocol kann benutzt werden, um die unterschiedlichsten Inhalte, wie etwa HTML-Dateien, Bilder oder Videos, auszutauschen. Durch die Nutzung des Transmission Control Protocols ist eine sichere, fehlerfreie und in der Reihenfolge der Daten unveränderte Übertragung garantiert.

Eine HTTP-Transaktion besteht immer aus einer Client-Anfrage und einer Server-Antwort. Die Nachrichten, die dabei ausgetauscht werden, bezeichnet man als HTTP-Nachrichten. Dabei enthält eine Client-Nachricht immer eine HTTP-Methode. Diese Methode bestimmt die gewünschte Aktion, die der Server ausführen soll. Zum Beispiel weist die GET-Methode den Server an, eine bestimmte Datei an den Client auszuliefern. Eine Antwort des Servers wiederum enthält immer einen Statuscode. Dieser Statuscode ist eine dreistellige Zahl und zeigt dem Client an, ob seine Anfrage mit Erfolg ausgeführt werden konnte oder ein Fehler aufgetreten ist. So bedeutet ein Statuscode mit dem Wert „200“ zum Beispiel eine erfolgreiche Verarbeitung der Anfrage.

Die Funktionsweise des Protokolls soll an einem kleinen Beispiel verdeutlicht werden. Nachdem der Client die IP-Adresse des Servers ermittelt hat, baut der Client eine TCP-Verbindung zu dem Server auf und sendet die Anfrage gemäß Listing 2.1. Der Server sendet daraufhin die Antwort, dargestellt in Listing 2.2, an den Client zurück. Danach wird die Verbindung geschlossen. Es sei noch darauf hingewiesen, dass die Schritte zur Ermittlung der IP-Adresse zur Vereinfachung weggelassen wurden.

```
1 GET /path/file.html HTTP/1.1
2 Host: www.someserver.net:8080
3 User-Agent: SomeBrowser
```

Listing 2.1: HTTP-Client Anfrage, vgl. [RFC2616]

```
1  HTTP/1.1 200 OK
2  Date: Fri, 25 Jan 2008 23:59:59 GMT
3  Content-Type: text/html
4  Content-Length: 1354
5
6  <html>
7          <body>
8          [more file content ...]
9          </body>
10 </html>
```

Listing 2.2: HTTP-Server Antwort, vgl. [RFC2616]

In den beiden oben ersichtlichen Nachrichten und der Darstellung des allgemeinen Aufbaus in Listing 2.3 lassen sich die elementaren Bestandteile einer HTTP-Nachricht erkennen [GOT02]:

- **Startzeile**
  Die erste Zeile einer jeden HTTP-Nachricht enthält bei einer Anfrage die auszuführende Aktion und die zugehörige Ressource. Bei einer Antwort ist hier der Status der ausgeführten Aktion hinterlegt. In beiden Fällen enthält die Zeile noch die verwendete Protokollversion. In den Beispielen 2.1 und 2.2 entspricht dies jeweils der ersten Zeile.
- **Kopfzeilen**
  Nach der Startzeile können keine oder beliebig viele Kopfzeilen folgen.

Eine Kopfzeile enthält ein Schlüsselwort gefolgt von einem Doppelpunkt und dem zugehörigen Wert (Listing 2.1 Zeilen 2 und 3, Listing 2.2 Zeile 2 bis 4). Das Ende des Nachrichtenkopfes wird immer durch eine Leerzeile angezeigt (Listing 2.2 Zeile 5).

- **Rumpf**
  Nach dieser Leerzeile folgen die Daten des Rumpfes (Listing 2.2 Zeilen 6 bis 10). Es gibt aber auch HTTP-Nachrichten, die ohne Rumpf gültig sind (siehe Listing 2.1). Beim so genannten „chunked encoding“ werden die Daten des Rumpfes in mehrere Blöcke aufgeteilt, die jeweils einzeln übertragen werden. Diese Blöcke sind dann durch eine Leerzeile voneinander getrennt, und die erste Zeile eines Blockes enthält jeweils dessen Länge in hexadezimalem Format.

```
1 Message = Request-Line | Status-Line
2           *(Message-Header CRLF)
3           CRLF
4           [ Message-Body ]
```

Listing 2.3: HTTP-Nachricht nach RFC 2616

Um HTTP in Verbindung mit anderen Protokollen oder Applikationen zu nutzen, die kein HTTP unterstützen, oder auch flexiblere Netzstrukturen zu ermöglichen, wurden verschiedene Konzepte entwickelt (nach [GOT02] und [RFC2616]):

- **Proxy**
  Ein Proxy besitzt mindestens zwei Netzwerkschnittstellen und benutzt für alle Schnittstellen das gleiche Protokoll. Der Proxy agiert an der Schnittstelle zum Client als Server und an der Schnittstelle zum Server als Client. Ein Proxy kann als Zwischenspeicher, Zugriffssteuerung und/oder Filter genutzt werden.

- **Gateway**
  Ein Gateway (auch Protokollumsetzer) besitzt ebenfalls mindestens zwei Netzwerkschnittstellen. Jedoch ist ein Gateway in der Lage ein Protokoll in ein anderes Protokoll zu „übersetzen“. Ein Gateway kann

zum Beispiel verwendet werden, um zwei unterschiedliche Netzwerke zu verbinden.

- **Tunnel**
  Tunnel ermöglichen es Applikationen, die HTTP nicht unterstützen, trotzdem über das Internet und vor allem durch Firewalls hindurch, zu kommunizieren. Dabei wird das jeweilige Applikationsprotokoll in HTTP „verpackt“. Der Tunnel beeinflusst die eigentliche Kommunikation nicht.

- **Relais**
  Ein Relais (Relay oder auch Repeater) ist eine sehr starke Vereinfachung eines Proxies. Allerdings „versteht“ ein Relais das jeweilige Netzwerkprotokoll nicht. Dies kann, je nach Protokoll, zu Problemen führen. Den eigentlichen Datenverkehr verändert ein Relais nicht.

Zur Verdeutlichung sind diese vier Konzepte noch einmal grafisch in Bild 2.9 dargestellt.

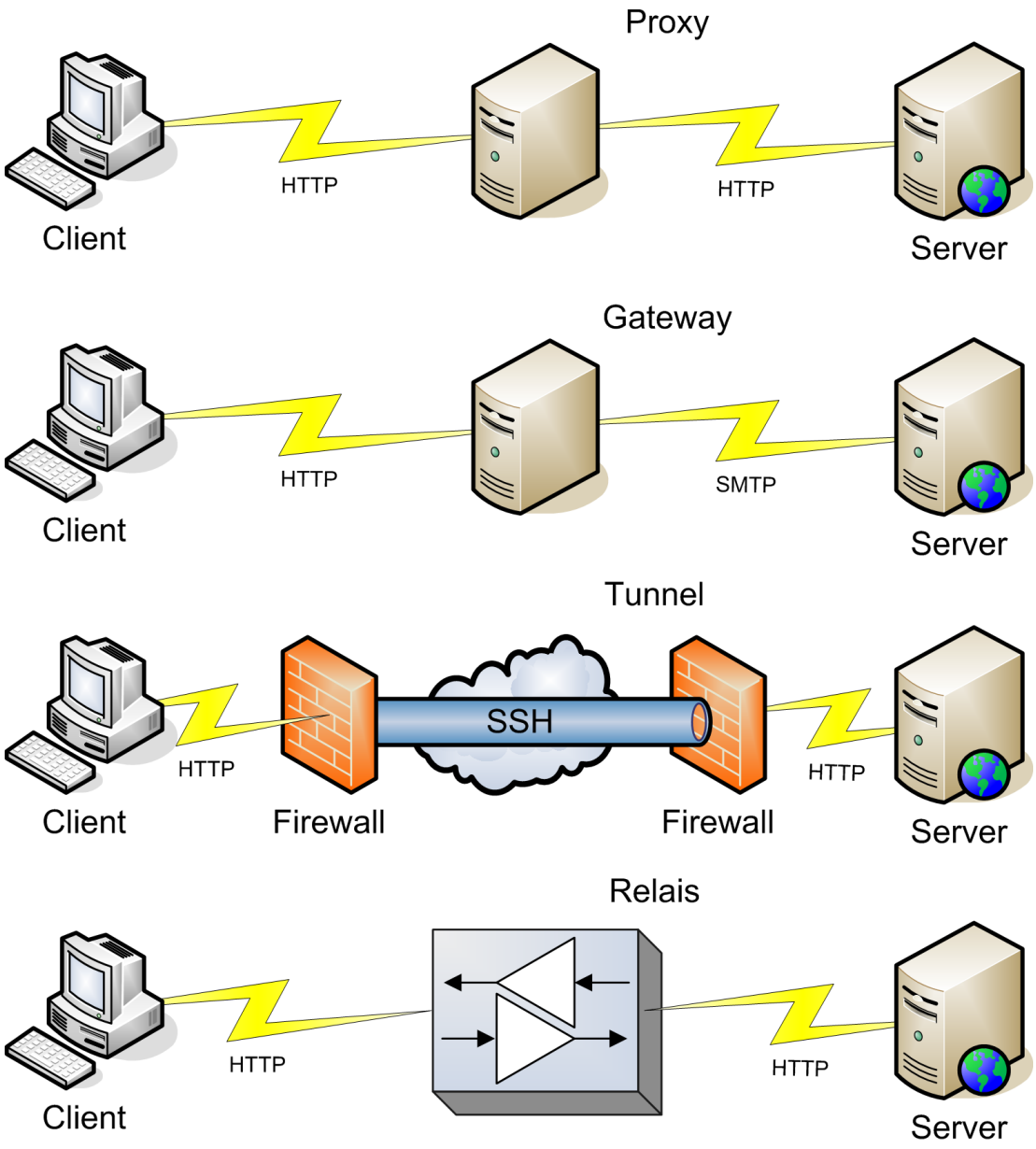


Bild 2.9: Grafische Darstellung von Proxy, Gateway, Tunnel und Relais

### 2.4.3 WebDAV

Web-based Distributed Authoring and Versioning (WebDAV) ist eine Erweiterung des Hypertext Transfer Protocols. Ein Webserver stellt, wie in Kapitel 2.4.2 bereits erläutert, Dateien zum Betrachten im Browser oder Herunterladen bereit. HTTP bietet nur wenige Befehle zum Manipulieren von diesen Inhalten an. Insbesondere existieren keine Befehle für die Ordnerverwaltung und es fehlt auch die Unterstützung für verteilte Zusammenarbeit. Die von HTTP in dieser Hinsicht angebotenen Befehle, wie etwa `DELETE` zum Löschen einer Datei, sind bei vielen Webservern zudem nicht implementiert. So ist das World Wide Web hauptsächlich ein Nur-Lese-Medium und zusätzliche Protokolle, zum Beispiel das File Transfer Protocol (FTP), müssen verwendet werden, um die Inhalte auf den Webservern zu verwalten [DUS04].

WebDAV schließt diese Lücke und bietet dem Benutzer umfangreiche Befehle zur Manipulation der Inhalte eines Webservers. Weiterhin erlaubt WebDAV verteiltes Editieren, verteiltes Dokumentenmanagement und Versionskontrolle. Initiiert wurde die Entwicklung 1996 durch Bildung einer Arbeitsgruppe aus Mitgliedern der Internet Engineering Task Force (IETF) und des World Wide Web Consortium (W3C) unter Führung von Jim Whitehead. Die Ergebnisse dieser Arbeitsgruppe wurden offiziell 1999 von der IETF bestätigt.

Insgesamt besteht WebDAV nicht aus einem einzelnen Standard, sondern es wurden mehrere Standards mit jeweils unterschiedlichen Schwerpunkten geschaffen. Die Basis bildet dabei RFC 4918, die aktuelle Version des Basis WebDAV-Protokolls. Im Folgenden sind die zu WebDAV gehörenden Standards mit einer kurzen Beschreibung aufgeführt [WDA08]:

- **RFC 3253 – Versioning Extensions** [RFC3253]
  Die Versioning Extensions (DeltaV) beinhaltet die Definitionen für das Versionsmanagement von WebDAV.

- **RFC 3648 – Ordered Collections** [RFC3648]
  Das WebDAV Ordered Collections Protocol beschreibt, wie Datensammlungen geordnet serverseitig abgelegt werden können.
- **RFC 3744 – Access Control Protocol** [RFC3744]
  Das Access Control Protocol (ACP) erlaubt es Clients, Zugriffsbeschränkungen des Servers zu lesen und zu bearbeiten.
- **RFC 4316 – Datatypes Properties** – *experimentell* [RFC4316]
  Dieser Standard beschreibt eine Menge von Datentypen, die WebDAV-Clients und Server verwenden können.
- **RFC 4331 – Quota** [RFC4331]
  Die Definition von WebDAV Quotas dient der serverseitigen Umsetzung von Beschränkungen des zur Verfügung stehenden Speicherplatzes.
- **RFC 4437 – Redirect** – *experimentell* [RFC4437]
  Der WebDAV-Redirect-Standard ermöglicht dem Client das Anlegen und Bearbeiten von Weiterleitungen.
- **RFC 4709 – Mount** – *informativ* [RFC4709]
  Diese Empfehlung soll Client- und Browser-Entwicklern helfen, ein einheitliches Verhalten im Umgang mit WebDAV-Ressourcen zu erzielen.
- **RFC 4791 – CalDAV** [RFC4791]
  Die WebDAV Calendaring Extension (CalDAV) beschreibt einen auf dem iCalendar-Format basierenden Zugriff auf Ressourcen von Kalendern und Terminplanern.
- **RFC 4918 – WebDAV** [RFC4918]
  Dies ist die aktuelle Version der HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). Dieser Standard dient als Basis für alle vorherig aufgezählten Definitionen.

- **DAV Searching and Locating – DASL** [DAS08]
  DASL ist eine Erweiterung des WebDAV-Protokolls, die das Durchsuchen und Auffinden von Ressourcen ermöglicht. Diese Erweiterung hat allerdings noch nicht den Reifegrad eines Standards erreicht.

Umgesetzt wird das WebDAV-Protokoll inzwischen von vielen Applikationen, allerdings selten in allen Teilen. So unterstützen populäre kommerzielle Programme, wie etwa Microsoft Windows und Office oder Adobe Photoshop, WebDAV. Weiterhin werden zahlreiche WebDAV-fähige Open Source Anwendungen angeboten, beispielsweise der KDE Desktop, DAVExplorer oder Subversion [WDP08].

Es folgt ein kleines Beispiel, an dem der Aufbau der WebDAV-Nachrichten erläutert wird. Der allgemeine Aufbau gleicht dem von HTTP-Nachrichten. Hinzugekommen sind zusätzliche Methoden und Stati. Der wichtigste Unterschied besteht allerdings im Aufbau des Nachrichtenrumpfes. Dieser nutzt bei WebDAV-Nachrichten immer das XML-Format zur Übertragung der Daten [RFC4918]. In Listing 2.4 ist beispielhaft eine Client-Anfrage dargestellt. Listing 2.5 zeigt die entsprechende Antwort des Servers. Der Rumpfteil der Antwort wurde dabei aus Gründen der Übersichtlichkeit gekürzt. Bei der dargestellten Anfrage wird die Methode `PROPFIND` genutzt (Zeile 1 Listing 2.4). Diese liefert die Eigenschaften einer Datei oder eines Verzeichnisses zurück. Wie aus Zeile 4 Listing 2.5 ersichtlich, nutzt die Antwort auf diese Anfrage „chunked encoding“. Einen Block einer so kodierten Antwort bilden die Zeilen 7 bis 29, auch als „chunk“ bezeichnet. Hierbei gibt Zeile 7 die Länge des Blockes in Byte an. Zeile 29 trennt diesen Block vom nächsten Block.

```
1  PROPFIND /slide/files/ HTTP/1.1
2  Host: localhost:9999
3  Content-type: text/xml
4  Content-length: 345
5
6  <?xml version="1.0"?>
7  <A:propfind xmlns:A="DAV:">
8      <A:prop>
9          <A:displayname/>
10         <A:resourcetype/>
```

```
        <A:getcontenttype/>
        <A:lockdiscovery/>
        <A:checked-in/>
        <A:checked-out/>
        <A:version-name/>
    </A:prop>
</A:propfind>
```

Listing 2.4: WebDAV-Client Anfrage

```
HTTP/1.1 207 Multi-Status
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 29 Apr 2008 11:45:14 GMT

61b
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:response xmlns:D="DAV:">
    <D:href>/slide/files</D:href>
    <D:propstat>
        <D:prop>
            <D:displayname>files</D:displayname>
            <D:resourcetype>
                <D:collection />
            </D:resourcetype>
            <D:lockdiscovery>
                <D:activelock>
                    <D:locktype><D:write /></D:locktype>
                    <D:lockscope><D:exclusive /></D:lockscope>
                    [...]
                </D:activelock>
            </D:lockdiscovery>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
</D:response>

[...]
```

Listing 2.5: WebDAV-Server Antwort (gekürzt)

## 2.5 Multithreading

Moderne Betriebssysteme erlauben mehreren Benutzern und/oder Applikationen gleichzeitig die vorhandenen Ressourcen, wie etwa Speicher, Prozessor und Festplatte, zu nutzen. Die Fähigkeit aktueller Rechner mehrere Aufgaben zu erledigen hat sich in den letzten Jahren durch die breite Einführung von Mehrkernprozessoren erheblich verbessert. Um dieses Potenzial optimal nutzen zu können, müssen bei dem Entwurf von Applikationen bestimmte Regeln berücksichtigt werden.

Im Allgemeinen wird eine Applikation in einem Betriebssystemkontext ausgeführt. Der Betriebssystemkontext besteht dabei aus Hauptspeicher, Rechenzeit und Zugriff auf andere Ressourcen. Diese werden jeweils vom Betriebssystem bereit gestellt und koordiniert. Eine solche Umgebung bezeichnet man auch als Prozess oder Task und sie steht einer Anwendung exklusiv zur Verfügung. Die Abarbeitung der Anwendung innerhalb einer solchen Umgebung wird auch als Ausführungsstrang oder Thread bezeichnet. Innerhalb eines Prozesses können mehrere Threads vorhanden sein. Diese werden je nach vorhandenen Ressourcen und Aufgaben abgearbeitet.

Um, wie oben bereits erwähnt, die Ressourcen optimal zu nutzen, ist es sinnvoll, verschiedene Aufgaben innerhalb einer Applikation auf verschiedene Ausführungsstränge zu verteilen. Zur Verdeutlichung ein Beispiel:

> Ein Programm mit grafischer Schnittstelle ermöglicht dem Benutzer Anfragen an entfernte Rechner über eine Netzwerkschnittstelle zu senden. Gleichzeitig soll der Benutzer aber auch in der Lage sein, die bisher empfangenen Daten auszuwerten.
>
> Eine Anwendung mit nur einem Ausführungsstrang würde beim Warten auf die Antwort eines anderen Rechners blockieren, das heißt aus der Sicht des Benutzers nicht mehr reagieren.
>
> Durch die Verwendung von eigenen Ausführungssträngen für die einzelnen Aufgaben, etwa der Verarbeitung der Benutzerin-

teraktion und des Netzwerkverkehrs, wird eine Parallelisierung und Entkopplung der Aufgaben erreicht.

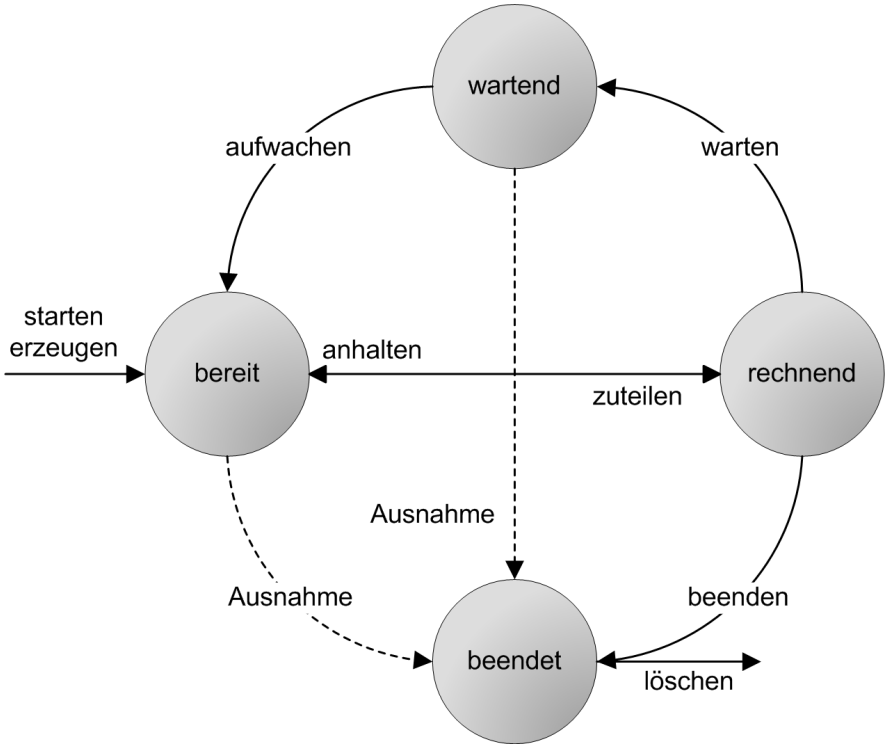


Bild 2.10: Threadzustände, vgl. [OEC01]

Anders als bei Prozessen steht der jeweilige Betriebssystemkontext allen Ausführungssträngen gemeinsam zur Verfügung. Dies hat sowohl Vor- als auch Nachteile. So ist die Erzeugung von Threads und der Wechsel zwischen Threads wesentlich günstiger als bei Prozessen, da das Betriebssystem bei einem Wechsel zwischen Threads keinen Kontextwechsel durchführen muss und bei der Erzeugung eines neuen Threads kein neuer Kontext erstellt werden muss [TAN02]. „Günstiger“ bedeutet in diesem Zusammenhang schneller und weniger ressourcenintensiv. Andererseits muss der Programmierer bei der Verwendung von mehreren Threads selbst dafür Sorge tragen, dass die verschiedenen Ausführungsstränge sich nicht gegenseitig blockieren oder gleichzeitig auf dieselbe Speicherstelle zugreifen. Auch der Datenaustausch zwischen verschiedenen Threads eines Prozesses muss koordiniert werden. Dies bezeichnet man auch als Synchronisation. Es exis-

tieren verschiedene Modelle (Semaphore, Mutexe, etc.) zur Lösung dieser Aufgabe (siehe [KRY02] und [OAW04]).

Zum besseren Verständnis des Ablaufs werden kurz die verschiedenen Threadzustände erläutert, siehe auch Bild 2.10:

- **bereit**
  Nach der Erzeugung und Start des Threads befindet sich dieser im Zustand „bereit“ und wartet auf die Zuteilung von Rechenzeit. Bekommt dieser Thread Rechenzeit zugeteilt, wechselt er in den Zustand „rechnend“.
- **rechnend**
  Wird ein Thread abgearbeitet, ist er im Zustand „rechnend“. Wird der Thread vom Betriebssystem angehalten, wechselt er wieder in den Zustand „bereit“. Muss der Thread auf eine Ressource oder Ergebnisse eines anderen Threads warten, geht er in den Zustand „wartend“ über. Ist eine Berechnung abgeschlossen und sind keine weiteren Aufgaben zu bearbeiten, wechselt der Thread in den Zustand „beendet“.
- **wartend**
  Im Zustand „wartend“ ist der Thread blockiert. Die Blockierung wird erst gelöst, wenn die entsprechenden Daten oder Ressourcen verfügbar sind.
- **beendet**
  Der Zustand „beendet“ markiert das Lebensende eines Threads. Dieser Zustand kann nicht mehr verlassen werden.

Falls in einem der Threadzustände ein Fehler auftreten sollte, wird der Thread in den Zustand „beendet“ versetzt.

## 2.6 Extensible Markup Language

Der globale Erfolg des Internets ist nur durch offene Standards wie TCP/IP, HTTP und HTML ermöglicht worden. Ebenfalls in dieser Reihe steht die „Extensible Markup Language“, kurz XML. Während die vorherig aufgezählten Protokolle für eine maschinenunabhängige Übertragung und Präsentation von Informationen sorgen, war zunächst kein Standard für die strukturierte Auszeichnung von Daten verfügbar. Diese Lücke füllt die von dem Word-Wide-Web-Konsortium (W3C) eingeführte Sprache XML [XML08]. Da das W3C sich bereits für die anderen Standards verantwortlich zeichnet, wurde auch dieser Standard durch das W3C etabliert.

Da die Extensible Markup Language keine Elemente für die Steuerung des Kontrollflusses eines Programms enthält, kann sie nicht den Programmiersprachen zugeordnet werden. Andererseits ist XML aber auch mehr als nur eine Auszeichnungssprache wie etwa die Hypertext Markup Language (HTML). So können in XML eigene Elemente definiert werden, die die einfache Abbildung auch von komplexen Strukturen erleichtern. XML ist also erweiterbar. Weiterhin beschreiben diese Elemente nicht die Darstellung der enthaltenen Daten, sondern deren Bedeutung. So lässt sich durch die Nutzung von XML auch eine Trennung von Struktur, Inhalt und Darstellung von Daten erzielen. Insgesamt lässt sich XML wohl am ehesten als Inhaltsbeschreibungssprache bezeichnen [VON07].

Die Extensible Markup Language wurde aus der Standard Generalized Markup Language (SGML) abgeleitet und ermöglicht sowohl Menschen als auch Maschinen den Inhalt eines XML-Dokumentes zu lesen und zu verarbeiten. Die Form eines XML-Dokumentes muss deswegen streng definiert sein. Die Struktur der Daten in einem XML-Dokument entspricht einer Baumstruktur. Als Knoten kommen Elemente und Attribute in Frage, und es existiert genau nur ein Wurzelelement. Wenn ein XML-Dokument diesen Definitionen entspricht, wird es als „wohlgeformt“ bezeichnet. Des Weiteren wird eine genaue Vorgabe durch die Verwendung von Strukturbeschreibungen erzielt. Die Strukturbeschreibungen sind wiederum XML-Dokumente einer speziellen Form. Zurzeit gibt es hiervon zwei

Ausprägungen, die Document Type Definition (DTD) und XML-Schema. Die Gültigkeit eines XML-Dokumentes kann mit einer solchen Strukturbeschreibung überprüft werden. Entspricht das Dokument der jeweiligen Strukturbeschreibung und erfüllt es die übrigen Regeln der XML-Definition, so wird das Dokument als „valide“ bezeichnet ([VON07] und [GOL99]).

Zur Verdeutlichung soll ein Adressbuch mit den Daten in einer XML-Datei und einer zugehörigen XML-Schema-Datei dienen. Der Inhalt der XML-Datei ist in Listing 2.6 abgebildet, die zugehörige Schema-Datei in Listing 2.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<adressbuch xmlns="http://www.adressbuch.de"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="simple.xsd">
  <person>
    <vorname>Rolf</vorname>
    <vorname>Herbert</vorname>
    <nachname>Müller</nachname>
    <ort>Köln</ort>
  </person>
  <person>
    <vorname>Horst</vorname>
    <nachname>Meier</nachname>
  </person>
</adressbuch>
```

Listing 2.6: Ein einfaches XML-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.adressbuch.de"
  xmlns="http://www.adressbuch.de"
  elementFormDefault="qualified">
  <xsd:element name="adressbuch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="person" minOccurs="0"
                                  maxOccurs="unbounded"/>
      </xsd:sequence>
```

```
12      </xsd:complexType>
13    </xsd:element>
14
15    <xsd:element name="person">
16      <xsd:complexType>
17        <xsd:sequence>
18            <xsd:element ref="vorname" minOccurs="0" maxOccurs="3"/>
19            <xsd:element ref="nachname" minOccurs="1" maxOccurs="1"/>
20            <xsd:element ref="ort" minOccurs="0" maxOccurs="1"/>
21        </xsd:sequence>
22      </xsd:complexType>
23    </xsd:element>
24
25    <xsd:element name="vorname" type="xsd:string"/>
26    <xsd:element name="nachname" type="xsd:string"/>
27    <xsd:element name="ort" type="xsd:string"/>
28  </xsd:schema>
```

Listing 2.7: Ein einfaches XML-Schema-Dokument

Der Aufbau des XML-Dokumentes ist dabei sehr einfach gehalten. Für das Wurzelelement `<addressbuch>` wird das zugehörige Schema (Zeile 2-4) angegeben. Danach folgen keine oder beliebig viele Elemente des Typs `<person>`. Dies wird durch die Zeilen 9 und 10 des Schemas festgelegt. Die Unterelemente von `<person>` wiederum sind im Schema in Zeile 15 bis 23 definiert, die Datentypen dieser Elemente in den Zeilen 25 bis 27. So kann das Element `<vorname>` minimal gar nicht und maximal dreimal vorhanden sein (Zeile 18). Weiterhin ist das Element `<vorname>` vom Datentyp „String“ (Zeile 25). Das XML-Dokument enthält zwei Personen. Die Person mit dem Nachnamen „Müller“ besitzt zwei Elemente des Typs `<vorname>` und die Person „Meier“ lediglich eins. Auf diese Weise kann die Datenstruktur eines XML-Dokumentes überprüft werden.

## 2.7 Java

Die Programmiersprache Java, ursprünglich unter dem Namen Oak (Object Application Kernel) entwickelt, wurde 1995 von Sun Microsystems veröffentlicht. Java ist aber mehr als eine Programmiersprache und wird von Sun Microsystems auch als Java-Technologie bezeichnet. Im Einzelnen besteht die Java-Technologie aus der Programmiersprache Java und verschiedenen Laufzeitumgebungen.

Eine in der Sprache Java geschriebene Anwendung wird normalerweise von einem Java-Compiler in Bytecode übersetzt. Dies ist ein binäres Zwischenformat, das unabhängig vom jeweiligen Betriebssystem ist und somit über Systemgrenzen hinaus genutzt werden kann. Der Bytecode wird dann auf dem Zielsystem durch die entsprechende Laufzeitumgebung, als Java Runtime Environment (JRE) bezeichnet, ausgeführt. Die Laufzeitumgebung besteht wiederum aus dem Interpreter, der Java Virtual Maschine (JVM), und den entsprechenden Bibliotheken.

Die Programmiersprache Java selber ist eine objektorientierte Sprache und unterstützt bereits in der Grundversion Multithreading, Netzwerkprogrammierung, grafische Oberflächen und Ausnahmen. Weiterhin werden eine automatische Speicherbereinigung (Garbage Collection) und Laufzeitumgebungen für die unterschiedlichsten Plattformen angeboten. Zusätzlich gehören noch umfangreiche Klassenbibliotheken zum Lieferumfang der verschiedenen Ausgaben.

### 2.7.1 Vererbung und Polymorphie

Eines der elementaren Konzepte der Objektorientierung ist die Vererbung. Im Folgenden wird dieses Konzept in Bezug auf die Programmiersprache Java betrachtet. Die Essenz dieses Konzeptes ist es, gemeinsame Eigenschaften von Objekten zu finden und diese in einer Klasse – Oberklasse oder Basisklasse – zusammen zu fassen. Eigenschaften, die nicht durch diese Abstraktion abgedeckt werden, werden in einer davon abgeleiteten,

spezialisierten Klasse – Unterklasse – realisiert. Die daraus resultierende Hierarchie wird auch als Klassenhierarchie bezeichnet.

Die Unterklasse erhält durch die Vererbung alle Attribute, Methoden und Beziehungen der Oberklasse. Die Eigenschaften der Oberklasse können somit auch ohne weiteren Aufwand von der Unterklasse genutzt werden. Neue Eigenschaften können in der Unterklasse problemlos hinzugefügt werden und bei Bedarf können in der Unterklasse auch Eigenschaften der Oberklasse neu definiert werden. Letzteres bezeichnet man auch als „überschreiben“ [JOB02].

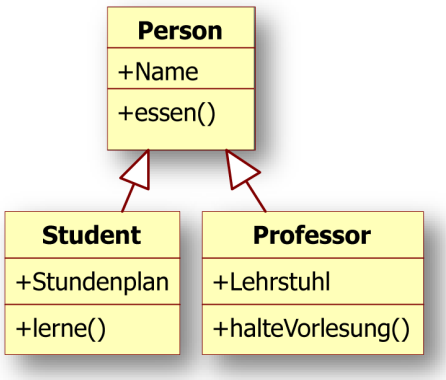


Bild 2.11: Ein Beispiel für Vererbung

Ein einfaches Beispiel ist in Bild 2.11 dargestellt. Die Klasse „Person“ besitzt das Attribut „Name“ und die Methode „essen()“. Da die Klassen „Student“ und „Professor“ von der Klasse „Person“ abgeleitet wurden, stehen diesen ebenfalls die Attribute und Methoden von „Person“ zur Verfügung. Zusätzlich realisieren die Unterklassen noch andere Attribute und Methoden.

Wenn eine Unterklasse von mehr als einer Oberklasse erben soll, was in Bild 2.12 dargestellt ist, bezeichnet man dies als Mehrfachvererbung. In dem dargestellten Beispiel erbt die Klasse „Hausboot“ sowohl von der Klasse „Haus“ als auch von der Klasse „Boot“. Dies kann man in der Programmiersprache Java nur durch die Nutzung von Schnittstellen (interfaces) umsetzten. Dabei werden bei der Definition der Schnittstelle die Methoden

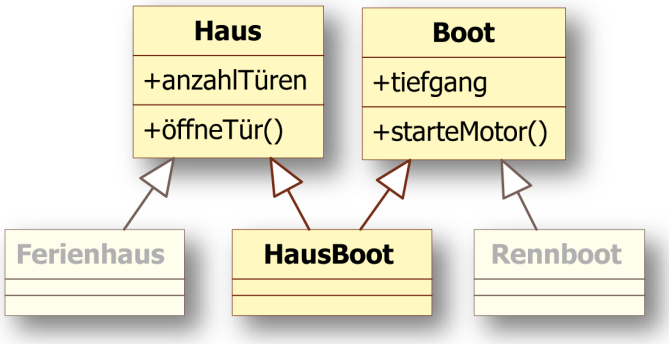


Bild 2.12: Ein Beispiel für Mehrfachvererbung

festgelegt, die eine Implementierung der Schnittstelle realisieren muss. Allerdings können so keine Attribute und Methodenrümpfe vererbt werden, sondern diese müssen von jeder Schnittstellen-Implementierung selbst verwirklicht werden. Dieses Konzept wird auch als Schnittstellenvererbung bezeichnet. Das in Bild 2.12 dargestellte Beispiel ist in Listing 2.8 in der Sprache Java umgesetzt[1].

```
1  // Klasse Boot
2  public class Boot {
3          public Object tiefgang;
4          public void starteMotor() {
5                  // tu etwas
6          }
7  }
8
9  // Schnittstelle Haus
10 public interface Haus {
11         void öffneTür();
12 }
13
14 // Klasse HausBoot
15 public class HausBoot extends Boot implements Haus {
```

[1]Anmerkung: Die Klassen- und Schnittstellendefinitionen müssen bei Java in separaten Dateien abgelegt werden. Aus Platzgründen wurden diese hier in einem Listing zusammen gefasst.

```
16          public Object anzahlTüren;
17          public void öffneTür() {
18                  // tu etwas anderes
19          }
20  }
```

Listing 2.8: Beispiel für Mehrfachvererbung in Java

Hierbei wurde der Typ „Boot“ als Klasse realisiert und der Typ „Haus“ als Schnittstelle. So kann die Klasse `HausBoot` nun sowohl von der Klasse `Boot` erben als auch die Methoden der Schnittstelle `Haus` realisieren.

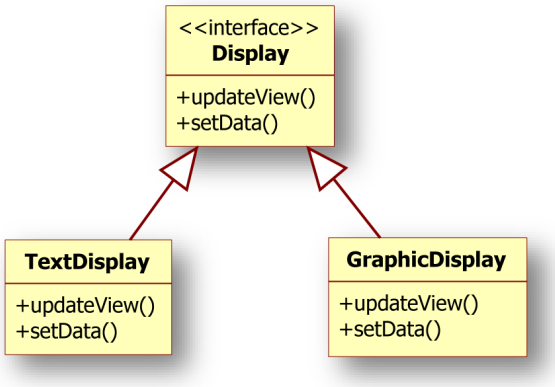


Bild 2.13: Ein Beispiel für Inklusionspolymorphie

Ein weiteres Kernkonzept der Objektorientierung ist die Polymorphie (Vielgestaltigkeit). Es existieren verschiedene Ausprägungen dieses Konzeptes. Inklusionspolymorphie bedeutet, dass eine Methode in der Vererbungshierarchie immer die gleiche Bezeichnung erhält, diese Methode jedoch von jeder Klasse auf unterschiedliche Weise realisiert wird. Daraus ist zu folgern, dass erst zur Laufzeit des Programms anhand des konkret vorhandenen Objektes entschieden werden kann, welche Implementierung der Methode auszuführen ist. Dies wird als „späte Bindung“ bezeichnet [JOB02].

Bild 2.13 verdeutlicht diese Zusammenhänge. Das Beispiel enthält die Schnittstelle „Display“ mit den zwei Methoden „updateView()“ und „set-

Data()“. Die Klassen „TextDisplay“ und „GraphicDisplay“ verwirklichen jeweils diese Schnittstelle. Abhängig von der konkreten Klasse sind die Methoden jedoch unterschiedlich implementiert. So erzeugt die Methode „updateView“ der Klasse „TextDisplay“ eine Textausgabe und die der Klasse „GraphicDisplay“ eine grafische Repräsentation der Daten.

Eine weitere Form der Polymorphie ist die „Überladung“. Bei dieser sind in einer Klasse zwei oder mehr Methoden mit der selben Bezeichnung, aber unterschiedlichen Signaturen vorhanden. Hier wird die auszuführende Methode durch den Typ der Parameter bestimmt.

### 2.7.2 Netzwerk-Kommunikation

Für die Netzwerk-Kommunikation stehen in Java verschiedene Typen von „Sockets“ zur Verfügung. Als „Socket“ bezeichnet man eine bidirektionale, vollduplexfähige Schnittstelle für die Interprozess-Kommunikation. Diese Schnittstelle ist ein Kommunikationsendpunkt, der durch eine Adresse, zum Beispiel eine Kombination von IP-Adresse und Port-Nummer (siehe Kapitel 2.4.1), eindeutig bestimmt ist. Weiterhin stellt diese Schnittstelle, in Java entspricht dies der Klasse *Socket*, dem Programmierer in der Basisvariante folgende Methoden bereit:

- Ein *Socket* kann eine Verbindung zu einem entfernten Rechner herstellen.
- Ein *Socket* kann Daten versenden.
- Ein *Socket* kann Daten empfangen.
- Ein *Socket* kann eine Verbindung trennen und schließen.

Dies vereinfacht die Nutzung von Netzwerkverbindungen für den Entwickler erheblich. Ein Socket stellt einen Ein- und Ausgabestrom bereit. Der Programmierer muss sich daher nicht um die korrekte Reihenfolge von Datenpaketen, Fehlererkennung, Fragmentierung der Daten und ähnliche Problematiken der Netzwerkebene kümmern [HAR05].

Ein Socket wird Hauptsächlich für Clients verwendet. Ein typischer Ablauf der Kommunikation einer Client-Anwendung besteht aus den folgenden Schritten:

1. Die Anwendung erstellt einen *Socket*.
2. Der Socket baut eine Verbindung zu dem entfernten Rechner auf.
3. Sobald eine Verbindung hergestellt ist, können der lokale und entfernte Rechner über den Ein- und Ausgabestrom des *Sockets* Daten austauschen.
4. Sind alle Daten übertragen, schließen beide Kommunikationspartner die Verbindung.

Da eine Client-Applikation ohne eine entsprechende Server-Applikation wenig sinnvoll ist, existiert speziell für Server-Anwendungen in Java die Klasse *ServerSocket*. Diese Klasse bietet als Ergänzung zu den Methoden der Basisvariante nachstehende Funktionen an:

- Ein *ServerSocket* kann an einen bestimmten Port gebunden werden.
- Ein *ServerSocket* kann auf eingehende Verbindungen warten.
- Ein *ServerSocket* kann Verbindungen von entfernten Rechnern auf dem gebundenen Port akzeptieren.

Der Ablauf einer Server-Applikation unter Verwendung der Klasse *ServerSocket* besteht aus den folgenden Schritten:

1. Die Anwendung erstellt einen *ServerSocket*. Dieser ist an einen bestimmten Port gebunden.
2. Der *ServerSocket* wartet auf eingehende Verbindungen.
3. Sobald eine eingehende Verbindung existiert, erzeugt der *ServerSocket* ein normales *Socket*-Objekt. Dieses ist dann für den eigentlichen Datenaustausch zwischen Client- und Server-Anwendung verantwortlich.
4. Daten zwischen Client- und Server-Anwendung können ausgetauscht werden.

5. Sind alle Daten übertragen, schließen beide Kommunikationspartner die Verbindung.
6. Die Anwendung kehrt zurück zu Schritt 2.

### 2.7.3 Swing

Swing ist ein Teil von insgesamt fünf Bibliotheken der Java Foundation Classes (JFC)[2] und bietet dem Entwickler die Möglichkeit, plattformunabhängige grafische Benutzerschnittstellen zu verwirklichen. Die Swing-Bibliothek wurde erstmals 1998 veröffentlicht und besteht aus einer Vielzahl von Klassen und Komponenten. Das Aussehen und Verhalten (Look & Feel) der grafischen Elemente lässt sich zur Laufzeit verändern und somit an die jeweilige Plattform anpassen.

Gegenüber dem Advanced Windowing Toolkit (AWT), der älteren Standard-Bibliothek für grafische Oberflächen, bietet Swing zahlreiche Erweiterungen und Verbesserungen [LOY03]:

- Anpassbares Aussehen und Verhalten (Look & Feel)
- Verbesserte und zusätzliche Komponenten, wie etwa Bäume, Tabellen und Unterfenster.
- Verwendung von Tastenkombinationen für Befehle von Komponenten.
- Mehrere Dokumente können innerhalb eines Fensters dargestellt werden (Multiple Document Interface – MDI).
- Leichtgewichtige Komponenten, das heißt die Komponenten sind selbst für die grafische Darstellung zuständig und nutzen hierzu Grafikprimitive der Java eigenen Grafik-Bibliothek.
- Die Realisierung der Komponenten erfolgt nach dem Modell-View-Controller-Muster (siehe Kapitel 2.3.4). Dies erleichtert die Wartung und Erweiterung der Komponenten.

[2] Die übrigen Bestandteile der JFC sind: Advanced Windowing Toolkit (AWT), Accessibility (Barrierefreiheit), 2D Application Programming Interface (API), Drag and Drop

Swing ist nicht thread-sicher und es müssen daher bei der Verwendung von Threads entsprechende Sicherungsmaßnahmen vorgenommen werden. Weiterhin existieren noch alternative Bibliotheken zur Erstellung von grafischen Oberflächen mit Java:

- **Advanced Windowing Toolkit (AWT)** [SDT08]
  AWT ist eine Bibliothek, die schwergewichtige und plattformunabhängige grafische Komponenten zur Verfügung stellt. Schwergewichtig bedeutet, dass die Bibliothek zum Erstellen und Zeichnen der grafischen Elemente direkt auf entsprechende Funktionen des jeweiligen Betriebssystems zurück greift. AWT ist Bestandteil der Standardausgabe von Java.

- **Standard Widget Toolkit (SWT)** [SWT08]
  Das Standard Widget Toolkit war ursprünglich eine Entwicklung der Firma IBM für die Entwicklungsumgebung „eclipse“. Diese Bibliothek stellt, ähnlich AWT, ebenfalls plattformunabhängige und schwergewichtige grafische Komponenten bereit. Da SWT kein Bestandteil der Standardausgabe von Java ist, muss der jeweilige plattformabhängige Teil der Bibliothek mitgeliefert werden.

- **Qt Jambi** [QTJ08]
  Qt Jambi baut auf der Bibliothek Qt der Firma Trolltech auf und stellt deren Funktionalität unter Java zur Verfügung. Qt ist wiederum eine in C++ geschriebene, plattformunabhängige Grafik-Bibliothek. Für Open-Source-Projekte muss die GPL verwendet werden, bei kommerziellen Projekten müssen entsprechende Lizenzen erworben werden. Diese Bibliothek befindet sich zurzeit in der Beta-Phase und eignet sich somit noch nicht für den produktiven Einsatz.

Swing zeichnet sich gegenüber diesen Produkten durch die gute Integration der Bibliothek in die Java Standardausgabe und die einfache Anpassbarkeit sowie die Reife des Produktes aus.

# 3 Konzept

In diesem Kapitel werden die Lösungsansätze und Ideen beschrieben, die zur Umsetzung des Prototypen, zukünftig als *DAVInspector* bezeichnet, genutzt werden. Zunächst erfolgt eine Präsentation der Ergebnisse der in diesem Rahmen durchgeführten Marktstudie und anschließend werden die Schlussfolgerungen daraus erläutert. Danach folgt ein Überblick über die Konzeption der Applikation. Die Ideen und Skizzen zu einzelnen Komponenten werden in eigenen Abschnitten ausführlicher erläutert. Abschließend erfolgt eine kompakte Zusammenstellung der Ergebnisse.

## 3.1 Marktanalyse

Nach einer Besprechung der Thematik zu Beginn des Projektes bestand die Aufgabe zunächst darin zu ermitteln, ob und in welcher Form bereits existierende Anwendungen zur Lösung der Aufgabenstellung herangezogen oder erweitert werden könnten. Die dabei erstellte und hier vorliegende Marktstudie (siehe Anhang D) gliedert sich in zwei Teile.

Im ersten Teil werden verfügbare Anwendungen zum Debuggen von HTTP- und WebDAV-Nachrichten sowie Applikationen zur Analyse von Netzwerkverkehr untersucht. Die dabei zugrunde liegenden Anforderungen sind:

- Die Lizenz der Software muss eine Erweiterung des Produktes zulassen und von Seiten des Auftraggebers (DLR) akzeptiert sein.
- Die Software soll plattformunabhängig sein.
- Auswertung und Manipulation des Datenverkehrs soll möglich sein.

- Die Software soll eine grafische Oberfläche zur Bedienung bieten.

Bewertet werden die betrachteten Werkzeuge nach den Kriterien:

- Funktionsumfang
- Einsatzzweck
- Erweiterbarkeit
- Aufwand für eine Erweiterung

Das Ergebnis dieser Studie weist ein Werkzeug als besonders geeignet für eine Erweiterung aus: die Software „NetTool“ des Autors Neil O'Tool. Diese Software hat alle technischen Anforderungen erfüllt, jedoch konnte die Lizenz nicht eindeutig geklärt werden. Weiterhin war zum Zeitpunkt der Marktstudie nur eine sehr alte Version des Quellcodes der Anwendung verfügbar. Da eine zeitnahe Lösung der Aufgabe angestrebt wurde, fiel die Entscheidung zugunsten einer vollständigen Eigenentwicklung des Werkzeugs. Einzelne Merkmale und Eigenschaften der Produkte „NetTool“, „Fiddler“ und „HTTP Probe“ sollten jedoch in der Eigenentwicklung besonders berücksichtigt werden. Eine genaue Erläuterung dazu findet sich in Kapitel 2.3 der Marktstudie. Eine aktuellere Version des Quellcodes von „NetTool“ wurde vom Autor der Software leider zu spät – die Arbeit an der Eigenentwicklung war schon fortgeschritten – zur Verfügung gestellt und hatte somit keinen Einfluss mehr auf diese Arbeit.

Aufgrund der im ersten Teil der Marktstudie getroffenen Entscheidung für eine Eigenentwicklung setzt sich der zweite Teil der Marktanalyse mit einer Auswahl an Programmiersprachen auseinander, die unter den gegebenen Randbedingungen als geeignet erscheinen. Auch hier werden die gewünschten Anforderungen definiert und Kriterien zur Bewertung festgelegt:

- Eine verbreitete Programmiersprache als optimale Voraussetzung für eine breite Nutzer- und Entwicklerbasis.
- Die Plattformunabhängigkeit soll einfach umzusetzen sein.

- Bibliotheken für Netzwerkprogrammierung, grafische Oberflächen, WebDAV- und XML-Verarbeitung sollen verfügbar sein.

Von der Seite des DLR wurden als Ausgangspunkt die Programmiersprachen Java, Python und C/C++ und deren Kombination vorgegeben. Die einzelnen Kriterien werden für jede Sprache im Rahmen der Marktstudie erörtert und schließlich die Entscheidung zugunsten der Programmiersprache Java begründet. Die wichtigsten Argumente sind an dieser Stelle kurz zusammengefasst: Java hat einen hohen Verbreitungsgrad bei Nutzern und Entwicklern. Dies erleichtert die Distribution der Anwendung und erhöht die Chancen, weitere Entwickler für das Projekt zu finden. Weiterhin ist im Lieferumfang der Java-Entwicklungsumgebung schon eine Vielfalt an Bibliotheken enthalten, und die Integration in verschiedene Plattformen gestaltet sich ebenfalls sehr einfach.

## 3.2 Gesamtkonzept

In diesem Abschnitt werden das Gesamtkonzept der Software und die Ideen und Vorgehensweisen beim Entwurf erörtert. Wie im vorherigen Kapitel beschrieben, existierte bereits zu Beginn der Arbeit eine Ideensammlung und ein grobes Anforderungsprofil an die gewünschte Software. Die Ergebnisse der Marktstudie konnten zu einer Präzisierung der Leistungsmerkmale genutzt werden, und verschiedene Eigenschaften der untersuchten Produkte dienten als Anregung bei der Entwicklung des DAVInspectors.

Zunächst ist aufgrund des Einsatzszenarios die Arbeitsweise auf Netzwerkebene des Produktes zu skizzieren. Das Hauptziel bei der Entwicklung der Software ist die Erstellung eines Werkzeugs, mit dem die Kommunikation zwischen einem WebDAV-Client und -Server ausgewertet und bei Bedarf manipuliert werden kann. Damit ergibt sich aus Perspektive der Netzwerkfunktionalität eine Anordnung des DAVInspectors zwischen WebDAV-Client und Server, wie sie in Bild 3.1 stark vereinfacht dargestellt ist. Die tatsächliche physikalische Verteilung der einzelnen Anwendungen auf

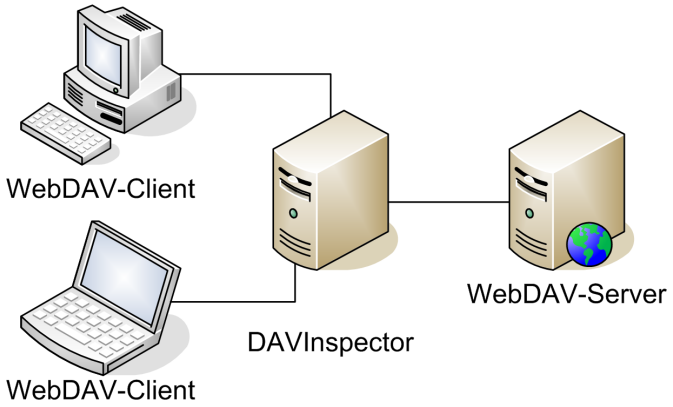


Bild 3.1: Einsatzszenario DAVInspector

Rechner kann dieser Darstellung entsprechen, es können aber auch alle Applikationen auf einem einzigen Rechner installiert sein.

Beispielhaft erfolgt eine schrittweise Beschreibung des Arbeitsablaufes. Dies soll zur Verdeutlichung der weiteren Entwurfsbeschreibung dienen. Eine von einem WebDAV-Client gestellte Anfrage wird an den DAVInspector geleitet. Dieser speichert die gestellte Anfrage ab, gibt dem Benutzer gegebenenfalls die Möglichkeit die Nachricht zu editieren und leitet sie anschließend an den WebDAV-Server weiter. Die Antwort wird in umgekehrter Reihenfolge abgearbeitet. Zusätzlich sind in Bild 3.2 die Anwendungsfälle der Software bezüglich der Auswertung des Datenverkehrs dargestellt. Exemplarisch illustriert die Grafik auch eine Möglichkeit in den Datenverkehr einzugreifen oder diesen für eine Vertiefung der Analyse zu speichern.

Um den Ablauf des Nachrichtenaustausches übersichtlich grafisch darzustellen, ist es erforderlich jede Nachricht eindeutig identifizieren zu können. Damit kann die Reihenfolge der versendeten Nachrichten nachvollzogen werden. Weiterhin ist es wichtig, in welcher Richtung eine Nachricht gesendet wurde, also ob die Nachricht vom Client zum Server (Request) oder vom Server zum Client (Response) gesendet wurde. Auch sollte ein möglichst einfacher Zugriff auf bereits abgearbeitete Nachrichten möglich sein. Zur besseren Visualisierung der Herkunft einer Nachricht ist eine

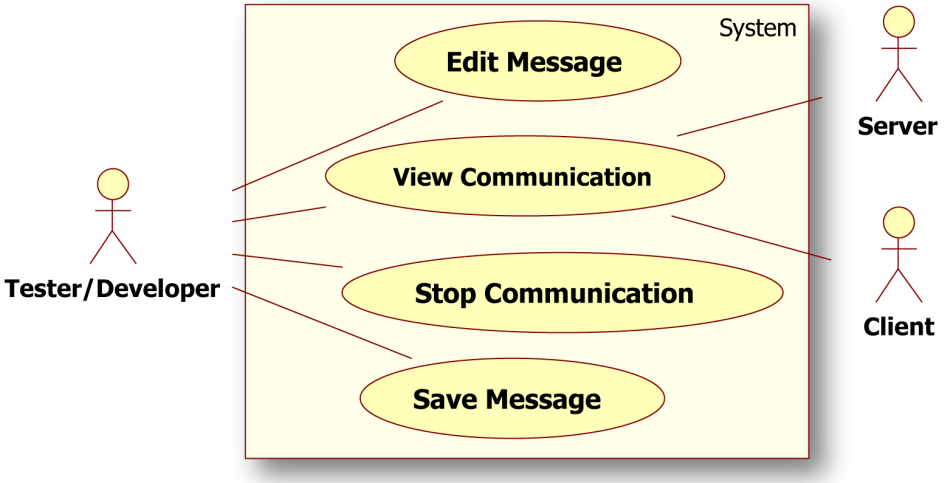


Bild 3.2: Anwendungsfalldiagramm DAVInspector

horizontal zweigeteilte Oberfläche, wie in Bild 3.4 dargestellt, erforderlich. So werden einer Seite die Client-Nachrichten und der anderen Seite die Server-Nachrichten zugeordnet und dadurch eine optische Trennung der beteiligten Kommunikationspartner erzielt. Für die visuelle Darstellung der Reihenfolge der versendeten Nachrichten ist die tabellarische Form besonders günstig, da hier weitere Informationen zu einer Nachricht für den Benutzer übersichtlich hinterlegt werden können.

Als Lizenz für die zu erstellende Software wurde die Apache Lizenz gewählt. Diese Entscheidung ist einerseits durch die Vorgabe der DLR-Juristen begründet. Andererseits ist durch diese Wahl die Kompatibilität zu anderen Open-Source-Projekten des DLR gegeben, welche ebenfalls die Apache Lizenz nutzen.

## 3.3 Gliederung des Entwurfs

Bereits während der Entwurfsphase der Software haben sich Aufgabengebiete und damit auch mögliche Subsysteme der Anwendung heraus kristallisiert. Dies ermöglichte eine sinnvolle Einteilung in Arbeitspakete und die Aufteilung des Konzeptes in einzelne Problemdomänen. Weiterhin wird die bei der Implementierung nötige Aufteilung in Module erleichtert. Folgend die einzelnen Arbeitsgebiete des Entwurfs:

- Grafische Benutzeroberfläche
- Netzwerkfunktionalität
- Nachrichtenprotokoll
- Plugin-Architektur
- Konzepte für Plugins
- Sonstige Konzepte

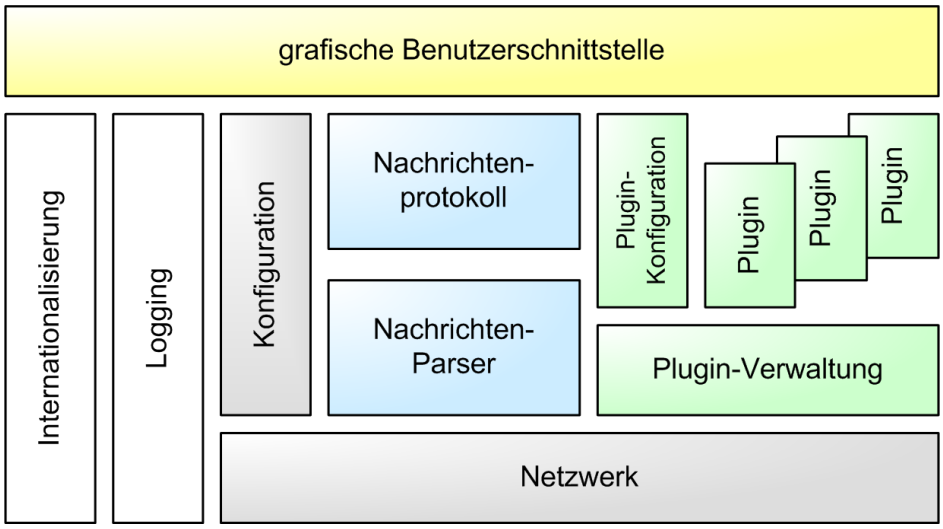


Bild 3.3: Funktionsblöcke der Software

Diese Aufteilung wird auch durch Bild 3.3 verdeutlicht. Die farbliche Gestaltung hebt die Zusammengehörigkeit der verschiedenen Teile zu einem

Themengebiet hervor. Der gelb gefärbte Block repräsentiert die Komponenten der Benutzeroberfläche. Die weiß gefärbten Blöcke sind allgemein zur Verfügung stehende Teile der Anwendung. Graue Blöcke enthalten die Netzwerkfunktionen. Das Plugin-System wird durch die Gruppe von grünen Blöcken visualisiert. Die blauen Blöcke fassen die Verarbeitung und Protokollierung der Nachrichten zusammen. Die Konzepte der einzelnen Arbeitsgebiete werden in den nachfolgenden Unterkapiteln diskutiert. Unter dem Punkt „Sonstige Konzepte“ sind Entwürfe zu verschiedenen, weniger umfangreichen Themen zusammengefasst.

### 3.3.1 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche stellt die Schnittstelle zum Benutzer dar. Sie soll eine intuitive und effiziente Bedienung der Anwendung erlauben. Zudem sollen die aufgezeichneten Daten leicht verständlich und übersichtlich präsentiert werden. Um diese Ziele zu erreichen, wurde unter anderem auf Erfahrungen aus der Marktstudie zurückgegriffen.

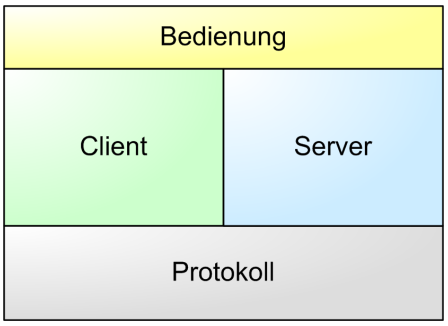


Bild 3.4: Anordnung der GUI-Komponenten

Aus persönlicher Erfahrung und den Erkenntnissen aus der Marktstudie erscheint dabei die in Kapitel 3.2 beschriebene Zweiteilung der Oberfläche in jeweils Client- und Serverseite als geeignete Darstellungsform. Ein Menü- und Navigationsbereich ist obligatorisch für grafische Anwendungen und befindet sich im oberen Bereich der Oberfläche. Das Protokoll des Netzwerkverkehrs ist am unteren Rand der Oberfläche angeordnet und vermittelt eine Übersicht über den Nachrichtenverlauf.

Die Anordnung der Elemente ist Bild 3.4 zu entnehmen. Bei der Verwendung von Knöpfen, Menüs und weiteren visuellen Komponenten werden ansonsten die allgemeinen Regeln des Oberflächenentwurfs berücksichtigt, siehe auch [LOY03].

### 3.3.2 Netzwerkfunktionalität

Der Entwurf der Netzwerkfunktionen bildet die Basis für alle anderen Komponenten. Gemäß Kapitel 2.4.2 existieren vier verschiedene Lösungsmöglichkeiten. Das angestrebte Anwendungsprofil (siehe auch Lastenheft Kapitel 3.1 Anhang C) verlangt nach einer Lösung, die den Datenverkehr unbeeinflusst weiterleiten kann. Trotzdem soll ein Eingriff in die Kommunikation und die Manipulation der Daten möglich sein. Als Konsequenz aus diesen Anforderungen ergibt sich das Relais als zielgerechte Lösung. Im Gegensatz zu einem Gateway oder Proxy muss ein Relais die gesendeten Daten nicht interpretieren können. Die Auswertung und Interpretation der Daten kann somit in höhere Schichten der Anwendung verlagert werden und erlaubt dadurch eine flexiblere Gestaltung. Bild 3.5 zeigt eine schematische Darstellung dieses Konzeptes.

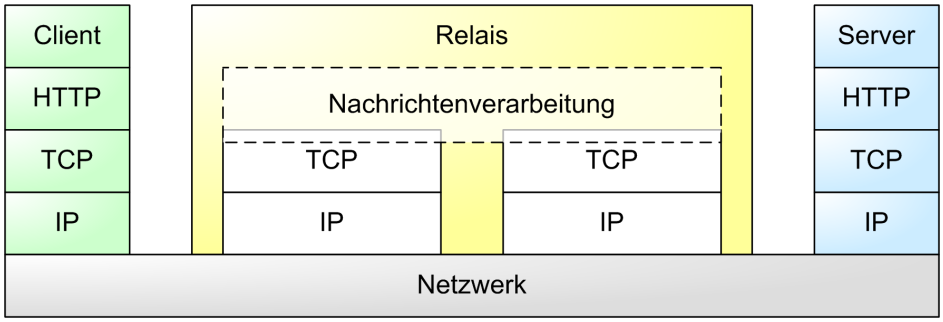


Bild 3.5: Vereinfachte Darstellung der Netzwerkfunktion

Hierbei befindet sich auf der linken Seite ein beliebiger HTTP-Client und auf der rechten Seite ein beliebiger HTTP-Server mit den entsprechenden Protokollstapeln. In der Mitte zwischen Client und Server befindet sich das Relais. Alle drei Anwendungen sind durch ein Netzwerk verbunden. Die

HTTP-Nachrichten werden mittels TCP/IP versendet. Die Schicht „Nachrichtenverarbeitung“ des Relais leitet Nachrichten zwischen Server und Client unverändert weiter. Gleichzeitig dient diese Schicht der Speicherung der Nachrichten und bietet eine definierte Schnittstelle zum Eingriff in den Datenverkehr. Die verschiedenen Ansatzpunkte der Schicht „Nachrichtenverarbeitung“ sind durch die gestrichelte und halbtransparente Darstellung visualisiert.

### 3.3.3 Nachrichtenhistorie

Die Nachrichtenhistorie speichert den Verlauf und die Daten des Netzwerkverkehrs. Dies erlaubt eine Visualisierung und gezielte Auswertung des Ablaufs und einzelner Nachrichten zu einem späteren Zeitpunkt. Jede Nachricht wird durch das Relais mit einer eindeutigen und fortlaufenden Kennung versehen. Dies ermöglicht einen gezielten Zugriff auf die abgespeicherten Nachrichten. Weiterhin ist dadurch auch die Reihenfolge der Nachrichten eindeutig definiert. Zusätzliche Eigenschaften einer Nachricht können nach Bedarf abgespeichert werden. Hier nun eine Zusammenstellung der wichtigsten Attribute einer Nachricht:

- ID – Eine eindeutige, fortlaufende Nummer zur Identifikation der Nachricht.
- Der Zeitstempel des Eintreffens der Nachricht.
- Die Senderichtung der Nachricht, also ob es sich um eine Anfrage (Request) oder Antwort (Response) handelt.
- Der eigentliche Inhalt der Nachricht. Dieser kann je nach Protokoll noch weiter aufgeschlüsselt werden. So können bei HTTP- und WebDAV-Nachrichten zum Beispiel noch Kopf- und Rumpfdaten separat gespeichert werden. Dies vereinfacht die spätere Verarbeitung der Nachrichten. Weitere Attribute einer Nachricht können abhängig vom verwendeten Protokoll ergänzt werden.
- Die Größe der Nachricht in Byte.

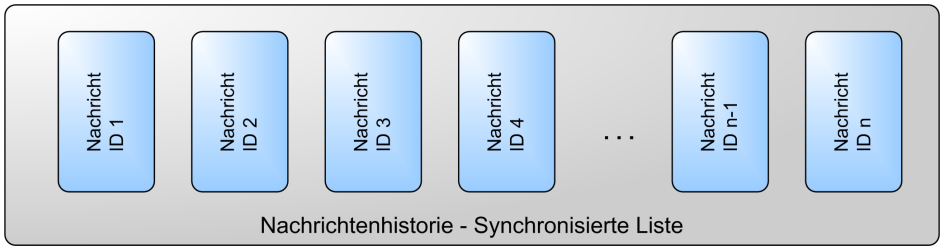


Bild 3.6: Struktur der Nachrichtenhistorie

Durch diese Attribute sind zunächst die Eigenschaften einer Nachricht bestimmt, die gespeichert werden. Diese müssen durch geeignete Methoden aus den Rohdaten des Datenverkehrs extrahiert werden. Um diese Daten dann geordnet abzulegen und bei Bedarf auch wieder zur Verfügung zu stellen, wird die Datenstruktur einer Liste verwendet. Bild 3.6 verdeutlicht diese Anordnung.

### 3.3.4 Plugin-Architektur

Eine Plugin-Architektur erlaubt die Erweiterung der Funktionalität einer Anwendung durch externe Entwickler. Ermöglicht wird dies durch eine exakt definierte Schnittstelle. Externe Entwickler müssen sich dann nicht mit den Interna der Anwendung auseinandersetzen, sondern es genügt die Kenntnis dieser Schnittstellendefinition. Damit minimiert sich der bei der Erstellung eines neuen Plugins erforderliche Einarbeitungsaufwand. Das elementare Entwurfsziel ist somit die Definition dieser Schnittstelle, die eine einfache Erweiterbarkeit der Auswertungsmöglichkeiten zur Laufzeit der Anwendung erlaubt. Dabei liegt der Schwerpunkt des Entwurfs auf einer einfachen Handhabung und Verwaltung der Plugins durch den Nutzer. Dieser kann in der Anwendung zur Laufzeit die gewünschten Plugins auswählen und damit ist ein unterbrechungsfreies Arbeiten und eine durchgängige Nachrichtenhistorie garantiert.

Die Distribution der Plugins erfolgt als Archiv auf Dateisystemebene. Die Schnittstellendefinition sollte einen Kompromiss zwischen einer möglichst

knapp gestalteten, statischen und einer umfangreichen, flexiblen Schnittstelle eingehen. Des Weiteren ergibt sich aus den teils unterschiedlichen Definitionen der Nachrichtentypen (siehe [RFC2616] und [RFC4918]) die Notwendigkeit, die Möglichkeit vorzusehen, die Plugins gezielt auch nur für einen Nachrichtentyp einzusetzen.

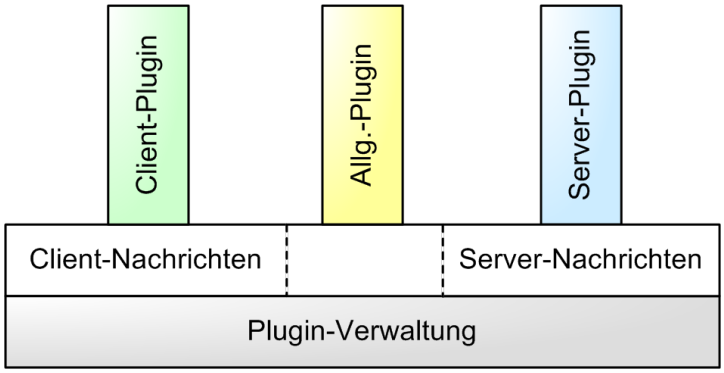


Bild 3.7: Vereinfachte Darstellung Plugin-Architektur

Bild 3.7 stellt schematisch die beteiligten Elemente der Plugin-Architektur dar. So bildet die Plugin-Verwaltung die Basis und damit das zentrale Element der Plugin-Architektur. Zu ihrem Aufgabenbereich gehört das Aktivieren und Deaktivieren von Plugins sowie die Verteilung der Daten. Die unterschiedlichen Plugins erhalten über verschiedene Schnittstellen die jeweils benötigten Daten.

### 3.3.5 Konzepte für Plugins

Um den eben vorgestellten Entwurf der Plugin-Architektur auch unter praxisnahen Bedingungen auf Tauglichkeit zu überprüfen, werden einige einfache Plugins konzipiert. Diese Plugins sind unterteilt in zwei verschiedene Typen:

- **Auswertende Plugins**
  Hierbei wird eine Kopie des Datenverkehrs an das jeweilige Plugin weitergeben, welches dann unabhängig von anderen Plugins die Daten auswerten, filtern oder anderweitig verarbeiten kann.

- **Manipulierende Plugins**
  Hierbei wird der eigentliche Datenstrom durch dieses Plugin geleitet und bietet somit die Möglichkeit die Daten vor dem Weitersenden zu verändern. Sind mehrere Plugins dieses Typs vorhanden, werden die Daten nacheinander von Plugin zu Plugin geleitet. Die Reihenfolge kann dabei nicht durch den Benutzer beeinflusst werden, sondern ergibt sich aus der Reihenfolge, in der die Plugins geladen werden.

Nach dieser allgemeinen Unterscheidung zwischen den verschiedenen Plugin-Typen werden nun die konkreten Entwürfe von Plugins vorgestellt.

**XML-Ansicht**
Ein Wunsch des Kunden ist eine formatierte Ausgabe des Rumpfes von WebDAV-Nachrichten. Da diese Rumpfdaten immer in XML-Form vorliegen, wird auch überprüft ob diese Daten im Sinne der XML-Definition valide sind. Die Überprüfung kann mit einem XML-Parser durchgeführt werden. Falls Fehler während des Parsens auftreten und die Daten somit nicht der Spezifikation entsprechen, werden entsprechende Fehlermeldungen ausgegeben.

**XML-Baumansicht**
Eine weitere Struktur, die sich auch besonders für die Darstellung von XML-Daten eignet, ist die Baumstruktur. Diese Perspektive erleichtert die Navigation innerhalb eines XML-Dokumentes und verbessert die Übersicht. Ebenso wie bei der XML-Ansicht können die Daten hierfür durch einen XML-Parser aufbereitet werden. Auch Fehler in der Struktur des XML-Dokumentes werden dem Nutzer entsprechend mitgeteilt.

**Kopfdatenansicht**
Einen wichtigen Teil von HTTP- und WebDAV-Nachrichten stellen die Kopfdaten dar. Wie in [RFC2616] und [RFC4918] spezifiziert, bestehen die Kopfdaten aus Schlüssel-Wert-Paaren[1]. Die Informationen aus diesem Teil einer Nachricht können somit gut in Tabellen abgelegt werden. Zusätzlich besteht noch die Möglichkeit den Inhalt dieser Tabellen zur Verbesserung der Übersicht in verschiedene Kategorien zu unterteilen.

[1] Eine vollständige Definition kann Kapitel 2.4.2 und Kapitel 2.4.3 entnommen werden.

**Aufzeichnungs-Plugin**
Das Aufzeichnungs-Plugin erlaubt die Speicherung und Protokollierung der Anfragen oder Antworten in einer durch den Benutzer spezifizierten Datei.

**Bearbeitungs-Plugin**
Bei dem Bearbeitungs-Plugin handelt sich um einen Demonstrator für ein manipulierendes Plugin. Dieses ermöglicht die Veränderung von Daten, bevor diese weiter versendet werden. Nach Beendigung der Bearbeitung der Nachricht muss der Benutzer dies quittieren, erst dann wird die Nachricht versendet.

Diese Liste von Konzepten für Plugins ist nicht abgeschlossen und kann durch die Anwender beliebig erweitert werden.

### 3.3.6 Sonstige Konzepte

In diesem Abschnitt werden die Entwürfe von weiteren Komponenten des DAVInspectors beschrieben.

**Konfiguration der Anwendung**
Um die Benutzung des DAVInspectors einfacher und komfortabler zu gestalten, soll die Konfiguration des DAVInspectors nicht bei jedem Programmstart erneut durchgeführt werden müssen. Deshalb ist vorgesehen die nötigen Daten in einer Datei zu speichern und bei Programmstart zu laden. Damit kann dann automatisch der zuletzt verwendete Konfigurationszustand des DAVInspectors wieder hergestellt werden. Die zu speichernden Daten umfassen zum Beispiel die Netzwerk-Konfiguration.

**Konfiguration der Plugins**
Die vorhandenen Plugins sollen durch den Benutzer zur Laufzeit aktiviert und deaktiviert werden können. Damit die jeweils gewählten Plugins beim nächsten Programmstart des DAVInspectors nicht erneut selektiert werden müssen, wird ebenfalls eine Speicherung der Plugin-Konfiguration in einer Datei vorgesehen. Diese Daten werden bei dem darauf folgenden Start geladen und die abgespeicherten Plugins automatisch aktiviert.

**Internationalisierung**
Um die Akzeptanz des DAVInspectors bei Entwicklern zu erhöhen, ist eine einfache Anpassung an die jeweilige Landessprache hilfreich. Die Vorbereitungen und Umsetzungen hierfür werden sowohl für das Hauptprogramm als auch für die Plugins vorgenommen. Dabei werden in einer separaten Datei für jede Landessprache die Übersetzungen für die verschiedenen Phrasen hinterlegt. Bei Programmstart wird dann durch Auswertung der jeweiligen Systemkonfiguration die entsprechende Sprachdatei geladen.

**Protokollierung**
Die Protokollierung (auch „Logging“) ermöglicht zum Zeitpunkt der Entwicklung und auch später im Betrieb die Gewinnung von detaillierten Informationen über den Ablauf des Programms. Diese Informationen können den Entwicklern bei der Behebung von Fehlern helfen und die Nachvollziehbarkeit von Fehlermeldungen erleichtern. An allen wichtigen Stellen des DAVInspectors werden daher Funktionen vorgesehen, die eine Ausgabe von relevanten Informationen erlauben. Diese werden in einer Datei gespeichert.

**Nachrichten-Erkennung**
Die Identifizierung einzelner Nachrichten ermöglicht die individualisierte Speicherung und die automatische Weiterleitung dieser Nachrichten. Dies erfordert eine Analyse der empfangenen Daten. Durch die enge Verwandtschaft von HTTP- und WebDAV-Nachrichten wird als Lösung ein einfacher gemeinsamer Parser entwickelt. Dieser Parser extrahiert dann aus den empfangenen Daten einzelne Nachrichten. Besondere Berücksichtigung findet dabei die blockweise Kodierung von HTTP- und WebDAV-Antworten (siehe „chunked encoding“ Kapitel 2.4.3) und die Auslegung für eine mögliche Erweiterung. Es können in zukünftigen Versionen des DAVInspectors dann weitere auf TCP aufbauende Protokolle verarbeitet werden, so zum Beispiel das File Transfer Protocol (FTP) [RFC959]. Die dafür nötigen Schnittstellen sind vorzusehen und erleichtern damit die Erstellung eines Parsers für das gewählte Protokoll.

## 3.4 Arbeitsumgebung

Die Beschreibung der Arbeitsumgebung umfasst alle Werkzeuge und Programme, die für die Realisierung der Applikation benötigt werden. Es folgt eine Auflistung und jeweils eine kurze Erläuterung dieser Hilfsmittel:

- **sourceforge.net**[2]
  Wie bereits in Kapitel 2.2 beschrieben, bietet SF für Open-Source-Projekte eine reichhaltige Auswahl an Werkzeugen. Nach einer Registrierung bei SF können neue Projekte angelegt und die Werkzeuge für diese je nach Bedarf konfiguriert werden. Die Konfiguration, die zur Umsetzung des DAVInspectors genutzt wird, besteht aus:
  - Subversion: Softwarekonfigurationsmanagement
  - Bug-Tracker: Erfassung von Fehlern, Änderungswünschen und Aufgabenverwaltung
  - Mailinglisten: Information über neue Versionen der Software und Änderungen des Quellcodes
  - Homepage: Dokumentation und Information für Nutzer und Entwickler
- **StarUML**[3]
  StarUML ist eine Open-Source-Applikation für die Erstellung von UML-Diagrammen. Die Diagramme entsprechen dem UML-Standard Version 2.0 und werden für Entwurf, Design und Dokumentation benutzt.
- **Java**[4]
  Für die Implementierung des DAVInspectors wird das Java Development Kit (JDK) in Version 1.5 genutzt. Hierbei sind nicht nur die Laufzeitumgebungen für die jeweilige Plattform mit den entsprechenden

[2] DAVInspector bei SF: http://sourceforge.net/projects/davinspector/
[3] StarUML: http://staruml.sourceforge.net/en/
[4] SUN Java 5 JDK: http://java.sun.com/javase/downloads/index_jdk5.jsp

Bibliotheken enthalten, sondern auch weitere Werkzeuge und Hilfsprogramme, die für die Erstellung von Java-Anwendungen notwendig sind.

- **eclipse**[5]
  Bei eclipse handelt es sich um ein Rahmenwerk einer integrierten Entwicklungsumgebung, die durch unterschiedlichste Plugins erweitert und angepasst werden kann. Für die Erstellung des DAVInspectors wird die Version „Europa“ verwendet. Die benutzten Erweiterungen werden innerhalb dieser Auflistung separat behandelt.

- **subclipse**[6]
  Subclipse ist ein Plugin für eclipse und ermöglicht die Nutzung des Softwarekonfigurationsmanagement-Werkzeugs Subversion. Das Konfigurationsmanagement ist somit vollständig in eclipse integriert.

- **Checkstyle**[7]
  Für die Überprüfung des Quelltextes nach formalen Gesichtspunkten wird das Checkstyle-Plugin für eclipse verwendet. Damit kann eine einheitliche Struktur des Programmcodes und der Kommentare erzielt werden. Die hierfür erforderlichen Vorgaben werden vom DLR zur Verfügung gestellt.

- **Ant**[8]
  Apache Ant ist ein in Java implementiertes Werkzeug, das eine Automatisierung des Erstellungsprozesses einer Software erlaubt. Java-Projekte werden von Ant durch spezielle Befehle unterstützt. Eine XML-Datei dient zur Konfiguration des Werkzeugs.

- **JUnit**[9]
  JUnit ist ein speziell für Java entwickeltes Werkzeug um Unit-Tests durchzuführen. Für das automatisierte Testen wird Version 4 von JUnit verwendet.

[5] eclipse: http://www.eclipse.org/
[6] subclipse: http://subclipse.tigris.org/
[7] Eclipse Checkstyle Plugin: http://eclipse-cs.sourceforge.net/
[8] Apache Ant: http://ant.apache.org/
[9] JUnit: http://www.junit.org/

Des Weiteren wurden die üblichen Hilfsmittel zur Erstellung von Dokumenten, Präsentationen und Grafiken verwendet.

## 3.5 Vorgehensweise

Da zu Beginn des Projektes nur die groben Spezifikationen zur Verfügung standen und eine exaktere Formulierung der Anforderungen nötig war, wurde als Vorgehensmodell die evolutionäre Entwicklung gewählt. Wie bereits in Kapitel 2.1 erläutert, existieren zwei Ausprägungen dieses Modells. Zunächst werden Wegwerf-Prototypen verwendet, um das generelle Konzept zu validieren und die Anforderungen zu verfeinern. Danach werden für dieses Projekt evolutionäre Prototypen genutzt.

Da die alleinige Anwendung eines Modells für alle Schritte der Softwareentwicklung jedoch nicht ausreichend ist, wird auch bei diesem Projekt eine Kombination von verschiedenen Modellen eingesetzt. Weiterhin ist es notwendig, dass bestimmte Phasen wiederholt werden können, um veränderte Systemanforderungen berücksichtigen zu können. Als Konsequenz wird der Ansatz der evolutionären Entwicklung mit den Phasen des Wasserfall-Modells und Iterationen zu einer inkrementellen Entwicklung verschmolzen. Als Vorteile ergeben sich:

- Die Entwicklung ist in einzelne Phasen unterteilt, ähnlich denen des Wasserfall-Modells. Dies bietet dem Entwickler den Vorteil abgegrenzter Abschnitte und vereinfacht somit die Einteilung in einzelne Arbeitspakete.

- Das Grundgerüst und bereits fertige Erweiterungen werden nur noch zwecks Fehlerbeseitigung angepasst. Diese Maßnahme vermeidet die bei der reinen evolutionären Entwicklung auftretenden ständigen Änderungen am System.

- Die Vorteile der evolutionären Entwicklung können genutzt werden. So sind weiterhin frühe Rückmeldungen der Benutzer möglich.

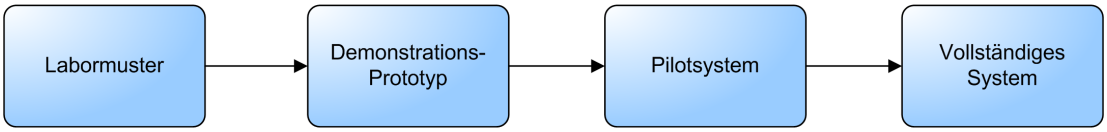


Bild 3.8: Abfolge der Prototypen

In Bild 3.8 ist die Abfolge der bei dieser Vorgehensweise verwendeten Prototypen dargestellt. Zunächst wird die grundsätzliche Funktion durch ein Labormuster nachgewiesen. Zur Verfeinerung der Spezifikation und insbesondere zur exakteren Formulierung der Anforderungen an die grafische Oberfläche wird ein Demonstrations-Prototyp verwendet. Nach Abschluss dieser Phase wird ein Pilotsystem entwickelt, welches schrittweise um weitere Funktionen ergänzt wird. Das Ergebnis dieses Vorgehens führt zu einem vollständigen System.

## 3.6 Zusammenfassung

Die Konzepte und Skizzen für die verschiedenen Komponenten des Projektes wurden im Rahmen dieses Kapitels vorgestellt und erläutert. Als Ansatzpunkt dienten zu Beginn die Ideensammlung und die Marktstudie. Die Anregungen und Lösungsansätze der dabei betrachteten Produkte wurden in die Entwürfe eingearbeitet. Um einen guten Überblick zu sichern und den Entwurf zu erleichtern, wurde das Gesamtsystem in mehrere Problemdomänen aufgeteilt und diese dann jeweils separat behandelt. Da weiterhin zu Beginn des Projektes nicht alle Anforderungen vollständig definiert waren, wurde ein entsprechendes Vorgehensmodell gewählt. Dies ermöglichte eine flexible Reaktion auf geänderte Anforderungen und neue Erkenntnisse, die im Laufe der Entwicklung des DAVInspectors zu Tage traten.

# 4 Design und Implementierung

Nachdem im vorherigen Kapitel die Entwürfe und Konzepte des DAVInspectors vorgestellt wurden, folgt nun eine Beschreibung der Realisierung. Dabei wird auf die aufgetretenen Probleme und deren Lösungen eingegangen.

Zunächst werden die Umgebungs- und Randbedingungen der Implementierung definiert. Danach werden die einzelnen Komponenten des Prototypen beschrieben und abschließend in einen gemeinsamen Zusammenhang gesetzt. Darauf erfolgt eine Darstellung der verwirklichten Plugins. Die durchgeführte Internationalisierung des DAVInspectors und die erstellte Dokumentation werden in gesonderten Abschnitten erläutert.

## 4.1 Konventionen

Eine einheitliche und durchgängige Vorgehensweise unterstützt die Einhaltung von Qualitätskriterien. Deshalb wurden für diese Arbeit Maßgaben und Randbedingungen, die bei der Entwicklung des Prototypen berücksichtigt werden müssen, festgelegt. Dazu gehört das Quellcode- und Release-Management, welches in Anhang A im Detail erläutert wird. Weiterhin werden die Vereinbarungen zum Aufbau des Quellcodes und die verschiedenen Arten der Kommentierung definiert. Eine ausführliche Diskussion dieser Thematik befindet sich in Anhang B.

## 4.2 Komponenten

In diesem Abschnitt wird die Realisierung der Software beschrieben. Um eine gute Architektur und damit Wiederverwendbarkeit der Software gewährleisten zu können, wird das Gesamtsystem partitioniert. Die Aufteilung erfolgt dabei gemäß dem Model-View-Controller-Muster.

Bild 4.2 zeigt den Entwurf der benötigten Klassen und Schnittstellen in einem UML-Diagramm. Hierbei ist zu beachten, dass, um eine bessere Übersicht zu bieten, dieses Diagramm nicht vollständig ist. Einzelne Klassen, zum Beispiel Hilfsklassen für die Darstellung, fehlen. Auch sind keine privaten Attribute oder Operationen aufgeführt. Die zentrale Komponente ist das `RelayModel`. Diese Klasse bildet zusammen mit den Klassen `MainController` und `MainView` das oben aufgeführte MVC-Muster ab. Weitere wichtige Klassen sind `ChannelThread` und `RelayThread`, die die Daten auf Netzwerkebene verarbeiten, die Klasse `PluginManager` zur Verwaltung und Nutzung der Plugins sowie die Klasse `MessageHistory`, die für die Speicherung des Verlaufs des Nachrichtenaustausches zuständig ist. Eine weitere Strukturierung der Software wird durch eine Aufteilung der Klassen des Projektes in verschiedene Pakete erreicht. Diese Maßnahme ermöglicht ein übersichtlicheres Design und bietet die Grundvoraussetzung für eine einfache Wiederverwendbarkeit von einzelnen Komponenten.

In Bild 4.1 ist die für den DAVInspector vorgesehene Partitionierung in verschiedene Pakete dargestellt. Im Paket „UI“ werden alle Klassen abgelegt, die für die grafische Benutzerschnittstelle benötigt werden. In dem Paket „Relay“ liegen die Klassen, die die Kernfunktionalität der Anwendung bilden. Von mehreren Teilen gemeinsam genutzte Klassen befinden sich im Paket „Common“. Klassen, die für die Konfiguration der Anwendung benötigt werden, befinden sich im Paket „Config“. Das Paket „Plugin“ enthält alle Pakete, die zur Verwaltung und Nutzung von Plugins benötigt werden. Im Paket „History“ befinden sich alle Klassen, die notwendig sind um die Verlaufsanzeige zur Visualisierung der Kommunikationssequenzen zu rea-

lisieren. Das Paket „plugins“ wiederum enthält konkrete Implementierungen von Plugins.

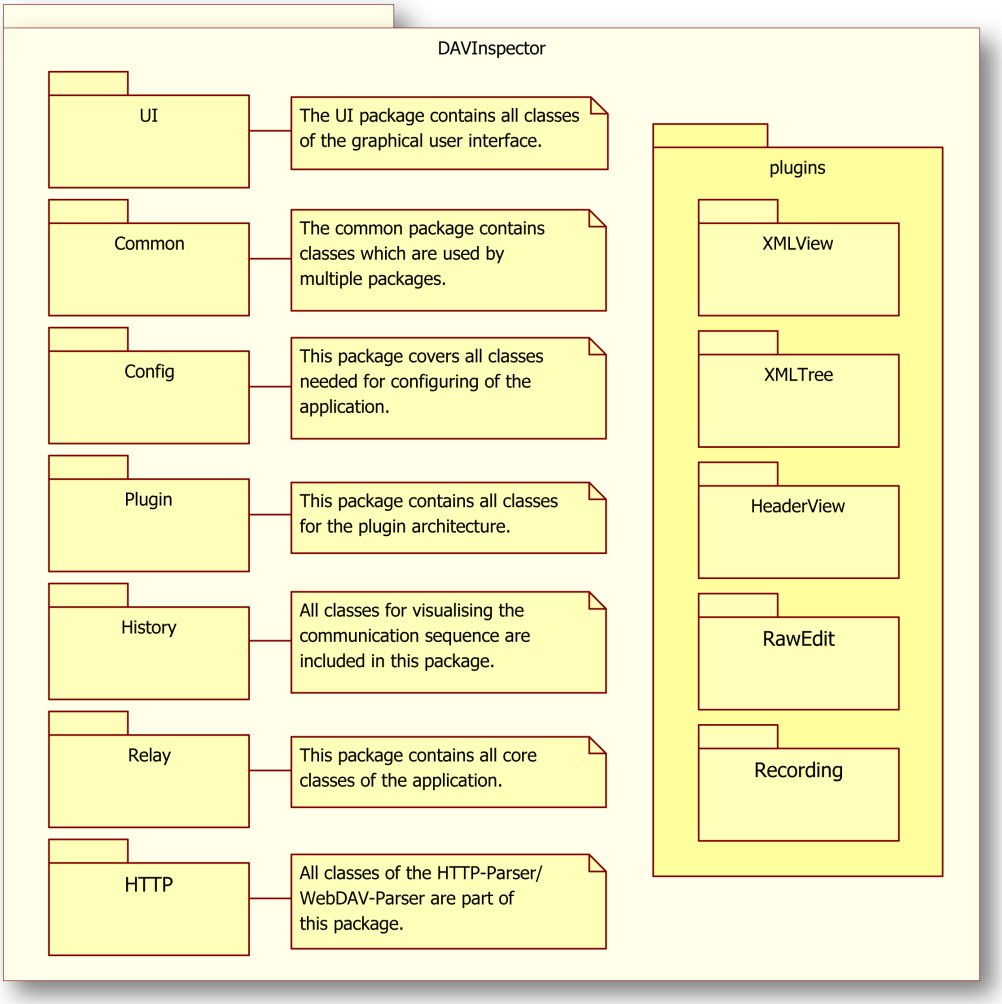


Bild 4.1: Paketstruktur DAVInspector

Exemplarisch sind hier die Plugins „XMLView“, „XMLTree“, „RawEdit“, „Recording“ und „HeaderView“ abgebildet. Diese befinden sich wiederum in jeweils eigenen Paketen. Das Paket „HTTP“ enthält die für den eingesetzten HTTP-/WebDAV-Parser benötigten Klassen und Ressourcen.

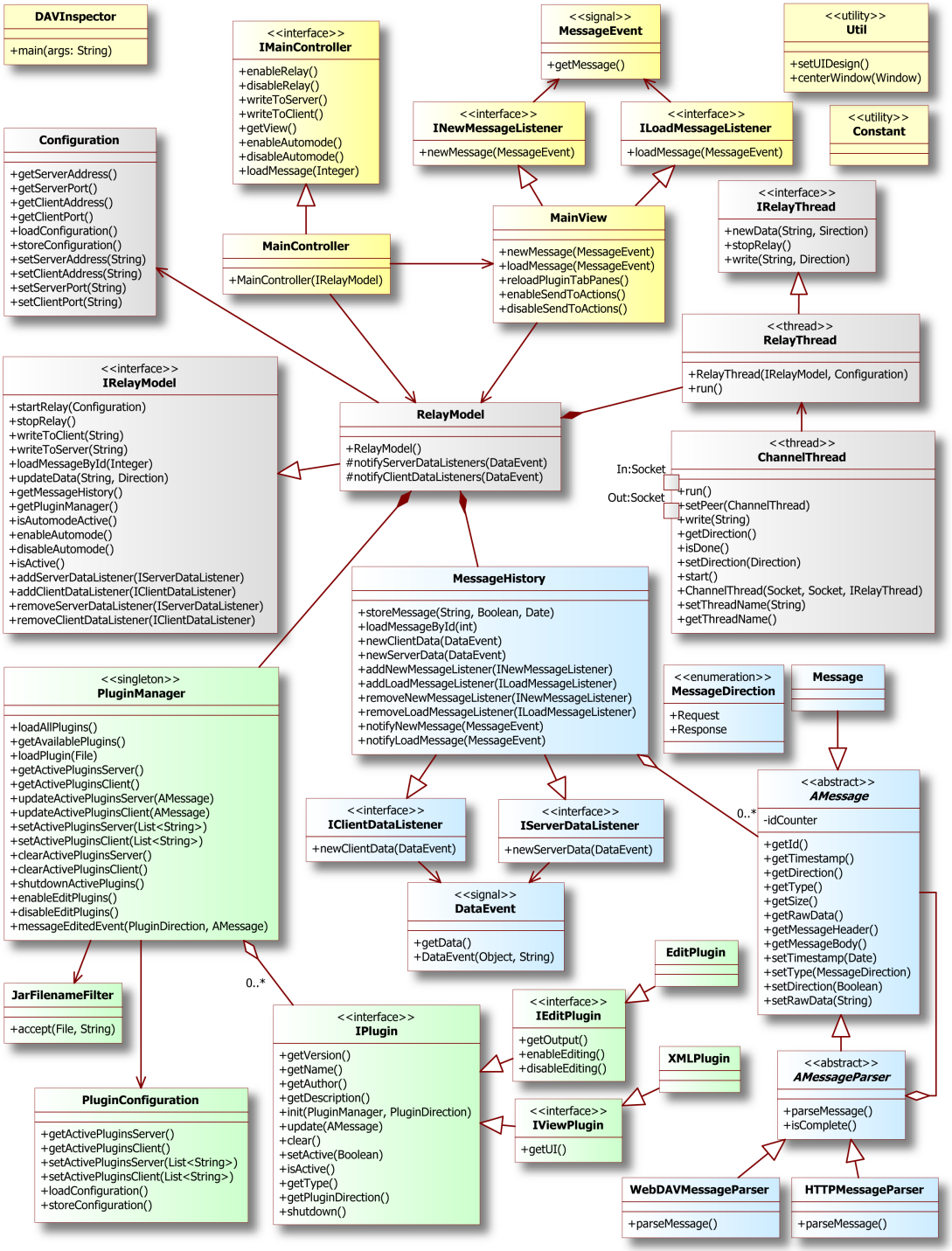


Bild 4.2: Klassendiagramm DAVInspector – Übersicht

### 4.2.1 Relais

Wie bereits beschrieben, ist eine wichtige Funktion der Anwendung die unveränderte Weiterleitung des Datenverkehrs. Diese Weiterleitung geschieht nach dem Prinzip eines Relais auf Schicht vier des OSI-Referenzmodells, also auf Ebene des TCP-Protokolls. Die einzelnen Datenpakete müssen hierbei durch manuellen Eingriff des Nutzers zum jeweiligen Kommunikationspartner weitergeleitet werden. Bild 4.3 zeigt das Klassendiagramm dieses zentralen Teils der Anwendung. Im Mittelpunkt steht die Klasse `RelayModel`, die die Schnittstelle für den Zugriff auf die Netzwerkfunktionen kapselt und die Konfiguration derselben vornimmt. Durch die Verwendung der Schnittstelle `IRelayModel` wird die einfache Erweiterbarkeit und Wartung des Designs sichergestellt. Die Klassen `PluginManager` und `MessageHistory` zeigen die Verbindung zu weiteren Teilsystemen der Anwendung auf.

Die genutzten Ports für die Kommunikation sind frei konfigurierbar. Dies ist eine Voraussetzung, um die Anwendung in verschiedenen Umgebungen nutzen zu können. Durch die flexible Anpassung der Kommunikationseinstellungen kann das Produkt in nahezu allen Netzwerken genutzt werden. Die Klasse `Configuration` speichert eine solche Konfiguration und wird von der Klasse `RelayModel` instanziiert. Die Java-Bibliothek `java.util.Properties` wird genutzt, um die Konfigurationsdaten der Software zu speichern und wieder einzulesen. Die Speicherung der Konfigurationsdaten erfolgt hierbei in einer einfachen Textdatei.

Die Weiterleitung des Datenverkehrs erfolgt durch die Klassen `RelayThread` und `ChannelThread`. Die Klasse `RelayThread` implementiert die Schnittstelle `IRelayThread` und enthält ein `ServerSocket`-Objekt, das auf eingehende Client-Verbindungen wartet. Sobald eine Verbindung aufgebaut ist, werden zwei `ChannelThread`-Objekte erzeugt. Diese sorgen für die Weiterleitung der Datenpakete. In Bild 4.4 ist das Sequenzdiagramm für den Ablauf bei der Durchleitung von Netzwerkverkehr zu sehen. Das Diagramm zeigt das gerade beschriebene Zusammenspiel der Klassen für eine Verbindung.

Zusätzlich wurde für eine komfortablere Bedienung des DAVInspectors eine Funktion realisiert, die automatisch erkannte HTTP- und WebDAV-Nachrichten weiterleiten kann. Dieser Betriebsmodus wird als „automatischer Modus“ bezeichnet. Der „normale“ Betriebsmodus, bei dem jede Nachricht durch den Eingriff des Benutzers weitergeleitet wird, wird als „manueller Modus“ bezeichnet.

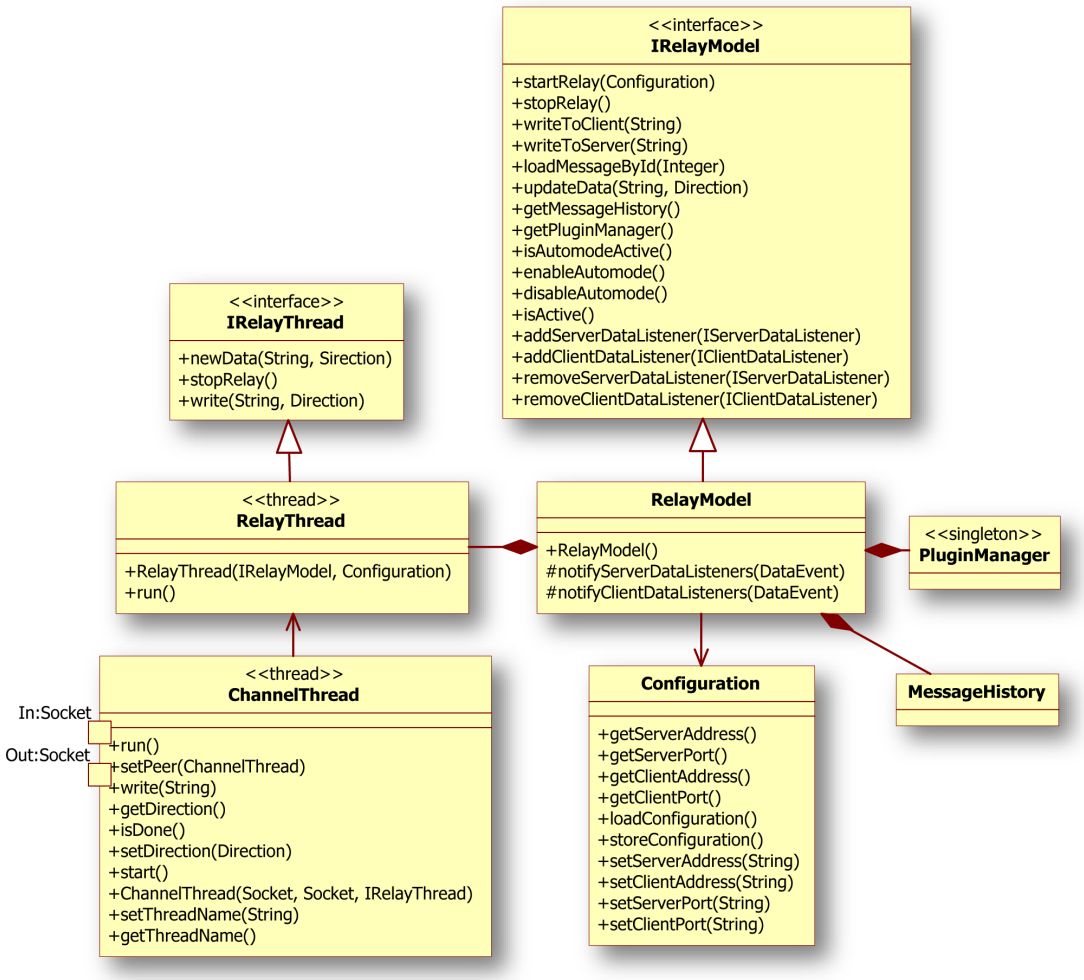


Bild 4.3: Klassendiagramm Relais

Für den automatischen Modus wird ein einfacher HTTP-/WebDAV-Parser benötigt, der Nachrichten erkennen kann. Somit kann dann eine Nachricht nach dem vollständigen Empfang automatisch weitergeleitet werden. Der

Parser wird im nächsten Kapitel beschrieben. Damit der Parser und andere Instanzen zur Weiterverarbeitung der Datenströme den Netzwerkverkehr erhalten, werden zwei Schnittstellen angeboten, `IClientDataListener` und `IServerDataListener`. Klassen, die diese Schnittstellen implementieren, werden bei Ankunft neuer Daten benachrichtigt und erhalten diese in Form von Objekten des Typs `DataEvent` (vgl. Kapitel 2.3).

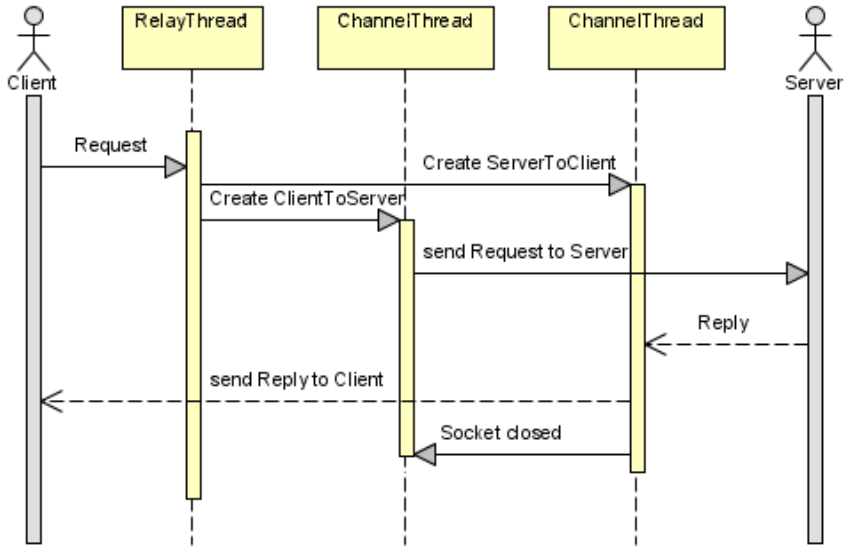


Bild 4.4: Sequenzdiagramm Nachrichtenweiterleitung

Ein weiteres wichtiges Ziel der Entwicklung ist die Möglichkeit den Datenverkehr zur Laufzeit verändern zu können. Die Abfolge der dabei nötigen Zustände ist in Bild 4.5 dargestellt. Die blau hinterlegten Elemente stellen die Verbindung vom Client zum Relais dar, die orange hinterlegten verdeutlichen die serverseitige Verbindung. Die grauen Zustände repräsentieren die Verbindungsverwaltung. Das Diagramm zeigt die Abfolge der Zustände für eine Verbindung. Der Datenfluss wird dabei durch die grauen Pfeile symbolisiert. Die beiden gelb hinterlegten Zustände markieren die Stellen, an denen in den Datenverkehr eingegriffen wird. Hier befinden sich die Schnittstellen zu den verschiedenen Anzeige- und Manipulationsmöglichkeiten, die in der Software vorhanden sind.

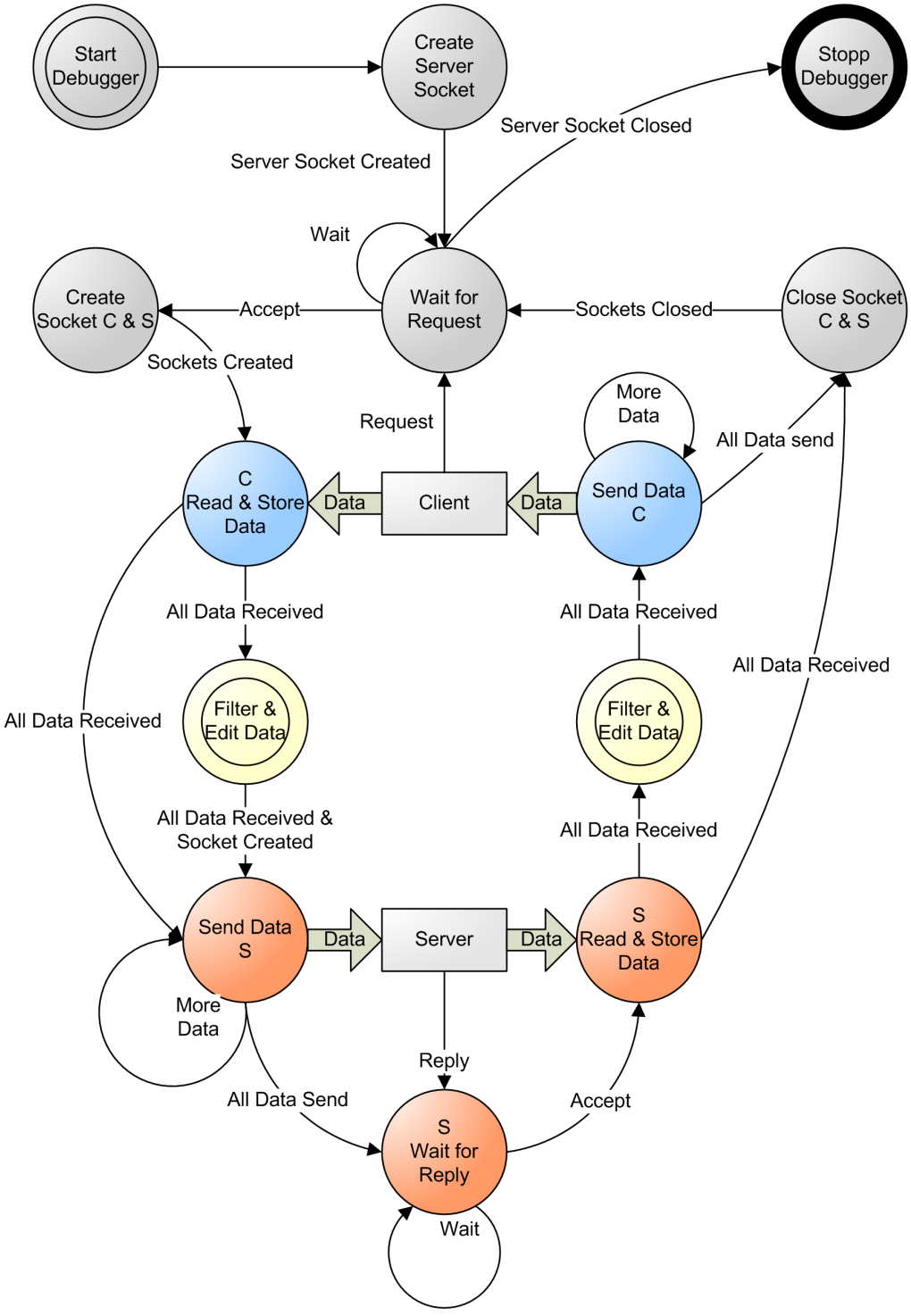


Bild 4.5: Zustandsdiagramm Relais

### 4.2.2 Nachrichtenhistorie

Um die gewünschte Nachrichtenhistorie realisieren zu können, sind weitere Funktionen nötig. Zunächst ist es erforderlich, dass überhaupt Nachrichten gespeichert und identifiziert werden können. Die Identifikation erfolgt durch einen entsprechenden Parser. Dieser extrahiert die Nachrichten aus dem Datenstrom. Dann wird bei jeder neuen Nachricht ein Nachrichtenobjekt erzeugt und dieses in einer Datenstruktur abgelegt. Die Nachrichtenobjekte selbst bieten Zugriff auf die wichtigsten Parameter einer Nachricht, wie etwa Nachrichtentyp, Größe und Ähnliches. Weiterhin ist es notwendig für eine spätere Verarbeitung und die Aktualisierung der Ansicht auf einzelne Nachrichten gezielt zugreifen zu können. Um dieser Forderung gerecht zu werden, erhält jede Nachricht eine eindeutige Nummer. Auch sind Funktionen vorhanden, die es erlauben, Client-Nachrichten und/oder Server-Nachrichten in eine Protokolldatei zu exportieren.

Bild 4.6 zeigt das Klassendiagramm aller für diesen Teil benötigten Klassen. Die zentrale Rolle übernimmt die Klasse `MessageHistory`. Diese Klasse speichert die Nachrichten in einer Liste und kapselt den Zugriff auf Nachrichten zur späteren Auswertung. Weiterhin löst diese Klasse Ereignisse des Typs `MessageEvent` aus. Durch Implementierung der Schnittstellen `ILoadMessageListener` und/oder `INewMessageListener` können sich andere Klassen aktualisieren lassen (vgl. Beobachter-Muster Kapitel 2.3). Die Klasse `MessageHistory` selbst implementiert die Schnittstellen `IClientDataListener` und `IServerDataListener`, um die vom `RelayModel` zur Verfügung gestellten Daten aus den jeweiligen Verbindungen des Typs `DataEvent` zu erhalten.

Die Klassen `Message` und `HTTPMessageParser`, die abstrakten Klassen `AMessage` und `AMessageParser` sowie die Aufzählungsklasse `MessageDirection` sind für die Verarbeitung und Extraktion der Nachrichten aus dem Datenstrom des Relais zuständig. Die Realisierung dieser Klassen setzt zudem das Beobachter-Muster um, damit gegebenenfalls neue Parser für weitere Protokolle ergänzt werden können. In Bild

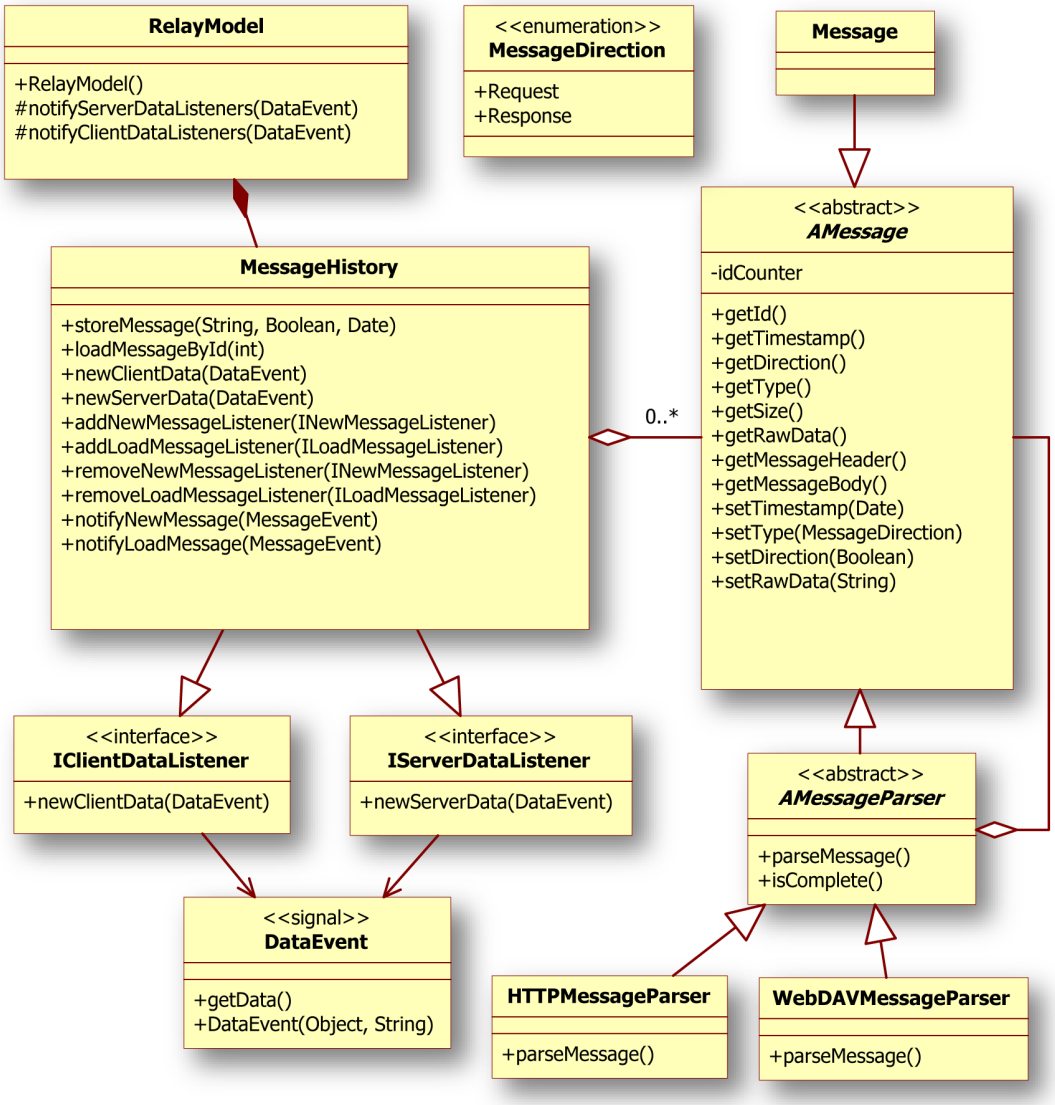


Bild 4.6: Klassendiagramm Nachrichtenhistorie

4.7 und Bild 4.8 ist der Nachrichten-Parser im Detail am Beispiel des HTTP/WebDAV-Parsers visualisiert.

Die für die übersichtliche Darstellung des Datenverkehrs erforderliche getrennte Anzeige von Anfrage („Request“ – Client-Seite) und Antwort („Reply“ – Server-Seite) sowie die Einteilung einer Nachricht in Kopf- und Rumpfdaten wird ebenfalls von dem Nachrichten-Parser durchgeführt. Die Unterscheidung erfolgt im Falle von HTTP- und WebDAV-Nachrichten anhand der Definition der Nachrichtentypen nach RFC 2616 [RFC2616] und RFC 4918 [RFC4918] (vgl. Kapitel 2.4.2 und 2.4.3).

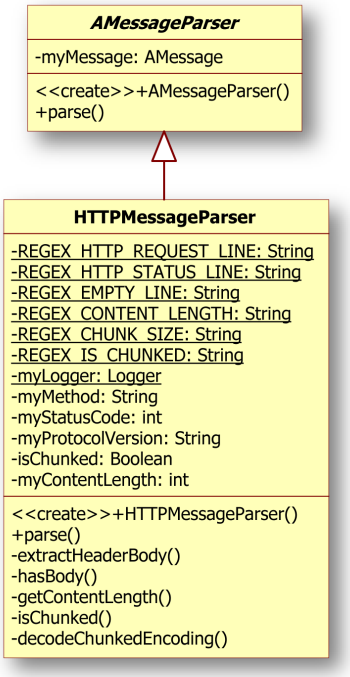


Bild 4.7: Klassendiagramm Nachrichten-Parser im Detail

Da es sich um einen simplen Nachrichten-Parser handelt, wurde für die Umsetzung der geforderten Funktionen auf reguläre Ausdrücke zurückgegriffen. In Bild 4.8 ist der vereinfachte Ablauf eines Analysevorgangs dargestellt. Zunächst wird die Richtung der Nachricht ausgewertet. Dies entscheidet, ob die Daten mit den regulären Ausdrücken für eine Anfrage oder Antwort ausgewertet werden. Als Besonderheit muss bei Antwortnachrich-

ten der Nachrichtenrumpf auf eine blockweise Übertragung („chunked“) hin untersucht werden und die Daten müssen gegebenenfalls extrahiert werden. Danach wird, sofern vorhanden, die Länge der Daten im Rumpfteil der Nachricht bestimmt. Weitere Eigenschaften der Nachricht und die Daten des Nachrichtenrumpfes und Nachrichtenkopfes werden dann in einem Nachrichten-Objekt abgelegt.

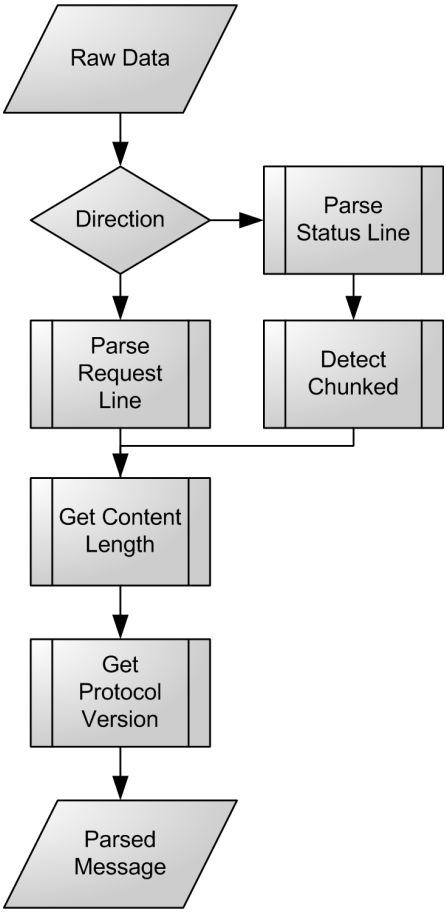


Bild 4.8: Struktur des Nachrichten-Parsers (vereinfacht)

Die vom DAVInspector erstellte Nachrichtenhistorie kann in eine Datei exportiert werden. Diese Datei kann später zum Erstellen eines Makros[1] dienen oder gegebenenfalls zu weiteren Analysen durch andere Werkzeuge herangezogen werden. Die Nachrichten werden hierbei in der Reihenfolge

[1]Hier: Eine Sequenz von einer oder mehreren HTTP/WebDAV-Nachrichten

ihres Eintreffens in einer Datei gespeichert. Der Dateiname der Exportdatei setzt sich aus dem Datum und der Uhrzeit zusammen. Alternativ kann der Nutzer den Namen frei wählen. Es wurde ein Dialog realisiert, der dem Benutzer erlaubt den Nachrichtentyp und den gewünschten Dateinamen für den Export der Daten auszuwählen. In Bild 4.9 ist die Ausführung dieses Dialoges dargestellt.

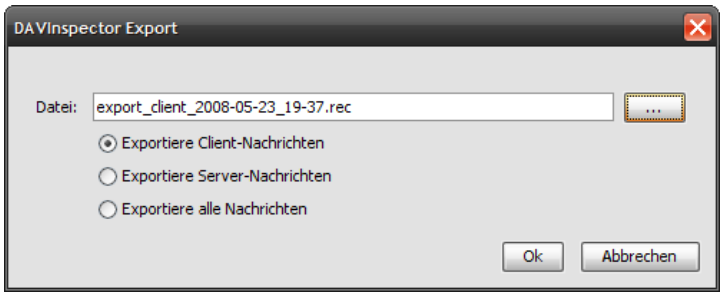


Bild 4.9: Export-Dialog

Bei der Durchführung des Exports werden abhängig von der vom Benutzer getätigten Auswahl alle Client-Nachrichten, alle Server-Nachrichten oder alle Typen der bisher aufgezeichneten Nachrichten in eine Datei geschrieben. Die Nachrichten werden durch drei Leerzeilen voneinander getrennt.

### 4.2.3 Plugin-Verwaltung

Die Verwendung einer Plugin-Architektur ermöglicht die Erweiterung der Funktionalität der Software. Entwicklern wird damit die Realisierung von weiteren Kommunikationsanalysefunktionen erleichtert. Die Plugins sind unterteilt in zwei verschiedene Typen:

- Auswertende Plugins: Hierbei wird eine Kopie des Datenverkehrs an das jeweilige Plugin weitergegeben, welches dann unabhängig von anderen Plugins die Daten auswerten, filtern oder anderweitig verarbeiten kann.

- Manipulierende Plugins: Hierbei wird der eigentliche Datenstrom durch dieses Plugin geleitet und bietet somit die Möglichkeit die Daten vor dem Weitersenden zu verändern. Sind mehrere Plugins dieses Typs vorhanden, war ursprünglich vorgesehen, die Daten nacheinander von Plugin zu Plugin zu leiten. Die Reihenfolge sollte dabei nicht durch den Benutzer beeinflusst werden können. Dies konnte im Zuge dieser Arbeit jedoch nicht verwirklicht werden, da kein praktikabler Ansatz zur Einhaltung der Reihenfolge in der grafischen Oberfläche gefunden werden konnte. Stattdessen wurde eine Lösung realisiert, bei der der Benutzer jederzeit editieren kann und durch ein grafisches Element in der Oberfläche die Beendigung des Editiervorgangs mitteilen kann. Daraufhin werden alle betroffenen Plugins mit den editierten Daten aktualisiert.

Das Klassendiagramm dieser Architektur ist in Bild 4.10 dargestellt. Das zentrale Element der Plugin-Verwaltung ist die Klasse `PluginManager`. Diese Klasse ist die Schnittstelle zum Relais und ist neben der Verwaltung der Plugins auch für deren Aktualisierung und Steuerung verantwortlich. Die Klasse `PluginManager` setzt das Einzelstück-Muster um, damit garantiert ist, dass nur eine Instanz dieser Klasse zur Laufzeit existiert. Dies ist für die korrekte Funktionsweise der Plugin-Architektur erforderlich. Die Schnittstellen `IPlugin`, `IViewPlugin` und `IEditPlugin` legen die Methoden fest, die Entwickler von Plugins implementieren müssen.

Beim Start des DAVInspectors wird das Verzeichnis „plugin“ nach Archivdateien mit der Dateiendung „.jar“ durchsucht. Die Filterung der Dateinamen wird von der Klasse `JarFilenameFilter` realisiert. Jedes gefundene Archiv wird einem `URLClassLoader` übergeben. Dieser versucht eine Instanz einer Klasse zu erzeugen, die denselben Namen trägt wie das „.jar“-Archiv. Konnte ein entsprechendes Objekt erzeugt werden, dann werden der Name und die Eigenschaften des Plugins in eine Liste von verfügbaren Plugins eingetragen. Die Bezeichnung des Plugins wird hierbei als Schlüssel verwendet.

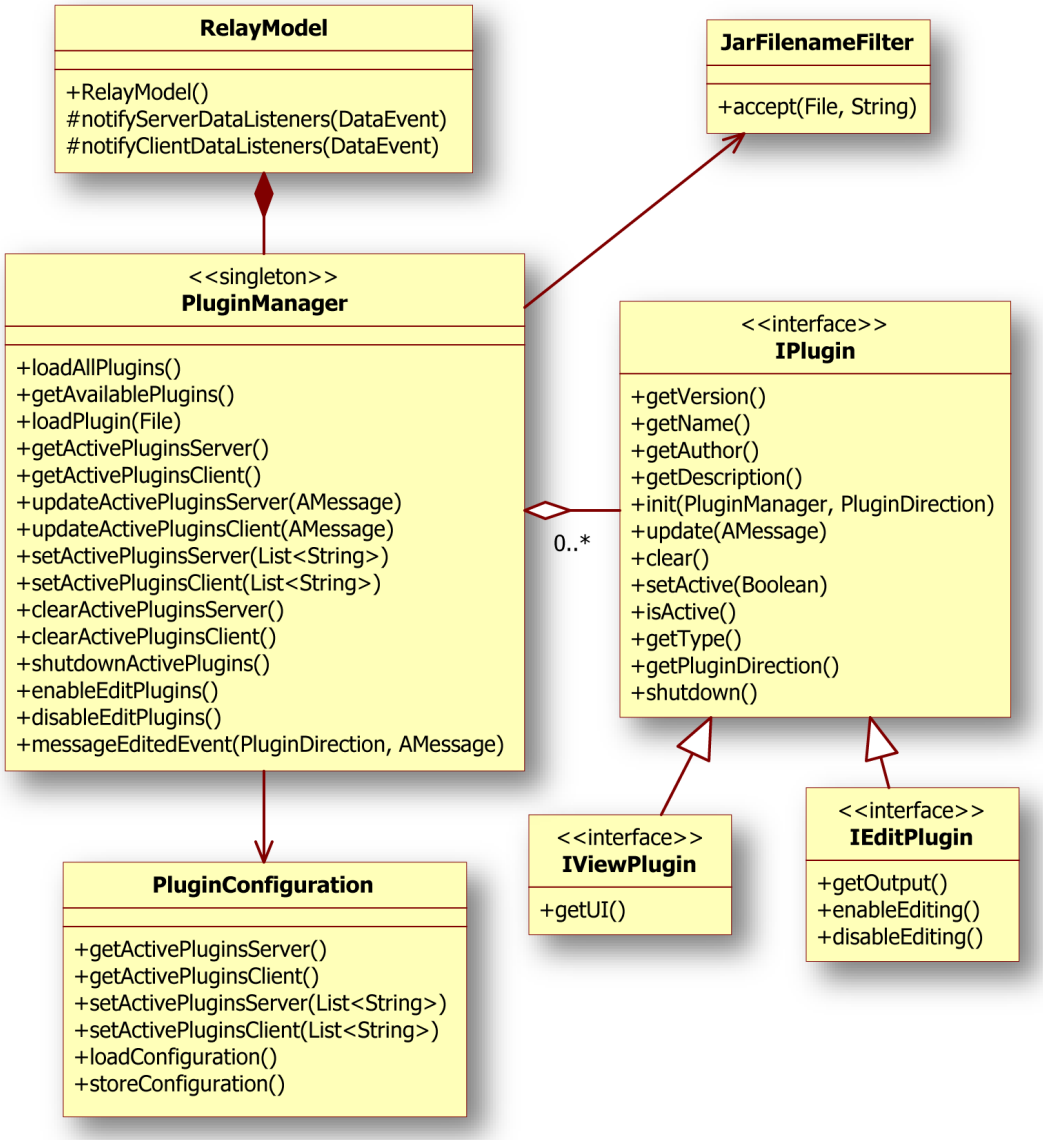


Bild 4.10: Klassendiagramm Plugin-Verwaltung

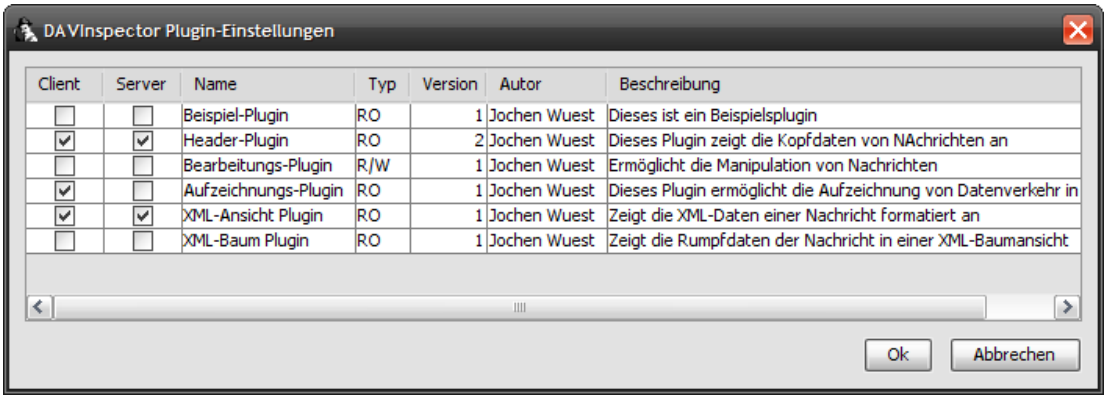


Bild 4.11: Dialog zur Konfiguration der Plugins

Zur komfortablen Verwaltung der aktivierten Plugins auf Client- und Serverseite dient ein Dialog bei dem der Nutzer für Client und Server getrennt die jeweilig zu nutzenden Plugins auswählen kann. Dieser Dialog ist in Bild 4.11 dargestellt. Hierbei werden in tabellarischer Form die verfügbaren Plugins und deren Eigenschaften angezeigt. Der Benutzer kann die Plugins jeweils getrennt für Server- und Client-Seite durch die Auswahl der entsprechenden Checkboxen aktivieren. Die aktivierten Plugins werden bei Beendigung der Konfiguration in die Liste der aktiven Plugins für Client- beziehungsweise Server-Seite abgelegt. Auf diese Listen greift später der `PluginManager` zurück, um die gewählten Plugins zu steuern und mit aktuellen Daten zu versorgen.

Die Konfiguration der Plugins wird mit Hilfe der Java-Bibliothek `java.util.Properties` gespeichert und wieder eingelesen. Die Konfigurationsdaten werden hierbei in einer einfachen Textdatei abgelegt. Diese Funktionalität wird von der Klasse `PluginConfiguration` gekapselt. Existiert beim Start des DAVInspectors eine Plugin-Konfiguration, so wird versucht die darin als „aktiv“ abgespeicherten Plugins in die jeweilige Liste der aktiven Plugins einzutragen. Sollte sich darunter ein Plugin befinden, das zu diesem Zeitpunkt nicht mehr verfügbar ist, so wird es aus der aktuellen Konfiguration entfernt.

### 4.2.4 Grafische Benutzeroberfläche

Die grobe Aufteilung der grafischen Oberfläche wurde bereits in Kapitel 3.3.1 erläutert. In Bild 4.12 ist die Umsetzung dieses Konzeptes dargestellt. Im oberen Bereich befinden sich das Menü der Anwendung und eine Werkzeugleiste, die den schnellen Zugriff auf häufig genutzte Funktionen ermöglicht. Darunter werden die bei der Kommunikation anfallenden Daten für die Client-Seite links und die Server-Seite rechts in optisch voneinander getrennten Bereichen angezeigt. In der Basisansicht werden die Daten unverändert angezeigt. Diese Ansicht wird auch als „Rohansicht“ bezeichnet und ist immer verfügbar. Weitere, durch Plugins realisierte Ansichten, werden in jeweils eigenen Registerkarten angezeigt. In der Rohansicht ist ein Kontextmenü verfügbar, dargestellt in Bild 4.13. Das Kontextmenü erlaubt das Kopieren, Einfügen und Ausschneiden sowie den Export der Daten in eine Datei.

Um die Übersichtlichkeit bei großen Datenmengen beziehungsweise komplexeren und umfangreicheren Kommunikationsabläufen zu verbessern, wird dem Benutzer im unteren Bereich der Anwendung eine Nachrichtenhistorie zur Verfügung gestellt. In einer tabellarischen Darstellung wird hierbei pro Anfrage und Antwort eine Zeile hinzugefügt. Diese Einträge spiegeln die Reihenfolge des Empfangs der Nachrichten wider. Hierbei sollte dem Nutzer bewusst sein, dass eine direkte Zuordnung von einer Antwort auf eine vorausgegangene Anfrage nicht immer richtig ist, vor allem wenn gleichzeitig mehrere Verbindungen vorhanden sind. Die Auswahl einer Zeile in der Tabelle aktualisiert die Ansichten mit den entsprechenden Daten der gewählten Nachricht. So ist zu jedem Zeitpunkt eine erneute Analyse der bereits aufgezeichneten Daten möglich.

Die Klassendiagramme in Bild 4.14 und Bild 4.15 visualisieren die zur grafischen Oberfläche gehörenden Klassen. Bild 4.14 verdeutlicht die Umsetzung des MVC-Konzeptes durch die drei Klassen `RelayModel`, `MainView` und `MainController` (vgl. Kapitel 2.3.4). Die Klasse `MainView` implementiert die Schnittstellen `INewMessageListener` und `ILoadMessageListener` damit die Ansicht bei neuen Nachrichten oder geladenen Nachrichten aktualisiert werden kann. Die nötigen Informationen

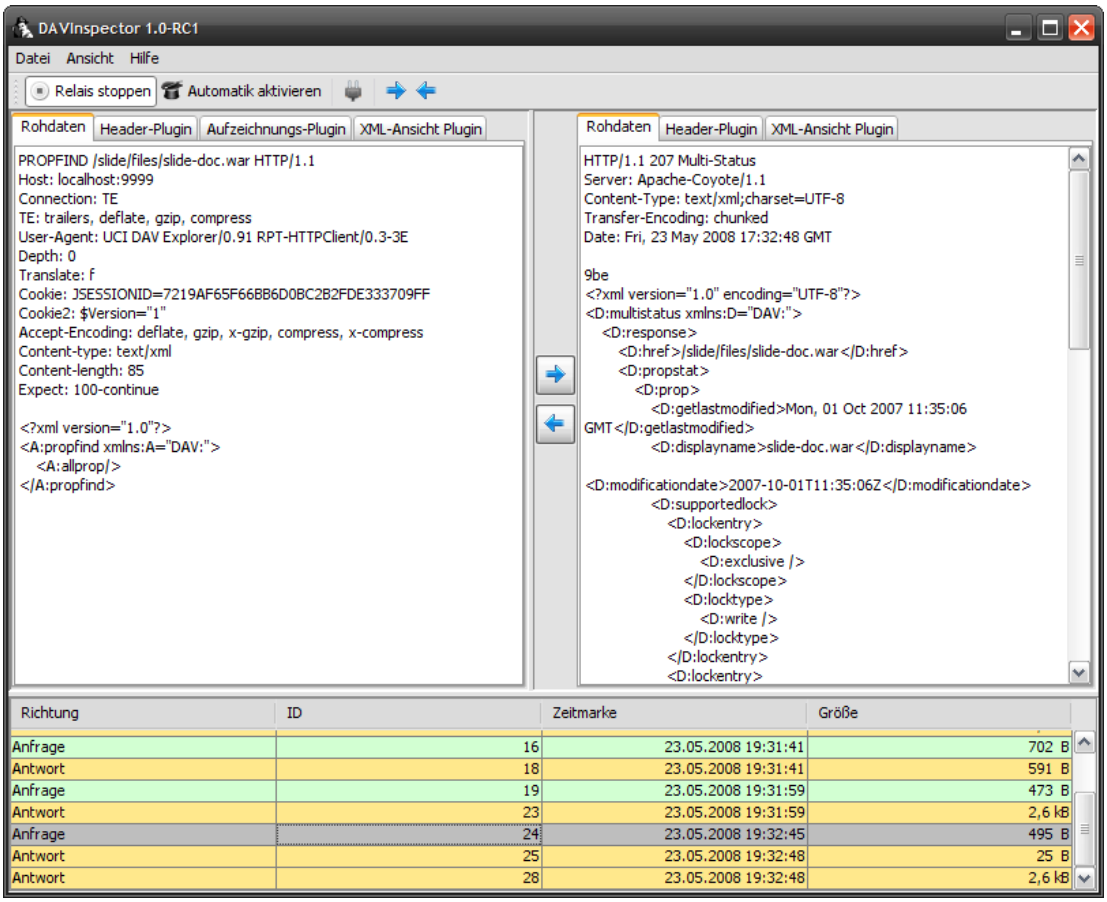


Bild 4.12: DAVInspector Hauptfenster

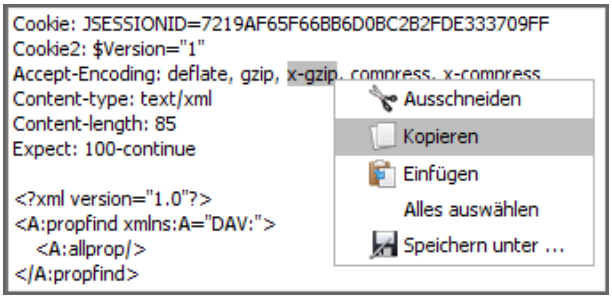


Bild 4.13: Kontextdialog der Rohansicht

dazu befinden sich in den Objekten des Typs `MessageEvent`. Weiterhin zeigt das Diagramm die Hilfsklassen `Constant` und `Util`. Diese Klassen stellen Funktionen und Konstanten bereit, die in verschiedenen Paketen der Anwendung genutzt werden. Die Klasse `Configuration` kapselt die Konfiguration des DAVInspectors und kann über entsprechende Dialoge verändert werden.

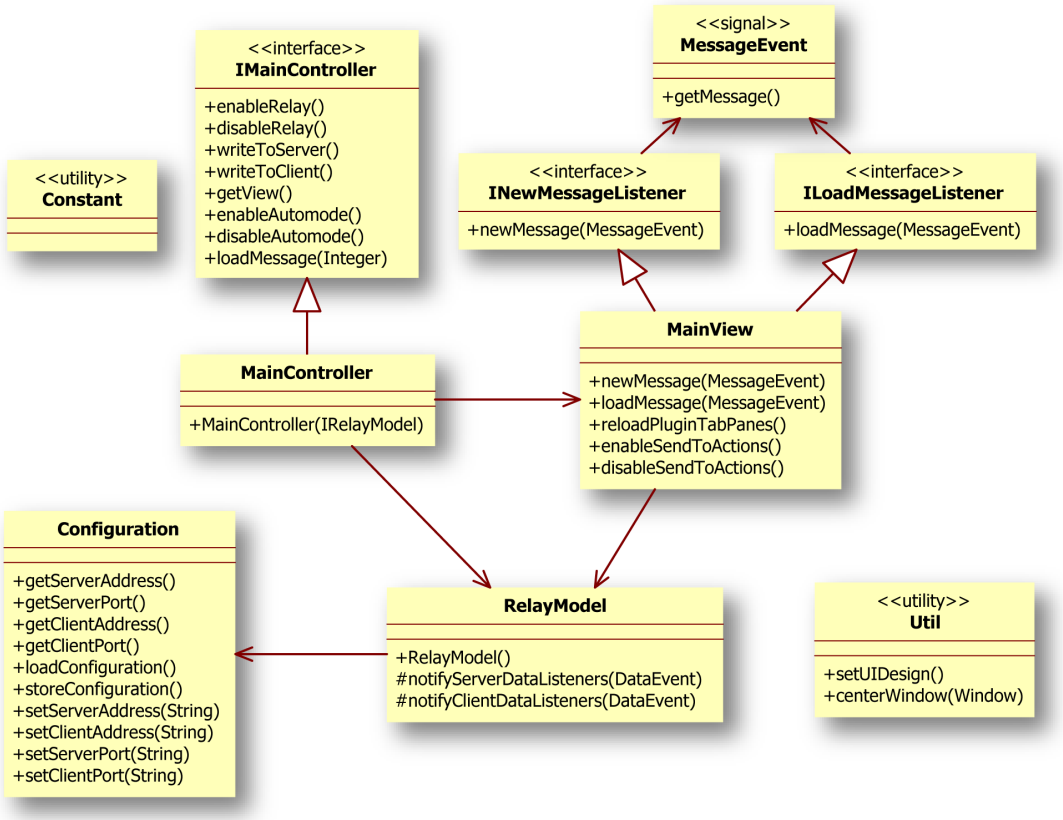


Bild 4.14: Klassendiagramm der grafischen Oberfläche

Bild 4.15 zeigt alle weiteren Klassen, die für die Realisierung der grafischen Oberfläche erforderlich sind. Die Klasse `JRawTextPane` implementiert die Rohansicht mit dem zugehörigen Kontextmenü. Die Klasse `ExportDialog` setzt den in Bild 4.9 abgebildeten Export-Dialog um. Die Konfiguration des DAVInspectors über die grafische Oberfläche ermöglicht die Klasse

`ConfigurationDialog`. Die Hilfsklasse `UIResource` kapselt den Zugriff auf Icons und Grafiken.

Ein Informations-Dialog wird von der Klasse `AboutDialog` realisiert. Um aus diesem Dialog direkt Hilfeseiten und weitere Informationen zum Programm öffnen zu können, wurde die Klasse `JHyperlinkLabel` implementiert. Diese Klasse erweitert die Oberklasse `JLabel`. Bei einem Klick auf ein solches Label-Element wird die hinterlegte Internetadresse in dem Standardbrowser des Systems geöffnet.

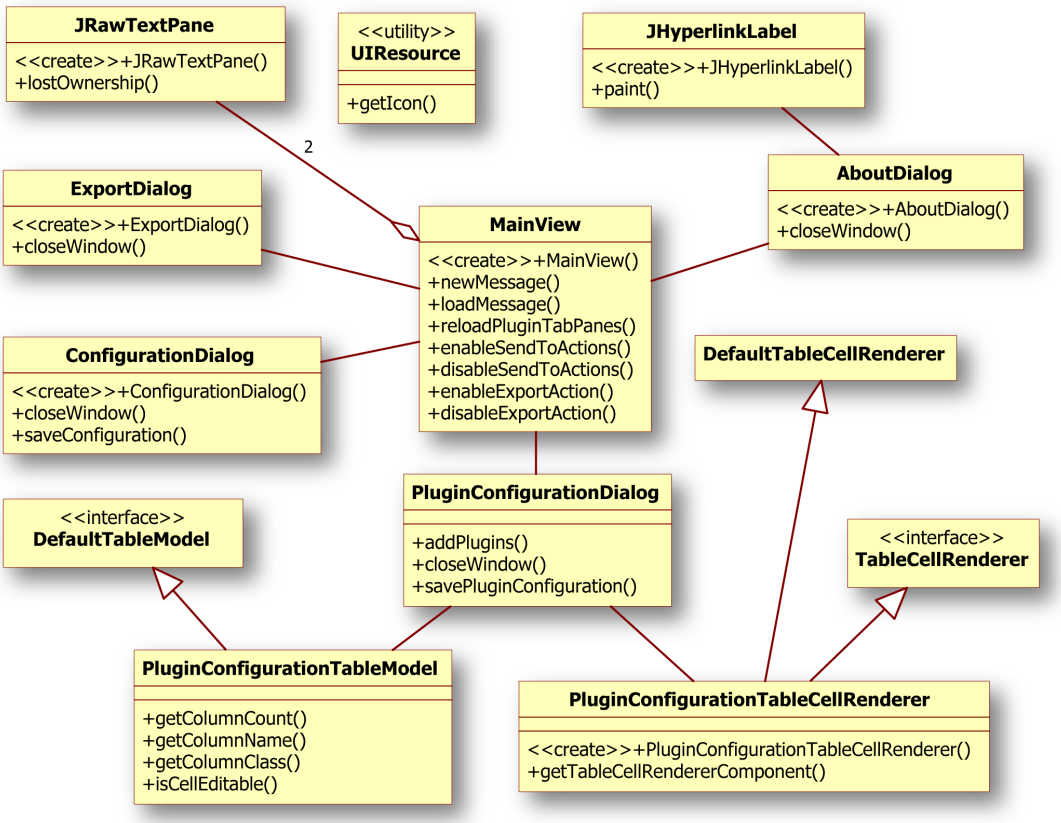


Bild 4.15: Klassendiagramm UI Details

Die Klasse `PluginConfigurationDialog` ermöglicht die grafische Konfiguration der Plugins. Die Maske dieses Dialogs ist in Bild 4.11 dargestellt. Um die hier gezeigte Darstellung realisieren zu können, musste für die Daten der Tabelle ein eigenes Datenmodell realisiert werden. Die Klas-

se `PluginConfigurationTableModel` setzt dieses Modell um und implementiert die Schnittstelle `DefaultTableModel`. Damit die einzelnen Plugins komfortabel per Checkbox aktiviert und deaktiviert werden können, ist es weiterhin erforderlich einen eigenen Renderer für die Zellen der Tabelle zu erstellen, der die grafische Umsetzung der Elemente erzeugt. Die Klasse `PluginConfigurationTableCellRenderer` kapselt diese Funktionalität. Diese Klasse ist eine Unterklasse von `DefaultTableCellRenderer` und realisiert die Schnittstelle `TabelCellRenderer`.

### 4.2.5 Protokollierung

Informationen, die durch die Protokollierung („Logging“) von Ergebnissen einzelner Methoden oder von Abläufen innerhalb des DAVInspectors gewonnen werden, können bei der Behebung von Fehlern helfen und die Nachvollziehbarkeit von Fehlermeldungen erleichtern. Deshalb wurden an entsprechenden Stellen, gemäß dem in Kapitel 3.3.6 vorgestellten Konzept, Protokollfunktionen vorgesehen. Als Logging-Werkzeug wird „Apache log4j 1.2“[2] verwendet. Die Bibliothek Log4j wurde speziell für Java-Applikationen entwickelt und bietet eine flexible Konfiguration. Die Ausgaben können durch verschiedene „Log-Level“ gesteuert werden. Abhängig vom gewünschten Detaillierungsgrad der Ausgabe können in der Konfigurationsdatei verschiedene Log-Level eingestellt werden. Verfügbar sind die Level „TRACE“, „DEBUG“, „INFO“, „WARN“, „ERROR“ und „FATAL“. Weiterhin kann das Format der Ausgabe je nach Bedarf definiert werden. Die Ausgabe der Log-Daten kann in Dateien, Konsolen, Netzwerkschnittstellen und externen Programmen erfolgen. Ein weiterer Vorteil ist, dass die Ausgabemöglichkeiten hierarchisch geschachtelt werden können. Somit ist die Möglichkeit gegeben, nur die Ausgaben einzelner Klassen oder die Ausgaben aller Klassen durch eine Änderung der Konfiguration zu erhalten.

[2] Apache log4j: http://logging.apache.org/log4j/

```
1  # Log-Level of the root Logger
2  log4j.rootLogger=FATAL, system
3  # Set Log-Level for DAVInspector
4  log4j.logger.de.dlr.davinspector=ERROR, system
5  log4j.logger.de.dlr.davinspector.plugin=ERROR, plugin
6  log4j.logger.de.dlr.davinspector.relay=DEBUG, relay
7  # Configuration of the general Log-File
8  log4j.appender.system=org.apache.log4j.FileAppender
9  log4j.appender.system.File=log/davinspector.log
10 log4j.appender.system.layout=org.apache.log4j.PatternLayout
11 log4j.appender.system.layout.ConversionPattern=%p %t %c - %m%n
```

Listing 4.1: Konfigurationsdatei für Log4j (verkürzt)

Die Konfiguration von Log4j ist in einer „.properties“-Datei abgelegt. In Listing 4.1 ist der Inhalt der Konfigurationsdatei in verkürzter Form dargestellt. Diese gliedert sich in drei Teile. Der erste Teil (Zeile 2) konfiguriert den Standard-Logger.

```
1  PropertyConfigurator.configure("log4j.properties");
2
3  public class RelayThread {
4          private static Logger myLogger
5                          = Logger.getLogger(RelayThread.class);
6
7          public void test() {
8                  myLogger.error(anIOExn.getMessage(), anIOExn);
9
10                 if (myLogger.isDebugEnabled()) {
11                     myLogger.debug("Method␣:" + myMethod);
12                     myLogger.debug("Chunked:" + isChunked);
13                 }
14         }
15 }
```

Listing 4.2: Beispiel für die Anwendung von Log4j

In dem zweiten Abschnitt (Zeile 4 – 6) werden drei Logger für die Pakete „davinspector“, „plugin“ und „relay“ definiert. Im dritten Abschnitt (Zeile 8 – 11) wird ein so genannter „Appender“ erstellt. Die Appender sind für ei-

ne formatierte Ausgabe der Log-Informationen zuständig. Im vorliegenden Fall wird eine Ausgabe in eine Datei mit dem Namen „log/davinspector.log“ definiert. Weiterhin wird noch das Format der Log-Informationen durch ein „PatternLayout“ spezifiziert.

Listing 4.2 zeigt ein Beispiel für die Anwendung von Log4j. In Zeile 1 wird der Logger mit der angegebenen Konfigurationsdatei eingerichtet. Den Zugriff auf eine Instanz des Loggers innerhalb einer Klasse zeigen die Zeilen 4 bis 5. Eine einzelne Log-Ausgabe ist in Zeile 8 abgebildet und Zeile 10 bis 12 verdeutlichen die Möglichkeit mehrere Log-Ausgaben zusammen zu fassen.

## 4.3 Plugins

In diesem Abschnitt wird die Realisierung einzelner Plugins erläutert. Die im Folgenden vorgestellten Plugins wurden alle für die Überprüfung des Plugin-Konzeptes benutzt und dienen gleichzeitig als Vorlage für weitere Plugins.

### 4.3.1 Kopfdaten-Plugin

Das Kopfdaten-Plugin stellt die Kopfdaten („Nachrichten-Header“) der Anfragen und Antworten des HTTP-/WebDAV-Protokolls in verschiedenen Kategorien tabellarisch aufbereitet dar. Bild 4.17 zeigt die Umsetzung dieses Plugins. Zurzeit werden drei Kategorien verwendet:

- **Allgemein** – In der Kategorie „Allgemein“ werden Informationen abgelegt, die sowohl in Anfragen als auch in Antworten vorhanden sind. Dazu zählen zum Beispiel „Protocol Version“ oder „Host“.
- **Anfrage/Antwort** – Diese Kategorie enthält auf der Client-Seite nur Kopfdaten, die Bestandteil von Anfragen sind. Auf der Server-Seite hingegen werden nur Antwort spezifische Informationen angezeigt.

- **Verschiedenes** – Hier werden alle übrigen Daten des Protokollkopfes abgelegt.

Diese Kategorien können jederzeit um weitere Kategorien ergänzt werden. So könnte zum Beispiel, eine Kategorie „Encodierung“ hinzugefügt werden, die alle Informationen zur verwendeten Zeichen- und Übertragungskodierung darstellt.

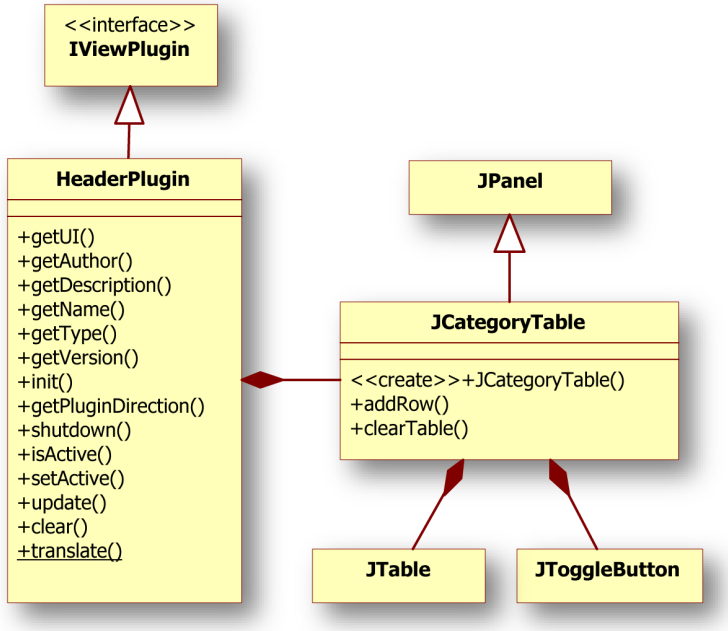


Bild 4.16: Klassendiagramm Kopfdaten-Plugin

Für die grafische Darstellung der Kategorien wurde eine eigene Komponente entwickelt. Diese Komponente kapselt die angezeigten Daten und sorgt für eine einheitliche und übersichtliche Darstellung derselben. Eine Kategorie-Komponente setzt sich aus mehreren grafischen Elementen zusammen. Bild 4.16 zeigt das Klassendiagramm des Kopfdaten-Plugins. Die Klasse `JCategoryTable` ist eine Ableitung von der Klasse `JPanel` und besitzt zusätzliche Methoden, um Daten zu einer Kategorie hinzuzufügen oder zu löschen. Weiterhin besitzt jedes Kategorie-Element eine Tabelle der Klasse `JTable` zur Darstellung der Daten und eine Schaltfläche der Klasse `JToggleButton`. Durch einen Klick auf diese Schaltfläche kann das grafi-

sche Element zusammengeklappt oder expandiert werden. So können nicht benötigte Kategorien optisch minimiert und bei Bedarf erneut ausgeklappt werden.

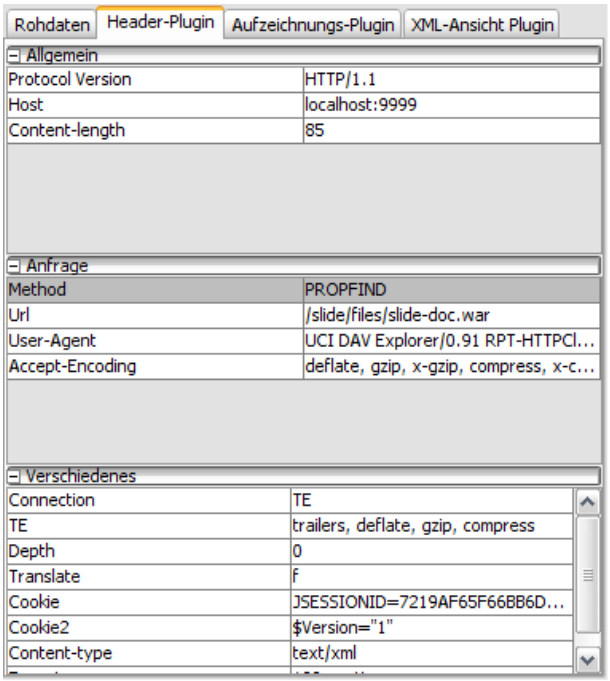


Bild 4.17: Kopfdaten-Plugin, Client-Seite

Das Plugin implementiert die Schnittstelle `IViewPlugin` und erhält hierdurch die aktuellen Kopfdaten einer Nachricht. Diese werden dann durch Auswertung von regulären Ausdrücken einer der Kategorien zugeordnet und zu der entsprechenden Tabelle hinzugefügt. Ein Tabelleneintrag setzt sich aus einem Schlüsselwort in der linken Spalte und den zugehörigen Werten in der rechten Spalte zusammen.

### 4.3.2 XML-Ansicht-Plugin

Da die Daten im Rumpf des WebDAV-Protokolls in einem XML-Format übertragen werden, ist es sehr hilfreich, wenn diese Daten farblich und durch Einrückungen aufbereitet werden. Diese Ansicht wird durch das XML-Ansicht-Plugin realisiert. Einige WebDAV-Applikationen liefern die Daten strukturiert aus, andere jedoch nicht. Dieses Plugin stellt somit eine einheitliche Ansicht der XML-Daten zur Verfügung. Zudem findet gleichzeitig eine Syntaxüberprüfung statt. Die Realisierung dieses Plugins ist in Bild 4.19 dargestellt. Bei der Syntaxüberprüfung auftretende Fehler werden im unteren Teil des Fensters angezeigt.

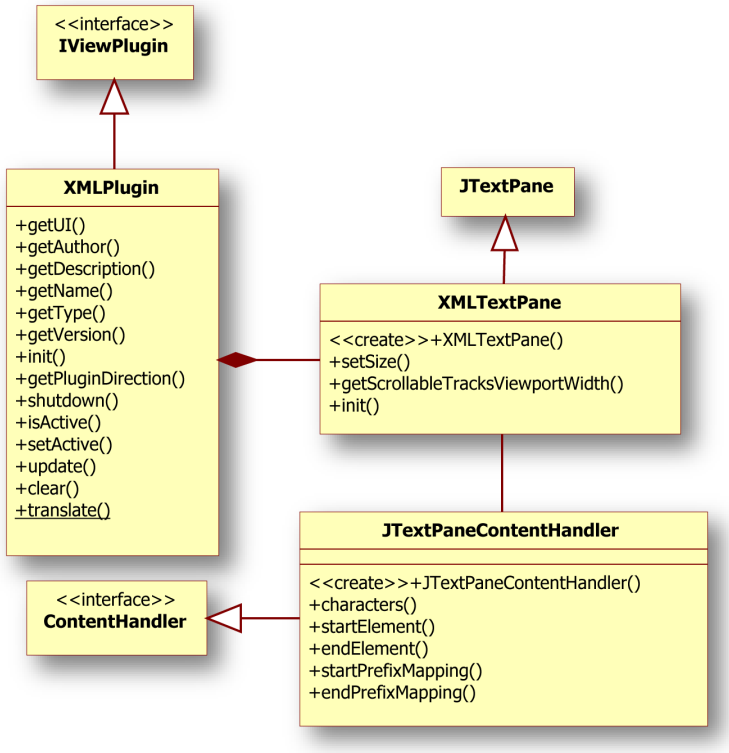


Bild 4.18: Klassendiagramm XML-Ansicht-Plugin

Bild 4.18 zeigt das Klassendiagramm dieses Plugins. Die Hauptklasse dieses Plugins, `XMLPlugin`, implementiert die Schnittstelle `IViewPlugin`. Die grafische Komponente besteht aus der Klasse `XMLTextPane`, die von

der Klasse `JTextPane` abgeleitet wurde. Um die XML-Daten einer Nachricht zu verarbeiten und gleichzeitig die Validität zu überprüfen, wird der SAX-Parser[3] von Java verwendet.

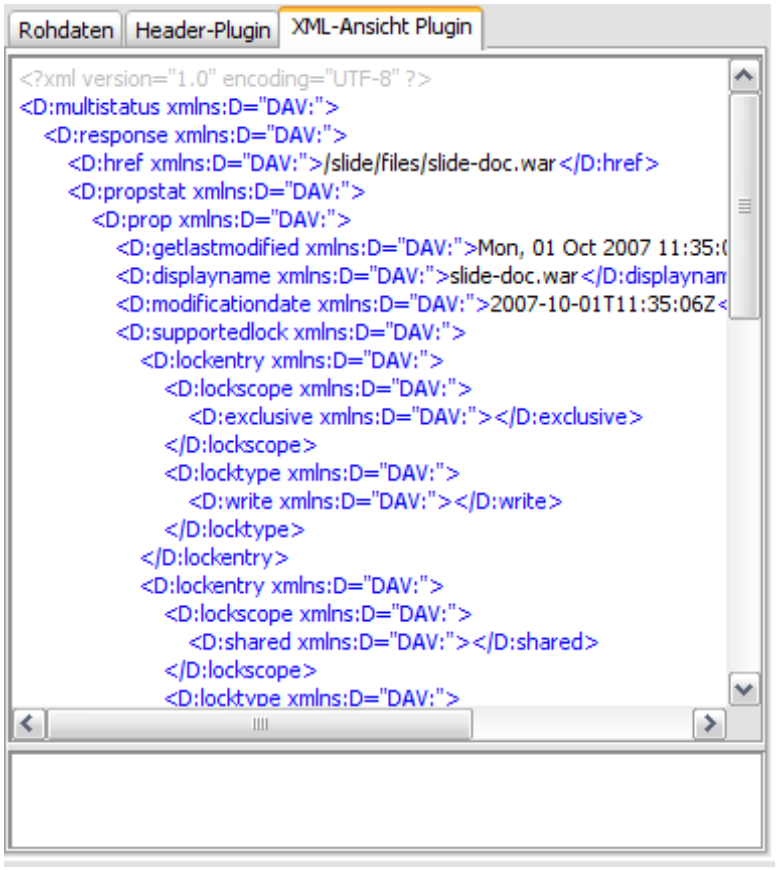


Bild 4.19: XML-Ansicht-Plugin

Weiterhin wird in der Klasse `JTextPaneContentHandler` die Schnittstelle `ContentHandler` realisiert, die einen Dokument-Handler definiert. Die XML-Daten werden sequenziell verarbeitet und bei bestimmten Ereignissen werden entsprechende Methoden ausgeführt. Die wichtigsten dieser Methoden sind `startElement()` (behandelt öffnende Tags), `endElement()` (behandelt schließende Tags) und `characters()` (behandelt Zeichen zwischen Tags).

[3] SAX: Simple API for XML, http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/parsers/SAXParser.html

### 4.3.3 XML-Baum-Plugin

Eine weitere Möglichkeit die XML-Daten des Nachrichtenrumpfes übersichtlich aufzubereiten bietet eine Baumdarstellung (Vgl. Kapitel 2.6). Diese Ansicht ist ebenfalls als Plugin realisiert worden und in Bild 4.20 dargestellt.

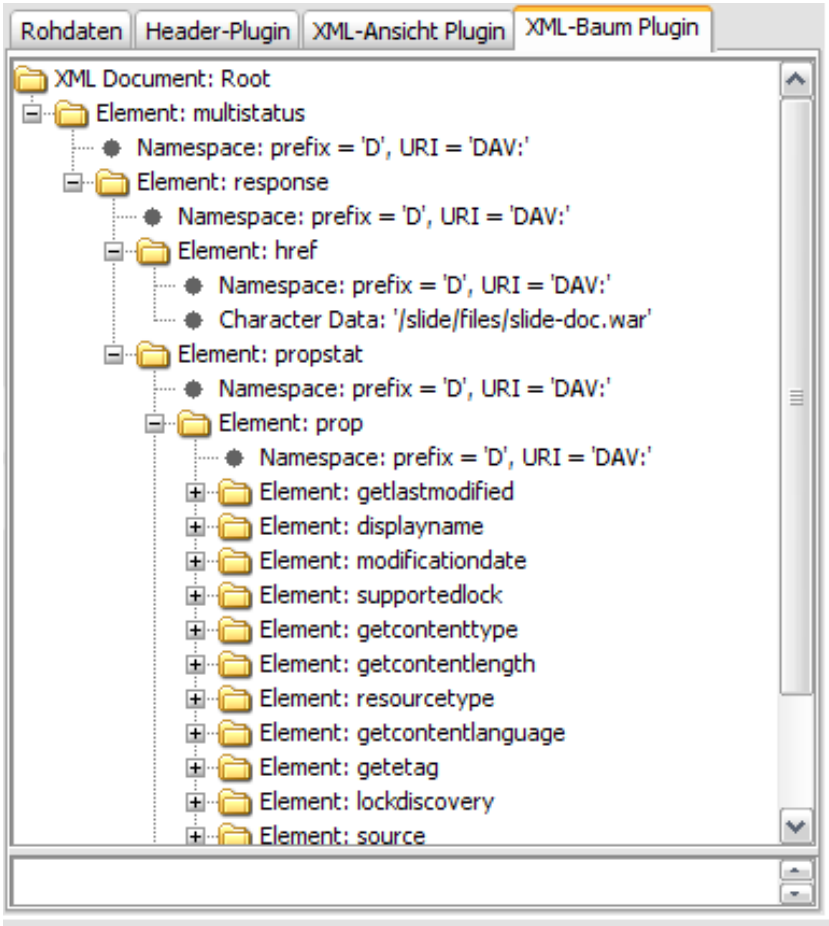


Bild 4.20: XML-Baum-Plugin

Die Verarbeitung der Daten erfolgt nach dem gleichen Prinzip, welches bereits in Kapitel 4.3.2 für das XML-Ansicht-Plugin beschrieben wurde. Die Ausgabe der Daten wird dabei jedoch durch eine von `JTree` abgeleitete Klasse erledigt. Fehler werden in einem extra Element im unteren Bereich des Plugins angezeigt. Bild 4.21 zeigt das zugehörige Klassendia-

gramm. Auch hier wird von der Hauptklasse `XMLTreePlugin` die Schnittstelle `IViewPlugin` realisiert.

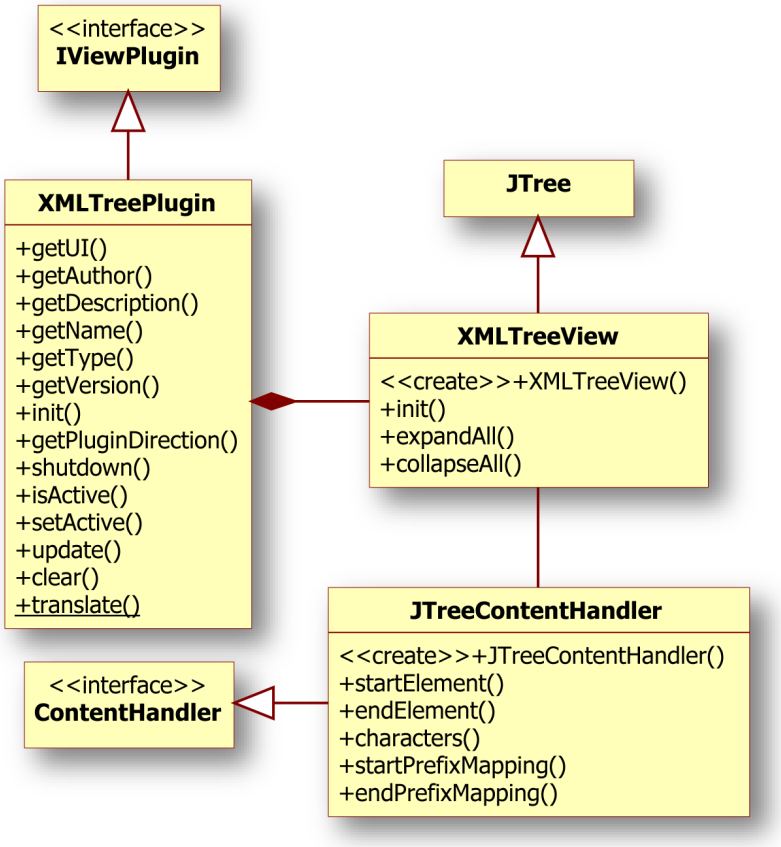


Bild 4.21: Klassendiagramm XML-Baum-Plugin

Die Klasse für die Baumdarstellung der Daten, `XMLTreeView`, verwendet die Klasse `JTreeContentHandler` als Dokument-Handler. Diese Klasse wiederum implementiert die Schnittstelle `ContentHandler` und verfügt damit über die erforderlichen Methoden, um die XML-Daten verarbeiten zu können.

### 4.3.4 Aufzeichnungs-Plugin

Das Aufzeichnungs-Plugin ist ein Beispiel für ein Plugin, welches keine visuelle Ausgabe erzeugt, sondern die grafische Oberfläche nur zur Konfiguration nutzt. Der Datenverkehr einer Datenrichtung wird von dem Plugin in einer Datei simultan zum Datenstrom abgespeichert. Die grafische Oberfläche dieses Plugins ist in Bild 4.23 abgebildet.

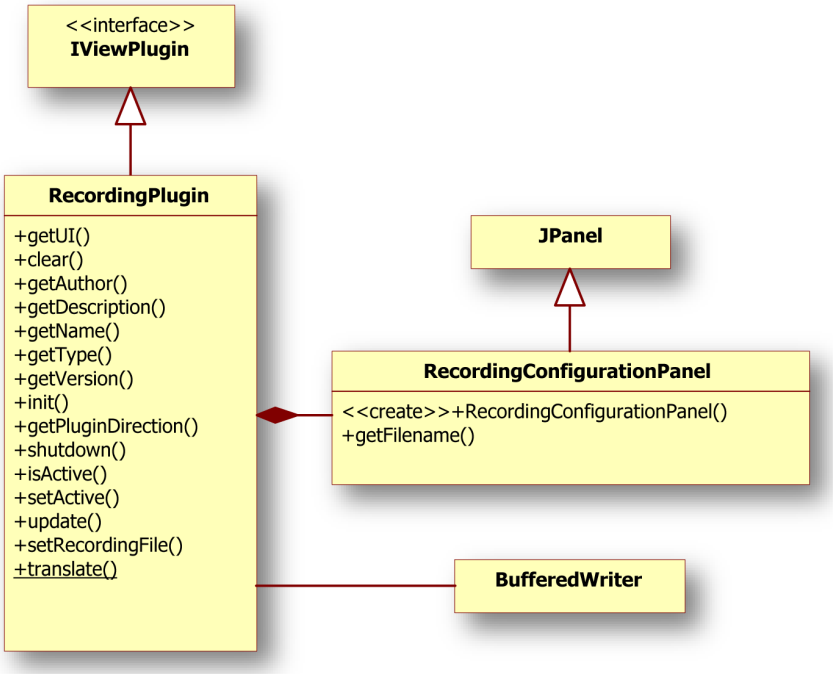


Bild 4.22: Klassendiagramm Aufzeichnungs-Plugin

Hierbei kann der Benutzer entweder den vorgeschlagenen Standard-Dateinamen verwenden oder einen individuellen Dateinamen für die Ausgabedatei vergeben. Der Standard-Dateiname setzt sich aus der Richtung der Nachricht, dem aktuellen Datum und der aktuellen Uhrzeit zusammen (Beispiel: `davi_client_2008-05-21_12-12-12.rec`).

Bild 4.22 zeigt das Klassendiagramm dieses Plugins. Die Klasse `RecordingPlugin` nutzt eine Instanz der Klasse `BufferdWriter`, um

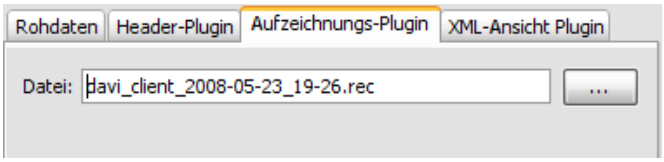


Bild 4.23: Aufzeichnungs-Plugin

die Daten in einer Datei abzulegen. Der Dateiname wird von der Klasse `RecordingConfigurationPanel` ermittelt.

### 4.3.5 Bearbeitungs-Plugin

Das Bearbeitungs-Plugin bietet dem Benutzer die Möglichkeit eine Nachricht vor der Weiterleitung zu modifizieren. Wie in Bild 4.25 dargestellt, setzt sich die Oberfläche dieses Plugins aus zwei Elementen zusammen.

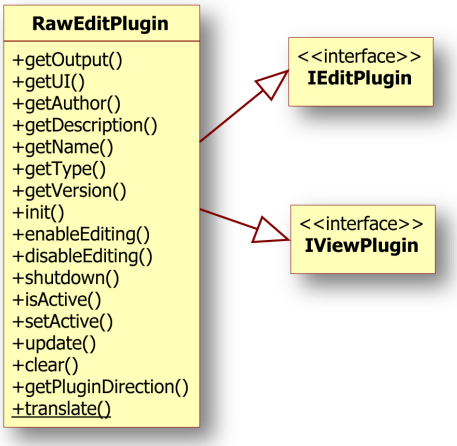


Bild 4.24: Klassendiagramm Bearbeitungs-Plugin

Im oberen Teil kann der Nutzer die aktuelle Nachricht editieren. Nach Beendigung der Bearbeitung muss dies durch einen Klick auf die Schaltfläche „Anwenden“ quittiert werden. Dadurch werden alle Plugins mit den editierten Daten aktualisiert.

Das Bearbeitungs-Plugin ist zurzeit das einzige Plugin, das die Editierung von Nachrichten ermöglicht. Das Klassendiagramm in Bild 4.24 zeigt, im Gegensatz zu den bisher betrachteten Plugins, dass die Klasse `RawEditPlugin` neben der Schnittstelle `IViewPlugin` auch die Schnittstelle `IEditPlugin` implementiert. Die Klasse besitzt damit zusätzlich die Methoden `getOutput()`, `enableEditing()` und `disableEditing()`.

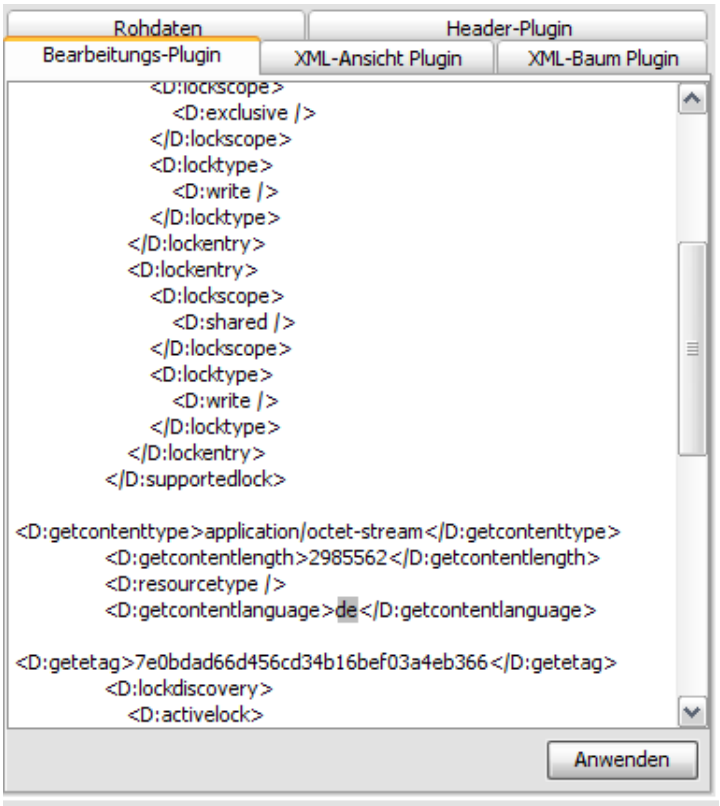


Bild 4.25: Bearbeitungs-Plugin

Die Quittierung der Bearbeitung löst ein Ereignis des Typs `messageEditedEvent` aus, und daraufhin kann die geänderte Nachricht durch die Methode `getOutput()` abgerufen werden. Die Methoden `enableEditing()` und `disableEditing()` können dazu verwendet werden die Bearbeitung von Nachrichten gezielt zu zulassen oder zu verhindern.

## 4.4 Internationalisierung

Mit den in Java vorhandenen Bibliotheken `java.util.Local` und `java.util.ResourceBundle` soll die Basis für die Internationalisierung der Applikation geschaffen werden. Exemplarisch wird dies für die deutsche Sprache durchgeführt. Hierbei besitzt die Bibliothek `java.util.Local` Methoden, die automatisch die richtige Sprachdatei auswählen, soweit diese vorhanden ist. Ansonsten wird die Standard-Sprachdatei verwendet.

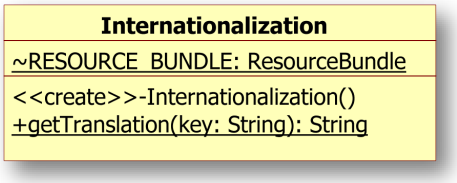


Bild 4.26: Die Hilfsklasse „Internationalization“

Der Zugriff auf die entsprechende Übersetzung erfolgt im Quellcode dann über die entsprechenden Schlüsselwörter. In Listing 4.3 wird exemplarisch anhand eines `JLabels` die Benutzung der Internationalisierung verdeutlicht. Statt direkt den Text des Labels zu setzen wird die Methode `getTranslation(String key)` der Hilfsklasse `Internationalization` genutzt. Diese Methode greift auf die von `java.util.Local` geladene Sprachdatei zurück und liefert die entsprechende Übersetzung als String zurück. Bild 4.26 zeigt die UML-Notation dieser Klasse.

```
1 jLabel = new JLabel();
2 jLabel.setText(Internationalization.getTranslation("lb_client_port"));
```

Listing 4.3: Beispiel für Internationalisierung

Das Datenformat der Ressourcendateien für die jeweiligen Sprachen wird an dieser Stelle kurz erläutert. Als Basis dient eine Datei in der Standard-Sprache. Im folgenden Beispiel wird diese Da-

tei mit `BeispielBundle.properties` bezeichnet. Der entsprechende Dateiname für eine Übersetzung in eine andere Sprache setzt sich dann aus dem Dateinamen der Standard-Sprachdatei und der entsprechenden Sprach- und Landesabkürzung zusammen. Zum Beispiel lautet der Dateiname für eine deutsche Sprachdatei dann `BeispielBundle_de_DE.properties`.

```
#Actions
ac_exit=Exit
ac_exit_description=Exit the program

#AboutDialog
dlg_about_title=About DAVInspector
dlg_about_close=Close

#menu
menu_file=File

#io error dialog
dlg_ioerror_text=File IO-Error
```

Listing 4.4: Inhalt der Datei „BeispielBundle.properties“ (Englisch)

```
#Actions
ac_exit=Beenden
ac_exit_description=Das Programm schließen

#AboutDialog
dlg_about_title=Über DAVInspector
dlg_about_close=Schließen

#menu
menu_file=Datei

#io error dialog
dlg_ioerror_text=Fehler bei der Ein-/Ausgabe
```

Listing 4.5: Inhalt der Datei „BeispielBundle_de_DE.properties“ (Deutsch)

Der Inhalt der Ressourcendateien setzt sich aus mehreren Schlüssel-Werte-Paaren zusammen. Hierbei steht der jeweilige Schlüssel links von einem Gleichheitszeichen und der Wert des Schlüssels rechts davon. Die Bezeichnungen der Schlüssel müssen zwingend für jede Sprachdatei identisch sein, die Werte enthalten die jeweils gewünschte Übersetzung. Die Listings 4.4 und 4.5 zeigen jeweils einen kleinen Ausschnitt aus einer englischen und deutschen Sprachdatei.

Um den Überblick bei umfangreichen Sprachdateien zu gewährleisten, ist es nötig die Ressourcendateien mit Kommentaren zu versehen und in verschiedene Abschnitte einzuteilen. Ein Abschnitt repräsentiert dann zum Beispiel alle Übersetzungen für eine Maske oder einen Dialog. Weiterhin ist eine durchgängige und sprechende Benennung der Schlüssel eine zwingende Voraussetzung, um auch externen Entwicklern die Erstellung einer Sprachdatei in ihrer Landessprache zu ermöglichen. Hierbei wurden die Schlüsselnamen in Anlehnung an die ungarische Notation aus mehreren Bestandteilen zusammengesetzt, die die Stelle der Übersetzung in der Anwendung verdeutlichen. Zum Beispiel handelt es sich bei dem Schlüssel `dlg_ioerror_text` um einen beschreibenden Text in einem Dialog, der Ein-/Ausgabefehler anzeigt.

## 4.5 Dokumentation

Die Dokumentation der Software DAVInspector besteht aus mehreren Dokumenten. Diese sind für unterschiedliche Zielgruppen bestimmt, und der Fokus liegt je nach Dokumententyp auf einem anderen Bereich. Die Zielgruppen des DAVInspectors gliedern sich in drei Kategorien: Entwickler, Plugin-Entwickler und Anwender. Die Form, in der die Dokumentation vorliegen kann, reicht hierbei von verschiedenen Dateiformaten (Text-, HTML-, PDF-, Grafikdateien) über Informationen im Internet (Wikis, Bugtracker) bis hin zu Mailinglisten.

**Anwender**
Die Dokumentation für Anwender besteht aus einer Installationsanleitung. Diese wird durch eine Schritt-für-Schritt-Anleitung ergänzt. Eine

CHANGES-Datei gibt eine Übersicht über die zuletzt durchgeführten Änderungen und neu hinzugefügte Funktionen. Ein obligatorischer Bestandteil ist die Lizenz der Software, die dem Anwender die Haftungs- und Nutzungsrechte erläutert.

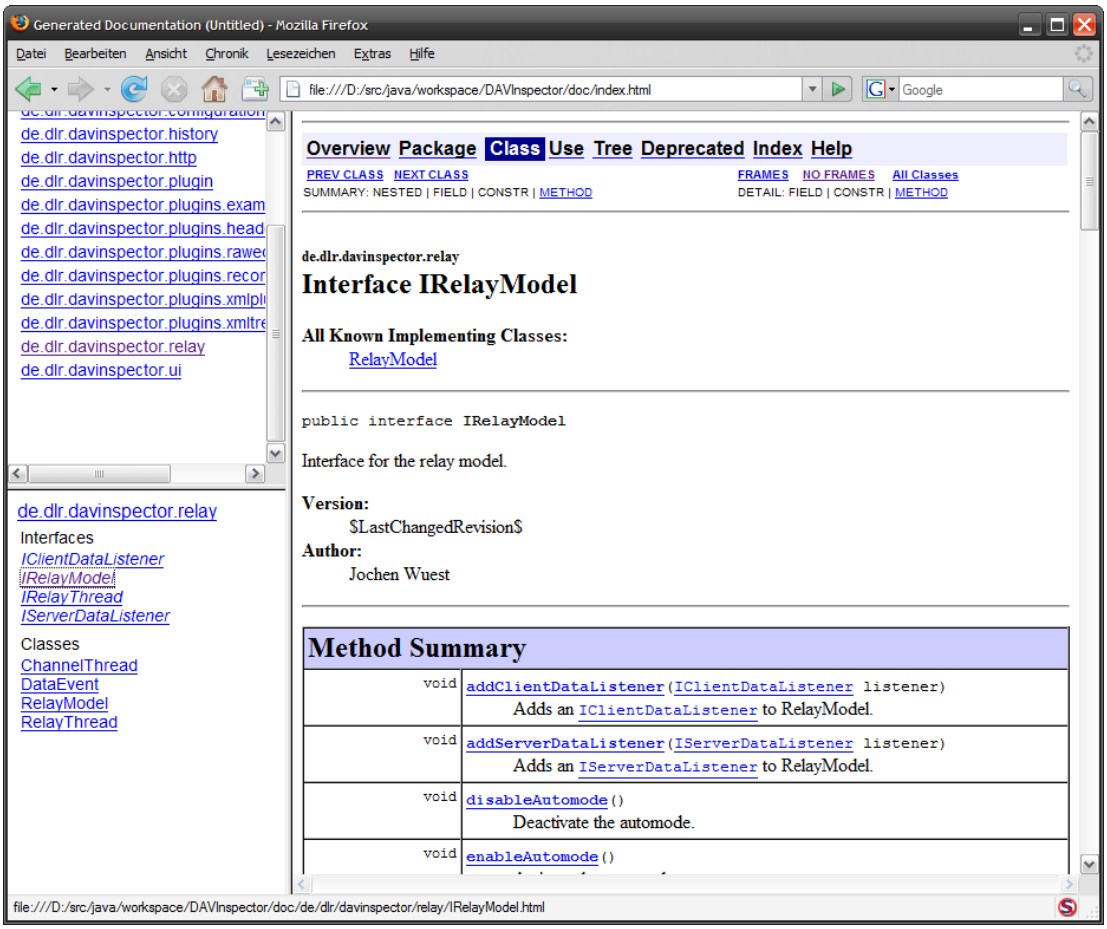


Bild 4.27: Entwickler-Dokumentation DAVInspector

### Entwickler

Die Dokumentation für Entwickler der Software besteht hauptsächlich aus den Kommentaren im Quellcode der Anwendung (vgl. Kapitel 4.1), den Anforderungsdokumenten (Lasten- und Pflichtenheft, vgl. Anhang) sowie verschiedenen UML-Diagrammen. Um die Kommentare aus dem Quellcode in eine besser lesbare Form zu überführen, wird unter Verwendung des Werkzeugs „javadoc“ eine Sammlung von entsprechenden HTML-Dokumenten erstellt. Bild 4.27 zeigt ein Beispiel eines solchen Dokumentes.

**Plugin-Entwickler**
Speziell für Plugin-Entwickler wurde eine Zusammenstellung der für die Entwicklung eines Plugins erforderlichen Schritte erstellt. Zusätzlich wurde ein Beispiel-Plugin erstellt, das als Hilfestellung und Praxisbeispiel auf Quellcode-Ebene dient.

## 4.6 Zusammenfassung

Die Umsetzung und Implementierung der in Kapitel 3 vorgestellten Konzepte wurde im Rahmen dieses Kapitels beschrieben. Hierbei wurden insbesondere die Schwerpunkte Kommunikation (Relais), Datenhaltung (Nachrichtenhistorie, Nachrichtenerkennung), grafische Benutzeroberfläche und Erweiterung (Plugin-Architektur, Plugins) behandelt. Neben den aufgezählten Schwerpunkten wurden die Aufteilung der Software in Komponenten, die durchgeführte Internationalisierung der Anwendung, die Nutzung der Protokollierung (Logging) und die Dokumentation diskutiert. Die bei der Erstellung der Software berücksichtigten Konventionen für das Konfigurationsmanagement und die verwendeten Programmierrichtlinien werden ausführlich in Anhang A und Anhang B erläutert.

# 5 Ergebnisse

Dieses Kapitel präsentiert die Ergebnisse der Arbeit und demonstriert anhand eines praxisnahen Beispiels die Funktionsweise der erstellten Software. Zunächst werden die Voraussetzungen zum Einsatz der Software definiert. Dazu werden die notwendige Software-Umgebung und die Konfiguration des DAVInspectors spezifiziert. Danach erfolgt die detaillierte Darstellung eines konkreten Anwendungsfalles.

## 5.1 Voraussetzungen und Umgebung

Der allgemeine Anwendungsfall des DAVInspectors ist in Bild 5.1 dargestellt. Hierbei kommen drei Komponenten zum Einsatz: ein WebDAV-Client, ein WebDAV-Server und der DAVInspector. Client und Server sind über den DAVInspector miteinander „verbunden“.

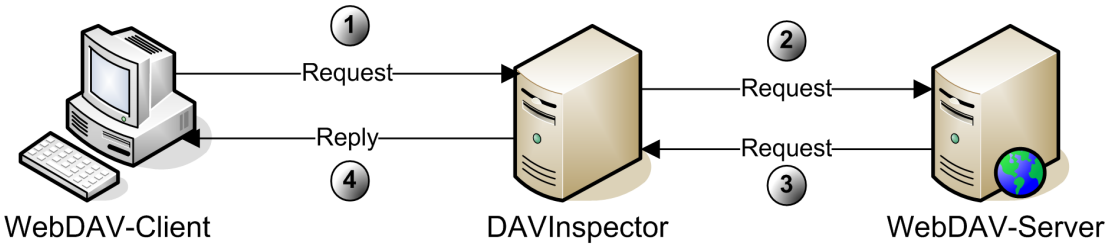


Bild 5.1: Anwendungsfall DAVInspector

Bei diesem Szenario ist es unerheblich, ob sich die drei Kommunikationspartner auf einem, zwei oder drei physikalischen Rechnern befinden. Wichtig ist lediglich, dass die Mitwirkenden über ein gemeinsam zugängliches Netzwerk verfügen. Der Nachrichtenaustausch zwischen den drei Teilnehmern läuft folgendermaßen ab:

1. Der WebDAV-Client stellt eine Anfrage an den DAVInspector.

2. Der DAVInspector leitet die Anfrage des Clients weiter. Im Automatikmodus erfolgt dies ohne Zutun des Benutzers. Im manuellen Modus wird die Nachricht erst nach Freigabe durch den Benutzer an den Server weitergeleitet.

3. Der Server verarbeitet die gestellte Anfrage und sendet eine entsprechende Antwort an den DAVInspector zurück.

4. Der DAVInspector leitet die Nachricht des Servers unter den gleichen Bedingungen wie unter 2. beschrieben an den Client weiter.

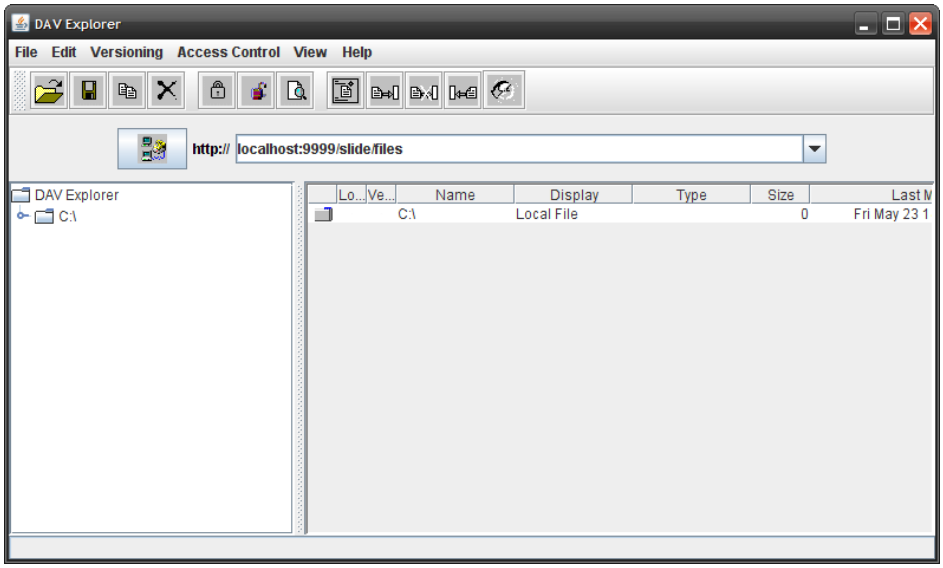


Bild 5.2: WebDAV-Client DAVExplorer

Für den im Folgenden geschilderten Anwendungsfall wird als WebDAV-Client die Software „DAV Explorer“[1] verwendet. Als Server-Software wird der „Apache Tomcat“[2] mit der Erweiterung für WebDAV „Slide“[3] des Apache Jakarta Projektes eingesetzt.

[1]DAV Explorer: http://www.davexplorer.org/
[2]Apache Tomcat: http://tomcat.apache.org/
[3]Apache Jakarta Slide: http://jakarta.apache.org/slide/howto-tomcat.html

Die Software DAV Explorer ist in der Programmiersprache Java geschrieben und unterstützt nahezu alle WebDAV-Methoden. Weiterhin ist die Anwendung einfach zu bedienen. Bild 5.2 zeigt die Oberfläche der Software im Ausgangszustand.

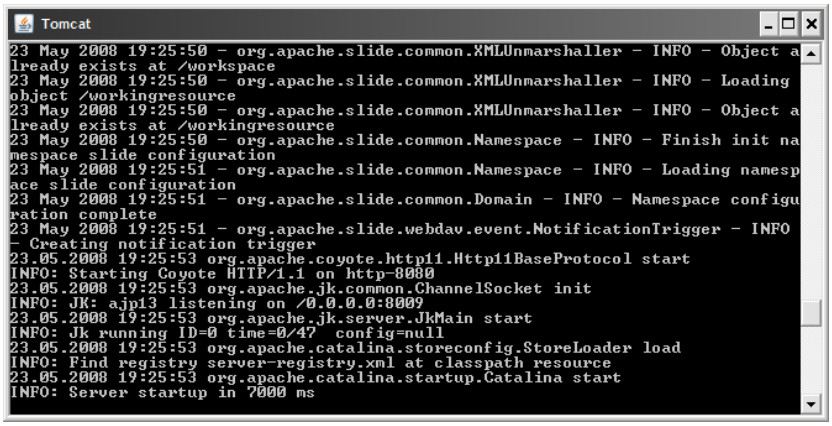


Bild 5.3: WebDAV-Server Tomcat/Slide

Die Server-Anwendung Apache Tomcat kann über ein Skript gestartet werden. Sobald die Erweiterung Slide installiert ist, steht die WebDAV-Funktionalität unter der Adresse `http://hostname:8080/slide` zur Verfügung. In Bild 5.3 ist die Konsolenausgabe des Servers dargestellt.

## 5.2 Konfiguration des DAVInspectors

Die Konfiguration des DAVInspectors besteht aus zwei Teilen. Es müssen die Einstellungen der Netzwerkschnittstellen und die der Plugins vorgenommen werden.

**Netzwerkschnittstellen**
Bild 5.4 zeigt den Konfigurationsdialog für die Netzwerkeinstellungen. Hier können vier Einstellungen vorgenommen werden:

- Client-Adresse: Dieses Feld bleibt üblicherweise leer. Es kann aber auch eine URL oder Netzwerkadresse spezifiziert werden. Dann werden nur Client-Anfragen an diese Adresse verarbeitet.
- Client-Port: In diesem Feld wird die Portnummer hinterlegt, auf der der DAVInspector Anfragen von Clients akzeptiert.
- Server-Adresse: Die URL oder IP-Adresse des Ziels muss in diesem Feld eingetragen werden. Befindet sich der Server auf demselben Rechner wie der DAVInspector, genügt die Angabe „localhost“.
- Server-Port: Hier wird der Zielport hinterlegt. In Verbindung mit dem oben erwähnten Tomcat/Slide-Server muss hier in der Standardkonfiguration die Portnummer „8080“ eingetragen werden.

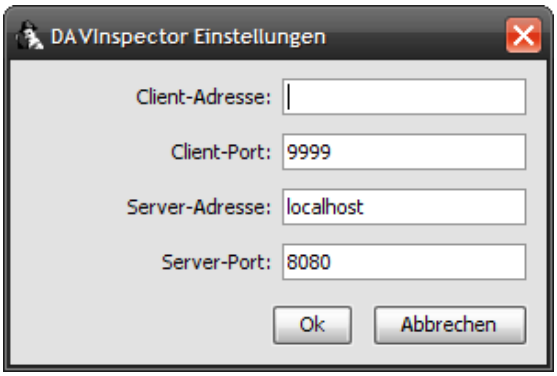


Bild 5.4: Konfiguration DAVInspector

**Plugins**

In Bild 4.11 ist der Konfigurationsdialog für die Plugins dargestellt. In tabellarischer Darstellung werden die wichtigsten Eigenschaften der vorhandenen Plugins aufgeführt: Name, Typ (RO – auswertende Plugins, R/W – manipulierende Plugins), Version, Autor und Beschreibung. Durch Markieren der Kontrollkästchen in den Spalten „Client“ und „Server“ werden die entsprechenden Plugins aktiviert. Sollte ein Plugin nur für die Client- oder Server-Seite verfügbar sein, so ist das jeweils andere Kontrollkästchen ausgegraut und nicht anwählbar.

## 5.3 Einsatz im Anwendungsfall

Bevor die eigentliche Beschreibung des Anwendungsfalls erfolgt, wird eine kurze Einführung in die grafische Oberfläche des DAVInspectors gegeben. Diese ist in Bild 5.5 dargestellt und besteht im Wesentlichen aus fünf Bestandteilen.

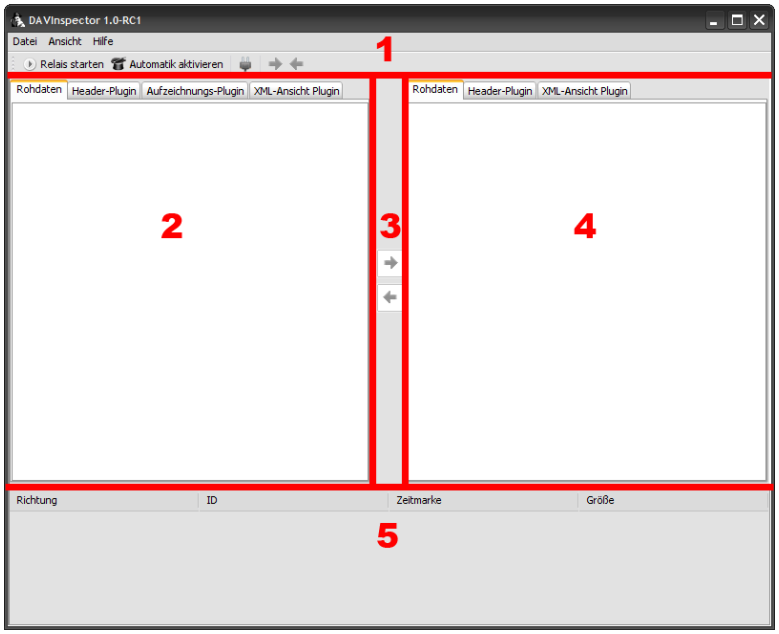


Bild 5.5: Aufteilung der Oberfläche des DAVInspectors

1. In diesem Bereich befindet sich die Menü- und Werkzeugleiste. Das Menü ermöglicht einen einfachen Zugriff auf alle wichtigen Funktionen des DAVInspectors. Die Werkzeugleiste erlaubt einen schnellen Zugriff auf häufig verwendete Funktionen. Dazu gehören etwa das Starten und Stoppen des Relais, die Aktivierung des Automatikmodus und die Konfiguration der Plugins.

2. In diesem Teil der Oberfläche werden die Plugins und die Rohdaten-Ansicht der Client-Seite visualisiert. Jedes Plugin ist dabei über eine eigene Registerkarte zugänglich.

3. Der Steg in der Mitte trennt die Client- von der Server-Seite und besitzt zudem zwei Schaltflächen, um im manuellen Betrieb des DAV-Inspectors die Nachrichten vom Client zum Server und umgekehrt weiterzuleiten.

4. Dieser Bereich stellt die Server-Seite dar. Auch hier wird jedes aktive Plugin durch eine eigene Registerkarte repräsentiert.

5. Die Nachrichtenhistorie wird in diesem Teil der Oberfläche angezeigt. Sofern bereits Nachrichten ausgetauscht wurden, werden diese in tabellarischer Form dargestellt. Durch einen Klick auf eine Tabellenzeile kann der Benutzer eine Nachricht auswählen. Diese wird dann, abhängig davon ob Anfrage oder Antwort, erneut durch die vorhandenen Client- oder Server-Plugins dargestellt.

Im Folgenden wird nun der in Abschnitt 5.1 beschriebene allgemeine Anwendungsfall Schritt für Schritt anhand der oben definierten Umgebung und Software nachvollzogen. Für das nachfolgend erörterte Beispiel wurde das Relais mit der in Abschnitt 5.2 beschriebenen Konfiguration gestartet und der Automatikmodus des DAVInspectors aktiviert.

Zunächst wird in der Software DAV Explorer die Adresse der gewünschten WebDAV-Ressource eingegeben. Diese Ausgangssituation wird auch in Bild 5.2 gezeigt. Hierbei muss beachtet werden, dass die URL auf die Client-seitige Adresse des DAVInspectors angepasst werden muss: Lautet die Adresse der WebDAV-Ressource ursprünglich `http://server-adresse:server-port/test/www` muss diese nun in `http://davinspector-adresse:davinspector-port/test/www` abgeändert werden. In der vorliegenden Konfiguration lautet die Adresse dann `http://localhost:9999/slide/files`. Daraufhin wird durch einen Klick auf die Schaltfläche „Connect“ im DAV Explorer eine Verbindung zu dieser Adresse aufgebaut. Es folgt eine Sequenz von zwei WebDAV-Anfragen und den zugehörigen Antworten. Das Ergebnis dieses Nachrichtenaustausches ist in Bild 5.6 und in Bild 5.7 dargestellt.

Wie zu Anfang angemerkt wurde der Automatikmodus des DAVInspectors verwendet. Wird der manuelle Modus verwendet, so muss der Benutzer

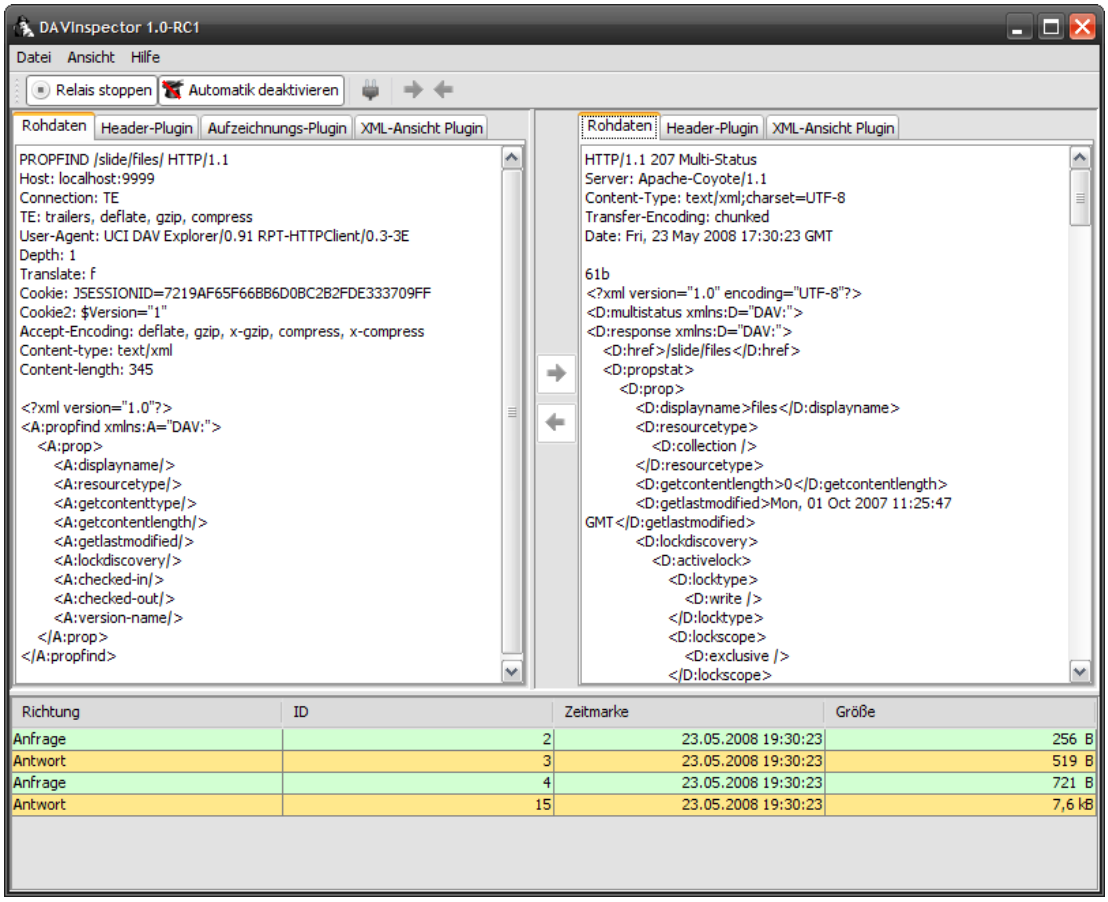


Bild 5.6: DAVInspector nach dem erfolgreichen Verbindungsaufbau

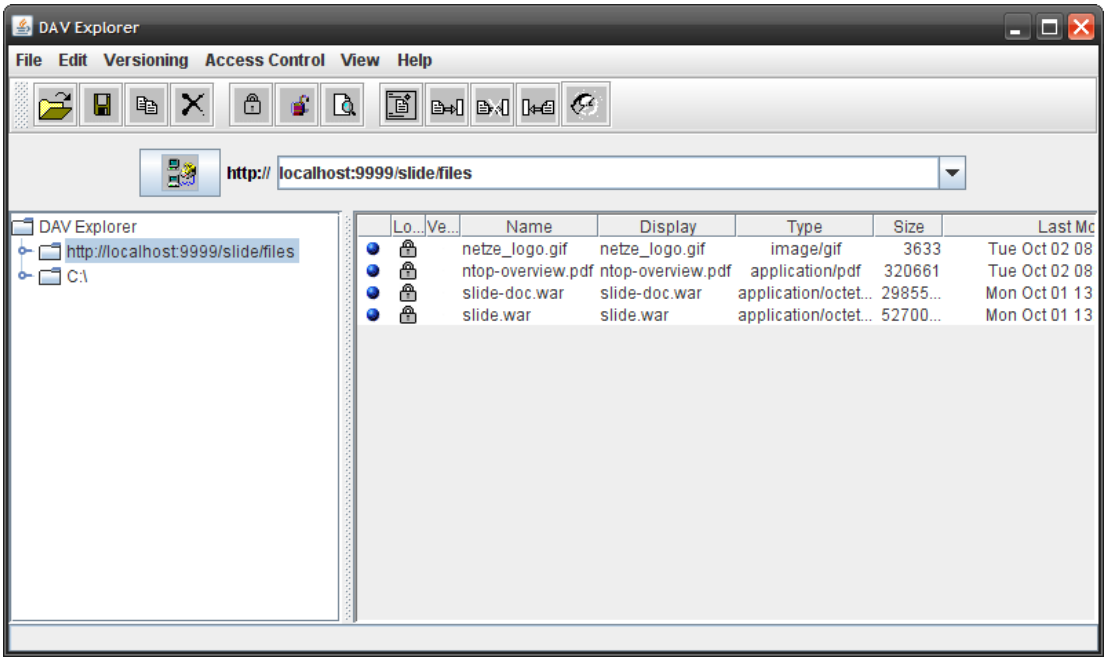


Bild 5.7: DAV Explorer nach dem erfolgreichen Verbindungsaufbau

selbst durch einen Klick auf die jeweilige Schaltfläche die Weiterleitung der Nachrichten veranlassen. Wenn die Weiterleitung nicht innerhalb einer gewissen Zeitspanne durchgeführt wird, bricht der Client den aktuellen Vorgang mit einem Fehler ab. Die Zeitbeschränkungen sind von der jeweils verwendeten Software abhängig und lassen sich gegebenenfalls in Grenzen verändern.

Beispielhaft werden zwei bei diesem Vorgang erzeugte Dateien betrachtet. Bild 5.8 zeigt einen Ausschnitt aus einer Log-Datei. In diesem Fall handelt es sich um die Ausgaben der Klasse `HTTPMessageParser` während eines Debugging-Vorgangs. Die hier abgebildeten Informationen können bei der Entwicklung und Fehlerbeseitigung verwendet werden. Im konkreten Fall bieten diese Ausgaben dem Entwickler die Möglichkeit die einzelnen Schritte des Parsers nachzuvollziehen und Ergebnisse einzelner Funktionen zu begutachten.

In Bild 5.9 ist ein Ausschnitt aus einer Datei des Aufzeichnungs-Plugins dargestellt. Hierbei wurde dieses Plugin während des oben beschriebenen

Anwendungsfalls aktiviert und hat alle Client-Anfragen in eine Datei geschrieben. Die von dem Aufzeichnungs-Plugin erzeugten Dateien können später für weitere Auswertungen oder für die Erstellung von Kommunikationssequenzen einer Emulation verwendet werden.

```
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - ---- new data event ----
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Direction: ClientToServer
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Direction: ClientToServer
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - found request line
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - content length found: 345
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Method :PROPFIND
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Status :0
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Version:HTTP/1.1
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Len    :345
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Chunked:false
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - normal message
DEBUG Thread-4 de.dlr.davinspector.http.HTTPMessageParser - Carry: 0
DEBUG Thread-3 de.dlr.davinspector.http.HTTPMessageParser - ---- new data event ----
```

Bild 5.8: Ausschnitt aus einer Log-Datei, gekürzt

```
OPTIONS /slide/files/ HTTP/1.1
Host: localhost:9999
Connection: Keep-Alive, TE
TE: trailers, deflate, gzip, compress
User-Agent: UCI DAV Explorer/0.91 RPT-HTTPClient/0.3-3E
Translate: f
Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress

PROPFIND /slide/files/ HTTP/1.1
Host: localhost:9999
Connection: TE
TE: trailers, deflate, gzip, compress
User-Agent: UCI DAV Explorer/0.91 RPT-HTTPClient/0.3-3E
Depth: 1
Translate: f
Cookie: JSESSIONID=4C59ACF9B064D246B1F229B60C8EDC2E
Cookie2: $Version="1"
Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
Content-type: text/xml
Content-length: 345

<?xml version="1.0"?>
```

Bild 5.9: Ausschnitt aus einer Rec-Datei (Aufzeichnungs-Plugin), gekürzt

Weiterhin überprüfen die Plugins „XML-Ansicht“ und „XML-Baum“ automatisch die XML-Daten eines Nachrichtenrumpfes. Sind die Daten valide, so werden sie von den beiden Plugins entsprechend dargestellt. Sollten die Daten jedoch einen oder mehrere Fehler aufweisen, so erfolgt keine Darstellung, sondern stattdessen die Ausgabe der Fehlermeldungen im unteren Teil der grafischen Oberfläche dieser Plugins.

Bild 5.10 zeigt die Fehlermeldung des XML-Ansicht-Plugins, wenn eine Nachricht ausgewertet wird, deren Rumpf HTML- statt XML-Daten enthält. Nach der XML-Definition sind HTML-Daten keine gültigen XML-Daten und damit ist der Rumpf dieser Nachricht nicht gültig.

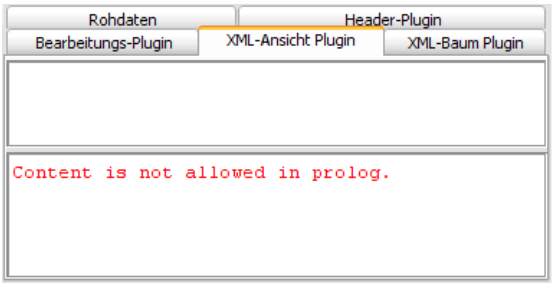


Bild 5.10: Fehlermeldung XML-Ansicht-Plugin

**Bearbeitungs-Plugins**

Bei der Verwendung von Bearbeitungs-Plugins muss gegenüber dem oben geschilderten Anwendungsfall beachtet werden, dass der Automatikmodus nicht verwendet werden kann. Die Nachrichten würden ansonsten direkt weitergeleitet, ohne dass der Nutzer eine Möglichkeit hätte, diese zu editieren. Es folgt eine kurze Schilderung der Nutzung eines Bearbeitungs-Plugins anhand einer `PROPFIND`-Abfrage.

Die Umgebung und das Vorgehen unterscheiden sich ansonsten nicht von dem bereits geschilderten Fall. Das bedeutet, der Benutzer verbindet den Client wie beschrieben über den DAVInspector mit dem Server. Die vom Client gestellte Anfrage (`PROPFIND`-Methode, Ressource „netze_logo.gif“) wird manuell an den Server weitergeleitet. Dieser verarbeitet die Anfrage und sendet eine entsprechende Antwort zurück. Bevor diese Antwort jedoch an den Client weitergeleitet wird, hat der Benutzer die Möglichkeit den Inhalt dieser Nachricht zu editieren.

Im vorliegenden Beispiel, siehe auch Bild 5.11, wird die Eigenschaft „displayname“ verändert (rote Markierung). Der ursprüngliche Wert dieser Variablen „netze_logo.gif“ wurde mit dem Wert „netze_kein_logo.gif“ überschrieben. Diese Änderung wird mit einem Klick auf die Schaltfläche „An-

wenden“ bestätigt und die Nachricht anschließend an den Client versendet. Das Ergebnis kann in Bild 5.12 betrachtet werden (rote Markierung).

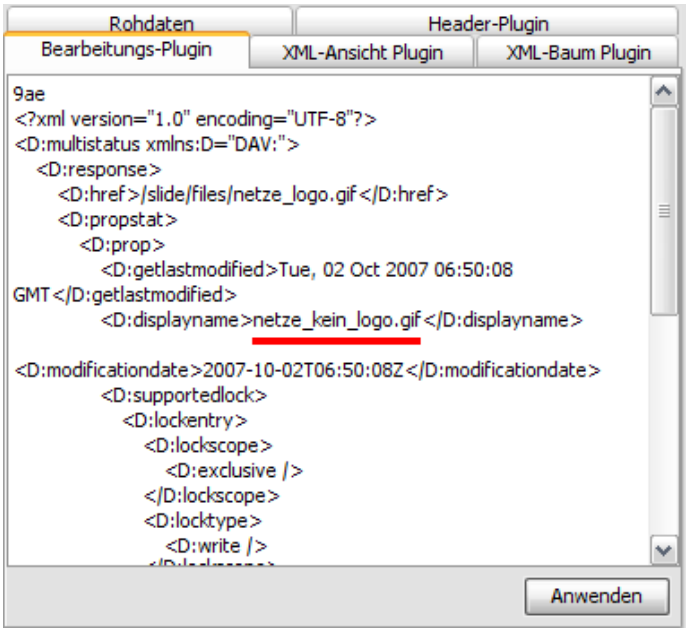


Bild 5.11: Bearbeitungs-Plugin DAVInspector

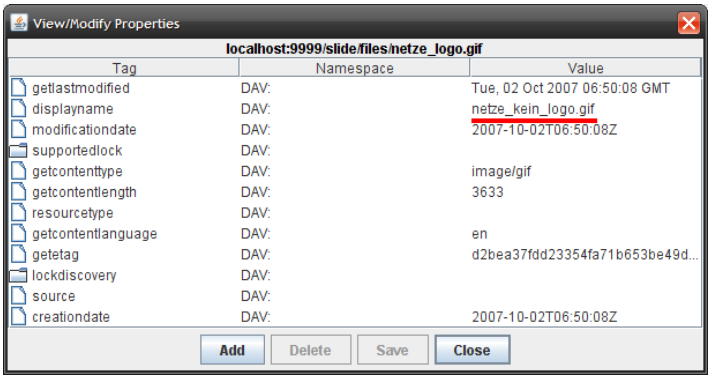


Bild 5.12: Ergebnis der Bearbeitung im DAV Explorer

**Projekt-Homepage bei SourceForge.net**

Im Rahmen dieser Arbeit wurde die durch SF gegebene Möglichkeit zur Einrichtung einer Projekt-Homepage genutzt. Die Gestaltung der Seiten wurde dabei von Mitarbeitern des DLR vorgenommen. Die Inhalte der Seiten wurden teilweise aus bereits vorhandenen Dokumenten entnommen.

Die Homepage erlaubt eine individuelle Präsentation des Projektes und die Bereitstellung von zusätzlichen Informationen für Anwender und Entwickler. Die Startseite ist in Bild 5.13 dargestellt.

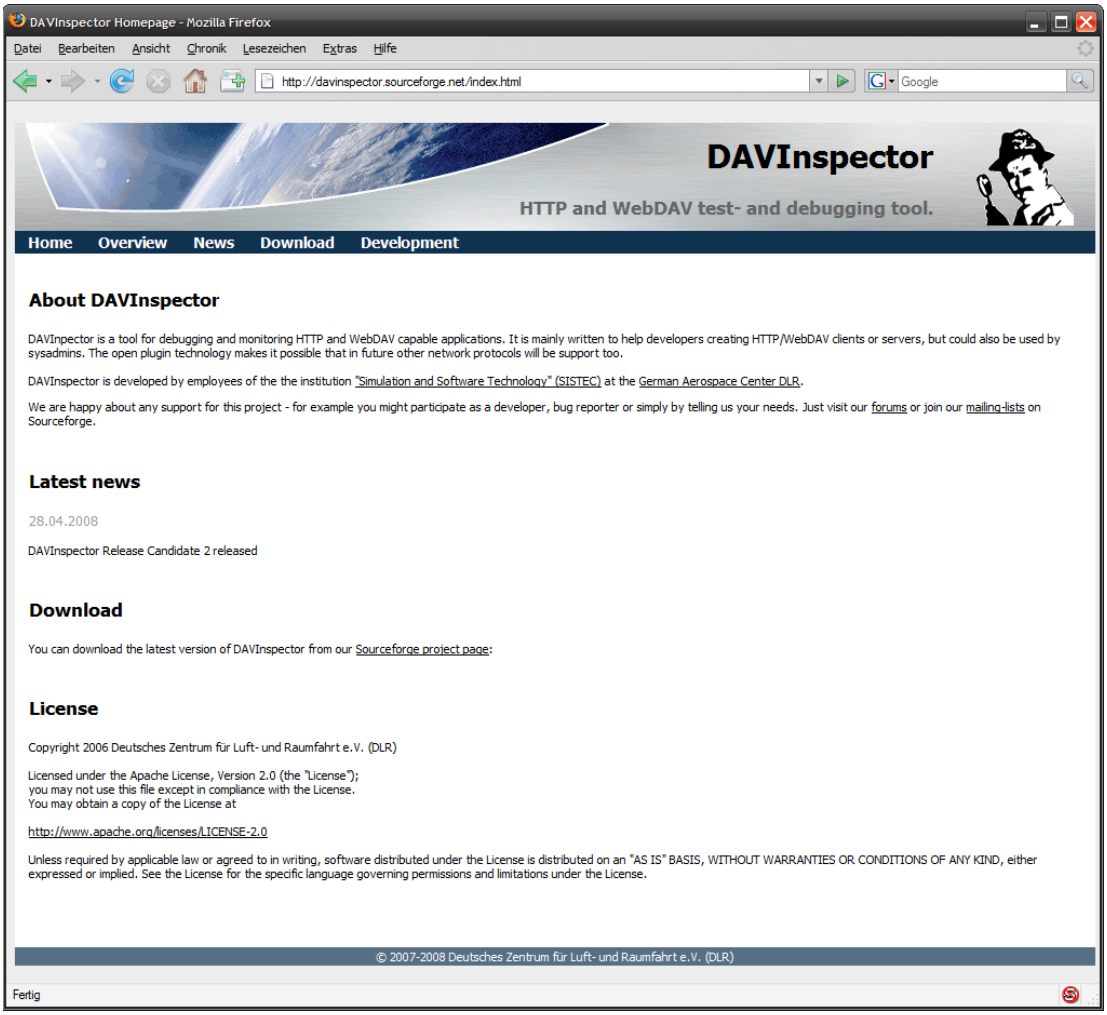


Bild 5.13: DAVInspector Projekt-Homepage bei SourceForge

Weiterhin gliedert sich das Projekt durch Verwendung der DLR-Vorlage einheitlich in die Darstellung von Open-Source-Projekten des DLR ein.

# 6 Zusammenfassung und Ausblick

Im Rahmen der Arbeit wurden ein Konzept und eine prototypische Implementierung eines grafischen Werkzeugs zum Testen und Auswerten von WebDAV-Anwendungen erarbeitet. Entsprechend den in der Aufgabenstellung gestellten Forderungen ermöglicht das Werkzeug eine Analyse der Kommunikation zwischen Client und Server. Den Entwicklern von WebDAV-Anwendungen im DLR steht damit ein Werkzeug zur Verfügung, das als Basis für eine effektive und anwenderfreundliche Testumgebung verwendet werden kann. Die erstellte Software unterstützt die Fehlersuche und -behebung in den vom DLR betreuten WebDAV-Produkten Catacomb und DataFinder und kann dadurch Entwicklungszeiten verkürzen.

Der Ablauf des Projektes gliederte sich in verschiedene Phasen. Zu Beginn der Arbeit wurde eine große Zeitspanne in die Einarbeitung und die Evaluierung verschiedener Lösungswege investiert. Dies schließt die Erstellung der Anforderungsdokumente und die Durchführung einer Marktanalyse ein. Eine ebenso große Zeitspanne wurde für die Implementierung und den Test der Applikation verwendet. Durch die guten Fortschritte bei der Realisierung war es zudem möglich, neben den geforderten Kriterien auch optionale Kriterien umzusetzen, wie etwa die Internationalisierung der Anwendung. Die verbleibende Zeit wurde für die Erstellung der Dokumentation verwendet.

Im Folgenden wird auf die Schwerpunkte der Arbeit gesondert eingegangen und die bei deren Erarbeitung erzielten Ergebnisse gesondert erläutert. Das Relais ist das zentrale Element der Software und kapselt die Kommunikation für die darauf aufbauenden Komponenten. Das Design wurde durch die Verwendung von Schnittstellen so ausgelegt, dass Änderungen und Erweiterungen einfach zu verwirklichen sind. So ist die Unterstützung

von verschlüsselten Verbindungen (HTTPS) eine mögliche Erweiterung dieser Komponente. Die Nachrichtenhistorie und die Nachrichtenerkennung sind zwei für die Datenverarbeitung notwendige Subsysteme. Bei der Erstellung dieser Komponenten wurde im Besonderen auf Erweiterbarkeit geachtet. Somit können zusätzliche Eigenschaften von Nachrichtentypen ergänzt werden und stünden damit für eine erweiterte Nachrichtenanalyse zur Verfügung. Da es sich bei der entwickelten Software um ein grafisches Werkzeug handelt, wurde der Realisierung der grafischen Oberfläche besondere Aufmerksamkeit gewidmet. Spezielles Augenmerk lag dabei auf einer intuitiven und übersichtlichen Bedienung der Anwendung. Hierfür wurden die Erkenntnisse aus der Marktanalyse verwendet und eigene Ideen in grafischen Elementen umgesetzt. Eine weitere Verbesserung der Oberfläche wird sich aus dem Einsatz der Software und den Rückmeldungen von Anwendern ergeben. Die realisierte Plugin-Architektur bildet die Grundlage für eine einfache Erweiterbarkeit der Funktionalität der Software. Zusätzliche Plugins, die zum Beispiel auf bestimmte Protokollvarianten des WebDAV-Protokolls spezialisiert sind (ACL, DASL, etc.), können zukünftig in das Werkzeug integriert werden. Verschiedene Plugins wurden prototypisch realisiert und erlauben die Analyse der Nachrichten oder von Teilen einer Nachricht. Weiterhin wurde ein Konzept ausgearbeitet, welches Plugins erlaubt den Datenverkehr zu manipulieren. Auch dieses Konzept wurde prototypisch in einem Plugin verwirklicht und getestet.

Die durch das Programm zur Verfügung gestellte Funktionalität ermöglicht den Entwicklern von WebDAV-Clients und Servern die Auswertung und Analyse von Kommunikation. Ferner ist eine Bearbeitung der Nachrichten anhand eines implementierten Plugins möglich. Die Abfolge des Nachrichtenaustausches kann durch die Nachrichtenhistorie nachvollzogen werden und erlaubt jederzeit den Zugriff auf bereits versendete Nachrichten. So ist auch eine nachträgliche Auswertung der Kommunikation möglich. Des Weiteren sind verschiedene Möglichkeiten vorgesehen, die einen Export der aufgezeichneten Nachrichten erlauben. Dadurch können auch externe Programme für eine weitere Analyse herangezogen werden. Im April 2008 wurden zwei „Release Candidates“ (RC) der Software auf der Projekt-Homepage anderen Anwendern zur Verfügung gestellt und seitdem 39-mal

heruntergeladen. Insgesamt konnte das Projekt bisher knapp 1300 Zugriffe auf die Homepage verzeichnen. Maßnahmen zur weiteren Verbreitung des Projektes sind in Planung. Die bisher nicht implementierten Funktionen, wie etwa die Emulation eines Clients oder Servers und die bereits aufgeführte Unterstützung von verschlüsselten Verbindungen sind zu empfehlende Ergänzungen. Die entsprechenden Schnittstellen wurden im Prototypen bereits vorgesehen.

# Literaturverzeichnis


[AGG03] Alkassar, A.; Garschhammer, M.; Gehring, F.; et al.:
„Kommunikations- und Informationstechnik 2010+3: Neue Trends und Entwicklungen in Technologien, Anwendungen und Sicherheit“
Bundesamt für Sicherheit in der Informationstechnik (BSI)
SecuMedia Verlag, Ingelheim 2003

[ALE77] Alexander, Christopher; et al.:
A Pattern Language
Oxford University Press, New York 1977

[APA07] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html
*15. Oktober 2007*

[ASF08] The Apache Software Foundation:
Apache License v2.0 and GPL Compatibility
www.apache.org/licenses/GPL-compatibility.html
*21. April 2008*

[BAL98] Balzert, Helmut:
Lehrbuch der Software-Technik, Band 2:
Softwaremanagement, Unternehmensmodellierung, Software-Qualitätssicherung
Spektrum Akademischer Verlag, Heidelberg 1998

[BEC00] Beck, K.:
Extreme Programming: Die revolutionäre Methode für Softwareentwicklung in kleinen Teams
Addison-Wesley, München 2000

[BOE88] Boehm, B.W.:
A spiral model of software developement and enhancement
IEEE Computer 21(5), 1988

[BRD04] Brügge, Bernd; Dutoit, Allen H.:
Objektorientierte Softwaretechnik
Pearson Studium, München 2004

[BSD08] BSD License
http:
//www.opensource.org/licenses/bsd-license.php
*21. April 2008*

[BUH04] Buhl, Axel:
Grundkurs Software-Projektmanagement
Carl Hanser Verlag, München 2004

[CCJ08] SUN: Code Conventions for the Java Programming Language
http://java.sun.com/docs/codeconv/index.html
*21. April 2008*

[COM95] Comer, Douglas E.:
Internetworking with TCP/IP, Volume I; 3. Ausgabe
Prentice Hall, New Jersey 1995

[DAS08] DAV Searching And Locating – DASL
http://www.webdav.org/dasl/
*14. Mai 2008*

[DUS04] Dusseault, Lisa:
WebDAV – next-generation collaborative Web authoring
Prentice Hall Professional Technical Reference, New York
2004

[FLA01] Flanagan, David:
Java in a Nutshell, 3. Auflage
O'Reilly Verlag, Köln 2001

[FRE04] Freeman, Eric; Freeman, Elisabeth:
Head First Design Patterns
O'Reilly Media, Inc., Sebastopol 2004

[FSF08] The Free Software Foundation
http://www.fsf.org/about/
*21. April 2008*

[GAM96] Gamma, Erich; Helm, Richard; Johnson, Ralph; John Vlissides:
Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software
Addison-Wesley, Bonn 1996

[GCP08] GNU – Copyleft
http://www.gnu.org/copyleft/copyleft.de.html
*21. April 2008*

[GFS08] GNU – The Free Software Definition
http://www.gnu.org/philosophy/free-sw.html
*21. April 2008*

[GOL99] Goldfarb, Charles F.; Prescod, Paul:
XML-Handbuch
Prentice Hall, München 1999

[GOT02] Gourley, David; Totty, Brian; Sayer, M.; Reddy, S.; Aggarwal, A.:
HTTP – The Definitive Guide
O'Reilly & Associates, Sebastopol 2002

[GPL3] GNU General Public License version 3
http://www.opensource.org/licenses/gpl-3.0.html
*21. April 2008*

[GRA04] Grassmuck, Volker:
Freie Software – Zwischen Privat- und Gemeineigentum
Bundeszentrale für politische Bildung (bpb), Bonn 2004

[GWO06] Gwosdz, Frank:
Diplomarbeit – Einsatz von Open Source zur Erstellung von Software
Fachhochschule München, München 2006

[HAR05] Harold, Elliotte Rusty:
Java Network Programming, 3. Ausgabe
O'Reilly Media, Inc., Sebastopol 2005

[IAN08] IANA – PORT NUMBERS
http://www.iana.org/assignments/port-numbers
*24. April 2008*

[ISO7498] ISO/IEC 7498
Information Processing Systems - Open Systems Interconnection: Basic Reference Model
International Organization for Standardization (ISO) and International Electrotechnical Committee (IEC), Genf 1996

[ISO9126] ISO/IEC 9126
Software engineering – Product quality
International Organization for Standardization (ISO) and International Electrotechnical Committee (IEC), Genf 2001

[JOB02] Jobst, Fritz:
Programmieren in Java
Carl Hanser Verlag, München 2002

[KER90] Kernighan, Brian W.; Ritchie, Dennis M.:
Programmieren in C
Hanser Fachbuch 1990

[KLO07] Klose, Bernd:
Scriptum zur Vorlesung „Objektorientierter Systementwurf"
Universität Siegen, 2007

[KRY02] Kredel, Heinz; Yoshida, Akitoshi:
Thread- und Netzwerk-Programmierung mit Java
dpunkt.verlag GmbH, Heidelberg 2002

[LGPL3] GNU Lesser General Public License
http://www.opensource.org/licenses/lgpl-3.0.html
*21. April 2008*

[LOY03] Loy, Mark; Eckstein, Robert; Wood, David; Elliot, James; Cole, Brian:
Java Swing; 2. Auflage
O'Reilly Media, Sebastopol 2003

[MUE99] Müller-Ettrich, Gunter:
Objektorientierte Prozeßmodelle
Addison-Wesley Longman, Bonn 1999

[NUE00] Nüttgens, M.; Tesei, E. :
Open Source: Produktion, Organisation und Lizenzen
Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 157
Universität des Saarlandes, Saarbrücken 2000

[OAW04] Oaks, Scott; Wong, Henry:
PJava Threads, 3. Ausgabe
O'Reilly Media, Inc., Sebastopol 2004

[OEC01] Oechsle, Rainer:
Parallele Programmierung mit Java Threads
Fachbuchverlag Leipzig, München 2001

[OSD08] OSI – Open Source Initiative
The Open Source Definition
http://www.opensource.org/docs/osd
*21. April 2008*

[OSI08] OSI – Open Source Initiative
About the Open Source Initiative
http://www.opensource.org/about
*21. April 2008*

[OSL08] OSI – Open Source Initiative
Open Source Licenses – Licenses by Name
http://www.opensource.org/licenses/alphabetical
*21. April 2008*

[PRE87] Pressman, Roger S.:
Software Engineering – A Practitioner's Approach; 2. Auflage
McGraw-Hill Book Co., Singapore 1987

[QTJ08] Qt Jambi Rich Client Java Development Framework
http://trolltech.com/products/qt/jambi
*30. April 2008*

[QUI99] Quibeldey-Cirkel, Klaus:
Entwurfsmuster: Design Patterns in der objektorientierten Softwaretechnik
Springer-Verlag, Berlin 1999

[RFC791] RFC 791 – Internet Protocol (IP)
http://www.ietf.org/rfc/rfc0791.txt
*24. April 2008*

[RFC793] RFC 793 – Transmission Control Protocol (TCP)
http://www.ietf.org/rfc/rfc0793.txt
*24. April 2008*

[RFC959] RFC 959 – FILE TRANSFER PROTOCOL (FTP)
http://www.ietf.org/rfc/rfc959.txt
*27. Mai 2008*

[RFC1122] RFC 1122 – Requirements for Internet Hosts – Communication Layers
http://www.ietf.org/rfc/rfc1122.txt
*27. Mai 2008*

[RFC1945] RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0
http://www.ietf.org/rfc/rfc1945.txt
*25. April 2008*

[RFC2616] RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt
*25. April 2008*

[RFC3253] RFC 3253 – Versioning Extensions to WebDAV
http://www.ietf.org/rfc/rfc3253.txt
*29. April 2008*

[RFC3648] RFC 3648 – Ordered Collections Protocol
http://www.ietf.org/rfc/rfc3648.txt
*29. April 2008*

[RFC3744] RFC 3744 – Access Control Protocol
http://www.ietf.org/rfc/rfc3744.txt
*29. April 2008*

[RFC4316] RFC 4316 – Datatypes for Web Distributed Authoring and Versioning
http://www.ietf.org/rfc/rfc4316.txt
*29. April 2008*

[RFC4331] RFC 4331 – Quota and Size Properties
http://www.ietf.org/rfc/rfc4331.txt
*29. April 2008*

[RFC4437] RFC 4437 – Redirect Reference Resources
http://www.ietf.org/rfc/rfc4437.txt
*29. April 2008*

[RFC4709] RFC 4709 – Mounting Web Distributed Authoring and Versioning (WebDAV) Servers
http://www.ietf.org/rfc/rfc4709.txt
*29. April 2008*

[RFC4791] RFC 4791 – Calendaring Extensions to WebDAV (CalDAV)
http://www.ietf.org/rfc/rfc4791.txt
*29. April 2008*

[RFC4918] Dusseault, Lisa, Reschke, Julian; Sinderson, Elias; Whitehead, Jim:
RFC 4918 – HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)
http://www.ietf.org/rfc/rfc4918.txt
*15. Oktober 2007*

[ROY70] Royce, Winston:
Managing the Development of Large Software Systems
Proceedings of IEEE WESCON, IEEE Computer Society Press, Los Alamitos 1970

[SDT08] Java SE Desktop Technologies
http://java.sun.com/javase/technologies/desktop/techoverview.jsp
*30. April 2008*

[SFN08] What is SourceForge.net?
http://alexandria.wiki.sourceforge.net/What+is+SourceForge.net%3F
*22. April 2008*

[SOM01] Sommerville, Ian:
Software Engineering, 6. Auflage
Pearson Studium, München 2001

[STE94] Stevens, W. Richard:
TCP/IP Illustrated: the protocols, Volume I
Addison-Wesley, New York 1994

[SWT08] SWT: The Standard Widget Toolkit
http://www.eclipse.org/swt/
*30. April 2008*

[TAN02] Tanenbaum, Andrew S.:
Moderne Betriebssysteme, 2. Auflage
Pearson Studium, München 2002

[TAN03] Tanenbaum, Andrew S.:
Computernetzwerke, 4. Auflage
Pearson Studium, München 2003

[VON07] Vonhoegen, Helmut:
Einstieg in XML – Grundlagen, Praxis, Referenzen; 4. Auflage
Galileo Press, Bonn 2007

[WDA08] WebDAV Resources
http://webdav.org/
*29. April 2008*

[WDP08] WebDAV Resources – WebDAV Projects
http://webdav.org/projects/
*29. April 2008*

[XML08] Extensible Markup Language (XML):
http://www.w3.org/XML/
*22. April 2008*

# Glossar

**ACL** Engl. Abkürzung für Access Control List. ACL's sind Listen, die Rechte von Benutzern auf Objekte speichern.

**Bug-Tracker** Software zur Verfolgung und Dokumentation von Fehlern in Software.

**Catacomb** Catacomb ist eine Erweiterung für das WebDAV-Modul des Apache Webservers. Die Daten werden hierbei statt im Dateisystem in einer relationalen Datenbank abgelegt.

Siehe auch: http://catacomb.tigris.org/

**CERN** Die Europäische Organisation für Kernforschung (franz. Conseil Européen pour la Recherche Nucléaire – CERN) ist eine Großforschungseinrichtung in der Nähe von Genf in der Schweiz.

Siehe auch: http://www.cern.de/

**Checkstyle** Checkstyle ist ein Werkzeug für Entwickler mit dem man Quelltext nach formalen Kriterien überprüfen kann.

Siehe auch:
http://eclipse-cs.sourceforge.net/

**CVS** Engl. Concurrent Versions System (CVS). CVS ist ein Versionsverwaltungssystem für Quellcode.

Siehe auch: http://www.nongnu.org/cvs/

**DASL** Engl. Abkürzung für DAV Searching and Locating. DASL befasst sich mit der serverseitigen Suche in WebDAV-Repositories. Die Arbeitsgruppe zu diesem Standard wurde allerdings mittlerweile aufgelöst.

**DataFinder** DataFinder ist eine Applikation auf Clientseite zur Verwaltung technisch-wissenschaftlicher Daten. Hierbei können die Daten sowie die Metadaten über verschiedenartige Speicherschnittstellen auf einem oder mehreren Servern abgelegt werden.

Siehe auch:
http://www.dlr.de/sc/desktopdefault.aspx/tabid-1189/1650_read-3140/

**DeltaV** Engl. Web Versioning and Configuration Management. Dieses Protokoll ist eine Erweiterung des WebDAV-Protokolls um Versions- und Konfigurationsmanagement.

**DLR** Deutsches Zentrum für Luft- und Raumfahrt.

Siehe auch: http://www.dlr.de

**DNS** Engl. Domain Name System (DNS). Aufgabe des DNS ist die Namensauflösung, das heißt die Umsetzung von Rechnernamen in IP-Adressen.

Siehe auch:
http://tools.ietf.org/html/rfc1034

**DoD**

Engl. United States Department of Defense (DoD) – Verteidigungsministerium der Vereinigten Staaten von Amerika

Siehe auch: http://www.defenselink.mil/

**DTD**

Eine Dokumenttypdefinition (englisch Document Type Definition, DTD) ist ein Satz von Regeln, der Dokumente eines bestimmten Typs repräsentiert und deren Struktur festlegt.

**Eclipse**

Eclipse ist ein Framework zur Entwicklung von Anwendungen und ist vergleichbar mit einer IDE, kann aber auch noch wesentlich mehr bieten, da es durch eine entsprechende Architektur einfach erweiterbar ist.

**FTP**

Engl. File Transfer Protocol (FTP). FTP ist ein Netzwerkprotokoll zur Dateiübertragung über TCP/IP-Netzwerke.

Siehe auch: http://tools.ietf.org/html/rfc959

**HTML**

Engl. Hypertext Markup Language (HTML)
Siehe Kapitel 2.4

Siehe auch: http://www.w3.org/TR/html4/

**HTTP**

Engl. Hypertext Transfer Protocol (HTTP)
Siehe Kapitel 2.4

Siehe auch:
http://tools.ietf.org/html/rfc2616

**IANA** Engl. Internet Assigned Numbers Authority (IANA). Die IANA ist eine Organisation, die die Vergabe von IP-Adressen, Top Level Domains und IP-Protokollnummern sowie die Vergabe der Portnummern von 0 bis 1023 regelt.

Siehe auch: http://www.iana.org/

**IDE** Engl. Abkürzung für Intergrated Developement Environment. Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software, das mehrere Komponenten in einer Oberfläche zusammenfasst.

**IETF** Engl. Internet Engineering Task Force (IETF). Die IETF ist eine offene und internationale Organisation, die sich mit der technischen Weiterentwicklung des Internets befasst.

Siehe auch: http://www.ietf.org/

**ISO** Internationale Organisation für Normung – ISO (von gr.: „isos“ – „gleich“). Die ISO ist eine internationale Vereinigung von Normungsorganisationen und erarbeitet internationale Normen in fast allen Bereichen.

Siehe auch: http://www.iso.org/iso/home.htm

**Loopback** Eine Loopback Netzwerkschnittstelle ist ein lokaler Informationskanal mit nur einem Endpunkt. Sender und Empfänger sind hierbei identisch.

**Mailingliste** Eine Mailingliste ist ein E-Mail-Verteiler für eine Gruppe von Menschen. Einzelne Mailinglisten haben häufig bestimmte Themen und dienen zur Kommunikation und Information.

**Makro** Ein Makro ist in der Programmierung eine vorgegebene Sequenz von Befehlen oder Aktionen. In diesem Zusammenhang bezeichnet ein Makro eine aufgezeichnete Kommunikationssequenz zwischen zwei WebDAV-Applikationen.

**MVC** Engl. Model-View-Controller. MVC ist ein zusammengesetztes Architekturmuster zur Gestaltung von Anwendungen. Um eine bessere Wiederverwendbarkeit und einfachere Erweiterbarkeit zu erhalten erfolgt eine Aufteilung in die Elemente Datenmodell (Model), Präsentation (View) und Steuerung (Controller).

**Open Source** Open Source Software ist quelloffene Software. Das heißt, der Quellcode dieser Software ist frei zugänglich und deren Bearbeitung und Verbreitung erlaubt und erwünscht. Es gibt verschiedene Lizenzmodelle, die eine genaue Definition der Verwendung und Verbreitung der jeweiligen Software regeln.

**Plugin** Ein Plugin ist ein Software- oder Hardwaremodul, welches in eine bestehende Software oder Hardware eingebunden wird und die Funktionalität der Software oder Hardware erweitert.

**Proxy** Ein Proxy ist im Zusammenhang mit Netzwerken ein Dienstprogramm zur Steuerung von Netzwerkverkehr. Ein Proxy besitzt sowohl eine ServerSchnittstelle als auch eine Client-Schnittstelle. Im einfachsten Falle leitet der Proxy den Netzwerkverkehr unverändert weiter. In diesem Fall spricht man auch von einem „Relais“.

**RFC** Engl. Request for Comments. Die RFCs sind eine Sammlung von technischen und organisatorischen Dokumenten zu den im Internet verwendeten Protokollen und Verfahren.

**SC** Abkürzung für die Abteilung Simulations- und Softwaretechnik innerhalb des DLR.

Siehe auch: http://www.dlr.de/sc/

**SCM** Engl. Software Configuration Management (SCM). Softwarekonfigurationsmanagement beschäftigt sich mit allen Aktivitäten, die im Bereich der Software-Entwicklung anzutreffen sind.

**SGML** Engl. Standard Generalized Markup Language (SGML) ist eine Metasprache. Auf Basis von SGML wurden verschiedene Auszeichnungssprachen definiert.

Siehe auch: http://www.w3.org/MarkUp/SGML/

**sourceforge.net** sourceforge.net ist eine Online-Plattform. Hier werden einem oder einer Gruppe von Entwicklern Werkzeuge zur Unterstützung bei der Anwendungsentwicklung und zur Projektverwaltung geboten.

Siehe auch: http://www.sourceforge.net/

**Subclipse** Eclipse-Plugin für die Versionsverwaltung Subversion.

**Subversion** Open Source Software für die Versionsverwaltung von Dateien und Ordnern.

Siehe auch: http://subversion.tigris.org/

**Tag** Ein Tag (engl. für Etikett, Anhänger, Aufkleber, Marke, Auszeichner) ist eine Auszeichnung eines Datums mit zusätzlichen Informationen.

**TCP** Engl. Abkürzung für Transmission Control Protocol. TCP ist ein verbindungsorientiertes Transportprotokoll der OSI-Schicht 4.
Siehe Kapitel 2.4

**UML** Engl. Abkürzung für Unified Modeling Language. Eine von der Object Management Group entwickelte Sprache zur Modellierung von Software.

**Unit-Test** Der Unit-Test wird auch als Modultest bezeichnet und dient zur Überprüfung der Korrektheit von Programmteilen.

**URI** Als Uniform Resource Identifier (URI) bezeichnet man einen Identifikator, der aus einer Zeichenfolge, die zur Identifizierung einer abstrakten oder physischen Ressource dient, besteht.

Siehe auch:
http://tools.ietf.org/html/rfc3986

**URL** Ein Uniform Resource Locator (URL) ist eine Unterart des URIs. URLs identifizieren eine Ressource über das verwendete Netzwerkprotokoll und den Ort der Ressource in Computernetzwerken. Beispiel: `http://www.uni-siegen.de/uni/universitaet/`

Siehe auch:
http://tools.ietf.org/html/rfc3986

**W3C** Das World Wide Web Consortium (W3C) ist ein internationales Konsortium, in dem verschiedene Mitglieder gemeinsam Web-Standards und Richtlinien entwickeln.

Siehe auch: `http://www.w3.org/`

**WebDAV** Engl. Abkürzung für Web-based Distributed Authoring and Versioning. WebDAV ist ein offener Standard zur Bereitstellung von Dateien im Internet und ist eine Weiterentwicklung des HTTP-Protokolls.
Siehe Kapitel 2.4

Siehe auch: `http://www.webdav.org/`

**Wiki** Ein Wiki ist eine Sammlung von Hypertextseiten, die von Benutzern gelesen, editiert und verknüpft werden können.

**XML** Engl. Abkürzung für Extensible Markup Language.
Siehe Kapitel 2.6

# CD-ROM

# A Softwarekonfigurationsmanagement

Wie bereits in Kapitel 2.2 und Kapitel 3.4 erläutert wird für die Versionsverwaltung die Software „Subversion“ genutzt. Die Daten können innerhalb von „Subversion“ in verschiedenen Ordnern organisiert werden. Die oberste Ebene wird dabei von den Verzeichnissen „trunk“, „tags“ und „branches“ gebildet. Das Verzeichnis „trunk“ ist das Basisverzeichnis und beinhaltet den Hauptzweig der Entwicklung. Zusätzlich werden dort bei diesem Projekt der Inhalt der Projekt-Homepage und Teile der Dokumentation abgelegt. Im Verzeichnis „branches“ können alternative Entwicklungszweige hinterlegt werden. Im Zuge der Entwicklung der Prototypen wurde dieses Verzeichnis auch für die Wegwerf-Prototypen verwendet. Das Verzeichnis „tags“ enthält die für eine Version der Software gültigen Revisionen der Quelldateien. Diese werden danach auch nicht mehr verändert und ermöglichen so jederzeit die Wiederherstellung einer Version.

Im Folgenden wird die Erstellung einer Version des Prototypen beschrieben. Eine solche Version wird auch als „Release“ bezeichnet und das nun beschriebene Vorgehen im allgemeinen auch als „Release-Management“. Ein Release der Software besteht dabei mindestens aus folgenden Teilen:

- **Anwendung**
  Die Applikation selbst bildet natürlich den wichtigsten Bestandteil eines Release.

- **Benutzerdokumentation**
  Die Benutzerdokumentation besteht aus Installations- und Betriebsanweisungen. Weiterhin können Schritt-für-Schritt-Anleitungen, Hinweise und sonstige Dokumente enthalten sein.

- **CHANGES-Datei**
  Diese Datei enthält alle seit der letzten Version der Software durchgeführten Änderungen und Neuerungen.
- **RELEASE-Datei**
  Die RELEASE-Datei enthält speziell Informationen zu diesem Release.
- **Lizenz**
  Bei Open-Source-Projekten ist es üblich, eine Kopie der verwendeten Lizenz mit auszuliefern.

Die Benennung einer Version richtet sich dabei nach dem folgenden Schema: `<Haupt-Nr>.<Neben-Nr>[.<Bugfix-Nr>][-Release-Typ]`
Die einzelnen Nummern des Schemas werden dabei bei einer neuen Version in der Regel um eins erhöht. Es existieren genaue Regeln, die festlegen welche Versionsnummer bei einer Änderung erhöht werden sollte. So wird die Hauptnummer nur bei großen Änderungen, beispielsweise dem Austausch von Kernmodulen oder dem Einsatz einer neuen grafischen Oberfläche, erhöht. Die Nebennummer dient zur Kennzeichnung und Unterscheidung des Entwicklungszweiges und des stabilen Zweiges der Software. Eine Version mit gerader Nebennummer entspricht somit einer stabilen Version der Software und eine Version mit ungerader Nebennummer einer in der Entwicklung befindlichen Version. Die Bugfix-Nummer wird nur für sicherheitskritische Nachbesserung der Software benutzt. Der Release-Typ kann zur weiteren Kennzeichnung der Version genutzt werden. Typische Beispiele für Bezeichnungen von Release-Typen sind „Alpha“, „Beta“ oder „RC – Release Candidate“.

Für die Erstellung einer Version sind folgende Schritte durchzuführen:

1. Die Versionsnummern müssen entsprechend den oben aufgeführten Regeln angepasst werden.
2. Die Quelldateien werden „ge-tagt“, also in ihren jeweiligen Versionen in Subversion als zu dieser Version gehörend gekennzeichnet.
3. Die Version wird erstellt und kontrolliert.

4. Die CHANGES- und RELEASE-Dateien werden erstellt.
5. Die Version wird veröffentlicht und eventuell durch weitere Maßnahmen bekannt gemacht.

Damit sind alle Schritte für eine einheitliche Versionsverwaltung beschrieben und die verschiedenen Versionsstände der Software können jederzeit nachvollzogen werden.

# B Programmierrichtlinien

Die einheitliche Gestaltung der Quelltexte und Kommentare ermöglicht Entwicklern eine schnelle Einarbeitung und sorgt im Allgemeinen für eine bessere Lesbarkeit, Verständlichkeit und Wartbarkeit. Somit führt die Einhaltung der Programmierrichtlinien zu einer besseren Quelltextqualität und damit letztendlich zu einer gesteigerten Qualität der Applikation. Im Folgenden werden die für dieses Projekt gültigen Konventionen in Ausschnitten vorgestellt.

Programmierrichtlinien beinhalten eine Vielzahl von Formatierungsvorgaben und sonstigen Regeln:

- Namenskonventionen
- Positionierung von Syntaxelementen
- Einrückungs- und Verschachtelungstiefe von Elementen
- Aufbau und Einsatz von Kommentaren

Diese Liste ist nicht vollständig. Eine umfassende Betrachtung dieses Themas würde den Rahmen dieser Arbeit sprengen, und es soll an dieser Stelle auf entsprechende Literatur [FLA01] und Regelwerke [CCJ08] verwiesen werden. Die Definition eines solchen Regelwerks wird auch als „Coding-Standard“ bezeichnet.

Als Basis für dieses Projekt wurden die Vorgaben eines anderen Projektes im DLR[1] verwendet. Beide Projekte werden in der Programmiersprache Java verwirklicht, und deshalb ist hier eine Übertragung des Regelwerks möglich. Nachstehend werden einige Auszüge aus den Programmierrichtlinien dargestellt und erläutert.

[1] Vgl. DLR RCE-Programmierrichtlinien 2007

### Positionierung und Einrückung

Bei der Positionierung und Einrückung der Syntaxelemente wird der von den C-Erfindern Brian W. Kernighan und Dennis Ritchie [KER90] etablierte Stil für Java adaptiert. Dieser Stil wird auch als „One True Brace Style“ bezeichnet und bietet eine kompakte und übersichtliche Schreibweise.

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          if (args.length > 0) {
5              for (String arg : args) {
6                  System.out.println("Hello, " + arg + "!");
7              }
8          } else {
9              System.out.println("Hello, world!");
10         }
11     }
12 }
```

Listing B.1: Positionierung und Einrückung

In Listing B.1 ist ein einfaches Beispiel dargestellt. Es werden für jeden Einrückschritt zwei Leerzeichen verwendet. Die kompakte Schreibweise ist, zum Beispiel, gut in Zeile 8 zu sehen. Dabei befinden sich sowohl schließende als auch öffnende Klammer in einer Zeile.

### Kommentare

Kommentare sind ein wichtiger Bestandteil von Quelltexten. Eine einheitliche Gestaltung fördert die Übersichtlichkeit und ermöglicht bei Verwendung von zusätzlichen Werkzeugen[2] die automatische Generierung von Dokumentation.

```
1  /**
2   * This class represents a single message.
3   *
4   * @version $LastChangedRevision$
5   * @author Jochen Wuest
6   */
7  public class Message {
```

[2] Vgl. SUN Javadoc Tool: http://java.sun.com/j2se/javadoc/

```
8
9     /**
10     * Determines whether the data is chunked or not.
11     *
12     * @param data      String. The data to send.
13     * @param encoding  String. Encoding of the data.
14     * @return          Boolean. Returns true if data is chunked.
15     */
16    private Boolean isChunked(String data, String encoding) {
17      [...]
18    }
19
20    // Hier fehlt etwas! Inline-Kommentar
21    [...]
22  }
```

Listing B.2: Kommentare

Listing B.2 zeigt Beispiele für Kommentare von Klassen (Zeilen 1-6) und Methoden (Zeilen 9-15) sowie einzeilige Kommentare (Zeile 20). Weiterhin enthält jede Quelldatei noch einen definierten Kommentar zu Beginn. Dort werden Dateiname, Autor, Copyright und Lizenz ausgewiesen. Einzeilige Kommentare werden zur allgemeinen Dokumentation verwendet. Kommentare von Klassen enthalten mindestens eine Beschreibung der Klasse und den Namen des Autors. Bei Methoden wird im jeweiligen Kommentar zusätzlich zur Beschreibung der Methode die Signatur der Methode hinterlegt.

**Namenskonventionen**

Ein weiterer wichtiger Bereich der Richtlinien betrifft die Benennung von Klassen, Methoden und Variablen. Ein einheitliches Vorgehen in diesem Bereich führt ebenfalls zu einer verbesserten Codequalität. Nachfolgende Regeln werden für dieses Projekt angewendet:

- Klassennamen: Verwendung von Binnenversalien, auch als „Camel Case“ bezeichnet, mit einem Großbuchstaben zu Beginn.
- Methodennamen, Paramter, Variablen: Verwendung von Binnenversalien mit einem Kleinbuchstaben zu Beginn.

- Konstanten: Verwendung von Großbuchstaben. Zur Trennung von Wörtern wird ein „_“ verwendet.
- Schnittstellen und abstrakte Klassen: Hier wird zur besseren Unterscheidung von „normalen“ Klassen den Bezeichnern ein „`I`“ oder „`A`“ vorangestellt.
- Der Bezeichner für ein Plugin besteht aus einem Namen und „`plugin`“.

**Sonstiges**
Des Weiteren werden verschiedene weitere Regeln definiert, die die Verwendung von Schlüsselwörtern einschränken (Zum Beispiel keine Verwendung von `System.out.println()`.) oder in irgendeiner anderen Weise begrenzend wirken. Zum Beispiel maximal dreimalige Verwendung von `return` innerhalb einer Methode oder beschränkte Zeilenlänge.

Das vollständige Regelwerk befindet sich zusammen mit dem Quellcode in der Versionsverwaltung. Bei der Um- und Durchsetzung dieses Programmierstils wird der Entwickler durch Werkzeuge, siehe Kapitel 3.4, unterstützt.

# C Lastenheft


Elektrotechnik und Informatik

Diplomarbeit

# Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen

Lastenheft

Revision 1.1

# Inhaltsverzeichnis

# 1 Einleitung

Das deutsche Zentrum für Luft- und Raumfahrt ist eine Forschungseinrichtung der Bundesrepublik Deutschland mit den Schwerpunkten Luftfahrt, Raumfahrt, Energie und Verkehr. Das Spektrum der Forschungen reicht von der Grundlagenforschung bis hin zu fertigen Produkten und Anwendungen. Hierbei werden sowohl nationale, internationale als auch industrielle Kooperationen umgesetzt.

Wissenschaftliche Einrichtungen müssen in der Regel eine große Menge von Daten verwalten. Hierbei sind üblicherweise sowohl die Menge der Daten als auch die vielen verschiedenen Formate und Prozesszugehörigkeiten der Daten problematisch. Zu einem Experiment gehören zum Beispiel Dokumentationen der verschiedenen Arbeitsschritte, Eingabe- oder Messdaten, Modelle sowie Ergebnisdaten. Zusätzlich können die Daten noch in verschiedenen Versionen vorliegen. Für das Management dieser Daten wird im DLR auf Clientseite die Anwendung DataFinder auf Grundlage des WebDAV-Protokolls eingesetzt. Als Server kann dafür jede Server-Software eingesetzt werden, welche die WebDAV-Spezifikation [RFC4918] hinreichend implementiert. Neben kommerziellen Servern (z. B. Tamino der Software AG oder SharePoint von Microsoft) werden zunehmend Open Source Produkte eingesetzt. Im DLR wird dafür am Open Source WebDAV-Server Catacomb mitentwickelt. Für das Entwickeln des Servers und des Clients sind genaue und automatisierte Tests eine wichtige Hilfe.

Im Rahmen der Diplomarbeit soll ein Open Source Werkzeug realisiert werden, mit dessen Hilfe WebDAV-Anwendungen grafikunterstützt getestet und ausgewertet werden können. Dabei soll das Werkzeug als Proxy zwischen Client und Server fungieren und somit den Datenverkehr auswerten, um eine Analyse der Kommunikation zwischen Server und Client zu

ermöglichen. Weiterhin sollen verschiedene Berichte generiert sowie vorher geplante Tests durchgeführt werden können.

Die gestellte Aufgabe ist innerhalb einer sechsmonatigen Diplomarbeit im Rahmen des Studiums „Angewandte Informatik“ mit Anwendungsfach Elektrotechnik an der Universität Siegen zu bearbeiten. Betreut wird die Arbeit auf universitärer Seite durch die Fachgruppe Technische Informatik, geleitet von Uni.-Prof. Hans Wojtkowiak. Ansprechpartner ist Dr.-Ing. Bernd Klose. Ansprechpartner in der Einrichtung „SC – Verteilte Systeme und Komponentensoftware“ des DLR sind Abteilungsleiter Andreas Schreiber und Dipl.-Inform. Markus Litz.

# 2 Produkteinsatz

## 2.1 Anwendungsbereiche

Für die Entwicklung WebDAV-fähiger Softwareprodukte ist es wichtig mit wenig Aufwand die erstellten Produkte zu testen um Fehler einfach finden und gegebenenfalls reproduzieren zu können. Das zu entwickelnde Werkzeug soll genau diese Möglichkeiten bieten und von Entwicklern und Testern dazu eingesetzt werden, die Kommunikation eines Clients und Servers über das WebDAV-Protokoll zu speichern, auszuwerten und gegebenenfalls wiederholen zu können.

Weiterhin soll die Software in der Lage sein die Gegenseite des jeweiligen Kommunikationspartners durch manuell zu erstellende oder abgespeicherte Kommunikationssequenzen (im Folgenden als Makros bezeichnet) zu emulieren. Hierbei soll zunächst der Schwerpunkt bei der Emulation des Clients liegen.

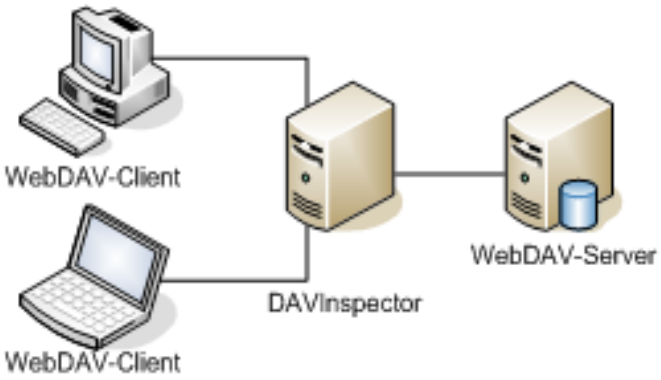


Bild 2.1: Einsatzszenario des DAVInspectors

In Bild 2.1 ist die Einsatzumgebung des zu erstellenden Werkzeugs mit dem Namen DAVInspector dargestellt. Die Kommunikation zwischen den

WebDAV-Clients und dem WebDAV-Server wird hierbei über den DAVInspector geleitet und kann dort nach Wunsch manipuliert und ausgewertet werden. Hierbei handelt es sich allerdings nur um eine schematische Darstellung. In der Praxis können die Komponenten alle auf einem physikalischen Rechner installiert sein oder aber auch auf zwei oder drei Rechner verteilt sein.

## 2.2 Zielgruppe

Zielgruppe für die Anwendung sind WebDAV-Applikationsentwickler. Dies umfasst sowohl Entwickler von Client-, als auch Serverapplikationen. Das Produkt ist nicht für den Einsatz in produktiven Umgebungen gedacht, sondern ist ein Werkzeug für Entwickler. Es soll das Testen und die Suche, sowie die Nachvollziehbarkeit von Fehlern vereinfachen.

# 3 Produktfunktionen

## 3.1 Musskriterien

Die Software muss alle der folgenden Kriterien erfüllen.

**/LF010/** Abspeichern von Datenverkehr in einer Protokolldatei (siehe Bild 3.1).

**/LF020/** Durchschleifen des Datenverkehrs (Prinzip eines transparenten Proxys).

**/LF030/** Eingriff in den Datenverkehr zur Laufzeit.

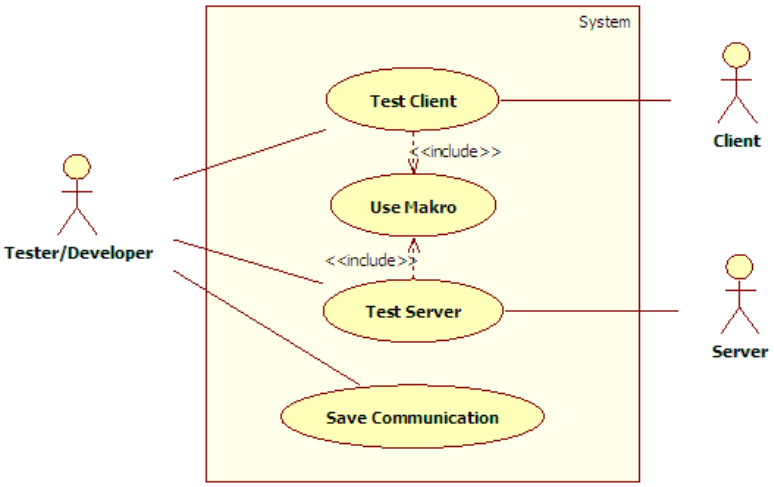


Bild 3.1: Anwendungsfalldiagramm UC 1

Anwendungsfalldiagramm UC 1 zeigt den Benutzer der Testsoftware und die beteiligte Software. Der Benutzer kann sowohl Serversoftware als auch Clientsoftware testen. Dazu können in beiden Fällen Makros verwendet

werden. Weiterhin hat der Benutzer die Möglichkeit den Datenverkehr aufzuzeichnen.

**/LF040/** Graphikunterstützte Darstellung des Datenverkehrs zwischen Server und Client durch Farbgebung, Trennung von Anfrage (Request) und Antwort (Reply) (siehe Bild 3.2).

**/LF050/** Getrennte Anzeige von Kopf- und Rumpfdaten (Head- und Bodydaten)
(siehe Bild 3.2).

**/LF060/** Verbergen und Anzeigen von Kommunikationsdetails (auf-/zuklappen)
(siehe Bild 3.2).

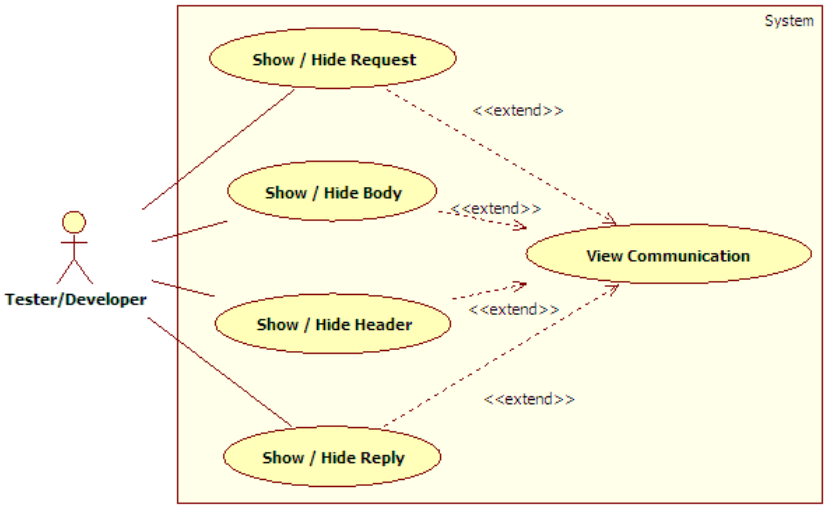


Bild 3.2: Anwendungsfalldiagramm UC 3

In Anwendungsfalldiagramm UC 3 ist dargestellt welche Möglichkeiten der Nutzer hat Details der ablaufenden Kommunikation anzuzeigen oder zu verbergen. So können Kopf- und Rumpfdaten ein- oder ausgeblendet werden. Die Kommunikationsstruktur wird durch die Trennung von Anfrage und Antwort verdeutlicht.

**/LF070/** Manuelles Zusammenstellen von Kommunikation zu einem Makro
(siehe Bild 3.4).

**/LF080/** Abspeichern von Teilen der Kommunikation als ein Makro (siehe Bild 3.4).

**/LF090/** Einseitige Simulation von Kommunikation durch Wiedergabe aufgezeichneter Kommunikationssequenzen (siehe Bild 3.3).

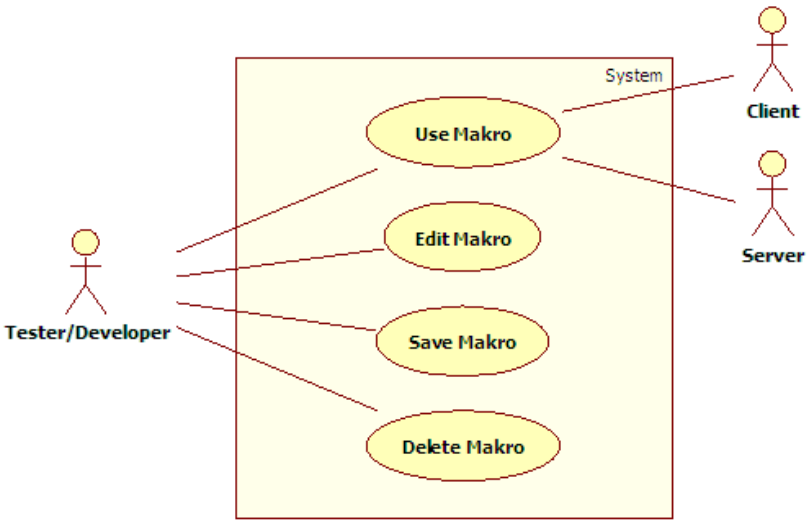


Bild 3.3: Anwendungsfalldiagramm UC 2

Das Anwendungsfalldiagramm UC 2 zeigt die Aktionen die dem Benutzer für ein Makro zur Verfügung stehen. So kann der Nutzer ein Makro bearbeiten, benutzen, speichern oder löschen.

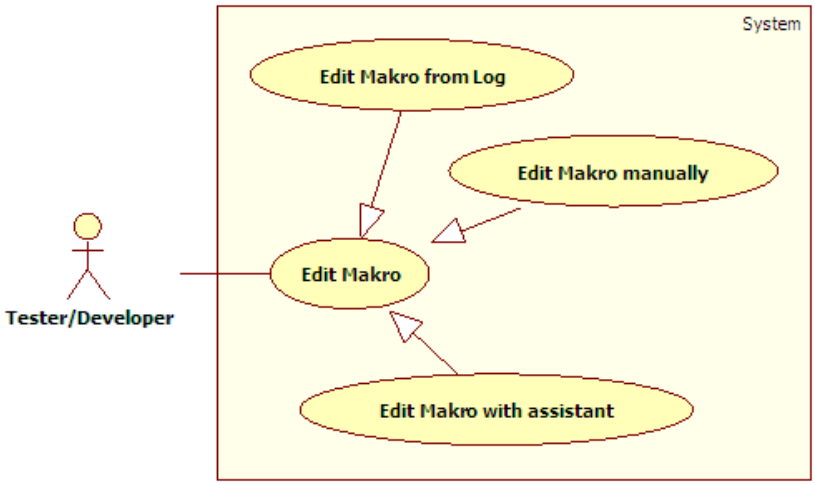


Bild 3.4: Anwendungsfalldiagramm UC 4

Das Anwendungsfalldiagramm UC 4 stellt im Detail die verschiedenen Bearbeitungsmöglichkeiten eines Makros dar. Im Einzelnen kann ein Makro damit aus aufgezeichneter Kommunikation, manuell oder unter zu Hilfenahme eines Assistenten erstellt werden.

## 3.2 Wunschkriterien

Es wäre wünschenswert, wenn die Software diese Kriterien erfüllt. Dies ist abhängig davon, ob noch genügend Zeit innerhalb des Zeitrahmens der Diplomarbeit zur Verfügung steht. Die Abarbeitung erfolgt dabei in der Reihenfolge der Nummerierung beginnend bei der niedrigsten Nummer.

**/LF100/** Assistent zum geführten Zusammenstellen von Makros (siehe Bild 3.4).

**/LF110/** Erweiterbarkeit der Kommunikationsanalyse durch Plugins vorsehen um z. B. später ACL, DASL oder DeltaV auswerten zu können (siehe Bild 3.5).

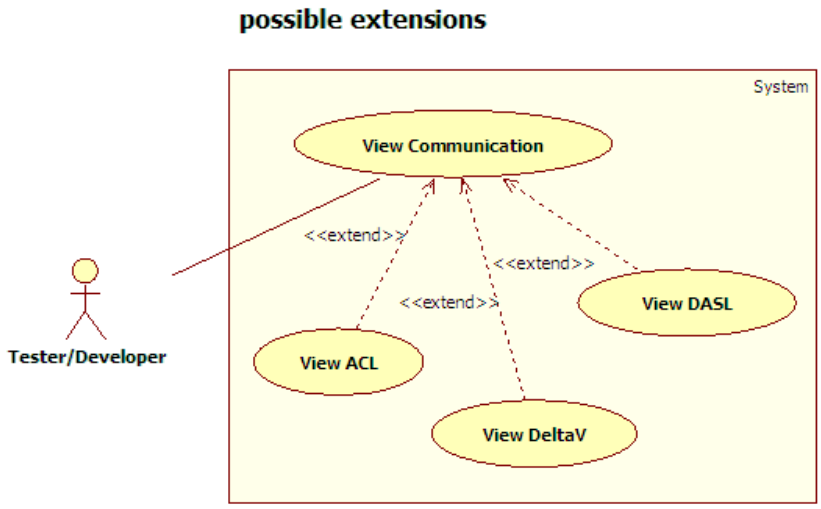


Bild 3.5: Anwendungsfalldiagramm UC 5

Anwendungsfalldiagramm UC 5 zeigt einige Erweiterungsmöglichkeiten für die Auswertung der Kommunikation. So kann das Programm später zum Beispiel um einen Filter für das ACL-Protokoll ergänzt werden und damit das Debugging für den Entwickler einer ACL-fähigen Software erleichtert und vereinfacht werden.

## 3.3 Abgrenzungskriterien

Um Klarheit zu schaffen, was die Software nicht leistet.

**/LF900/** Kein automatisierter Test von WebDAV-Software.

**/LF910/** Keine tiefgehende Analyse des WebDAV-Protokolls und Erweiterungen.

# 4 Produktdaten

Bei der Verwendung der Software wird zwischen folgenden Daten unterschieden:

**/LD010/** Konfigurationsdaten der Software.

**/LD020/** Protokolldatei des Datenverkehrs.

**/LD030/** Datenformat für gespeicherte Makros.

# 5 Produktleistung

**/LL010/** Das Produkt soll plattformunabhängig sein.

**/LL020/** Daten sollen dauerhaft auf einem Festspeicher gespeichert werden können.

**/LL030/** Die genutzten Ports für die Kommunikation sollen frei konfigurierbar sein.

**/LL040/** Das Produkt soll auf allen Rechnern mit mindestens einer Netzwerkschnittstelle lauffähig sein.

**/LL050/** Das Produkt soll komponentenbasiert entwickelt werden, damit das Produkt einfacher wartbar und erweiterbar bleibt.

**/LL060/** Für hinreichend schnelle Antwortzeiten und Analyse der Kommunikation wird der Einsatz von aktuellen Rechnersystemen vorausgesetzt.

# 6 Qualitätsanforderungen

Da es sich bei der zu entwickelnden Software um ein Werkzeug für Entwickler handelt, sind Ergonomie und intuitive Bedienbarkeit kein Hauptziel der Qualitätsbetrachtung. Trotzdem sollte auf eine übersichtliche und vor allem konsistente Gestaltung der Benutzerschnittstelle geachtet werden. Wichtiger jedoch ist die Zuverlässigkeit der Software, da sie als Werkzeug zum Entwickeln und Testen anderer Software dienen soll. Hierbei ist die Zuverlässigkeit unter dem Aspekt zu betrachten, dass die Software keine Fehler der zu analysierenden Software verdeckt und auch keine Fehler hinzufügt.

An dieser Stelle soll auch deutlich darauf hingewiesen werden, dass es sich bei der Software um ein Testwerkzeug handelt, das nicht für den Einsatz in einer Produktivumgebung gedacht ist. Unter einer Produktivumgebung ist in diesem Zusammenhang von der Produktivumgebung der zu testenden Software auszugehen, also der eigentlichen WebDAV-Applikation.

Die Anforderung an die Performance der zu entwickelnden Software ist unkritisch. Zunächst handelt es sich bei dem Http-/WebDAV-Protokoll um ein zustandsloses Protokoll. Das bedeutet, nach jeder erfolgreichen Datenübertragung kann die Verbindung geschlossen werden. Sollen erneut Daten übertragen werden, so muss eine neue Verbindung erstellt werden. Aufgrund der unterschiedlichen und nicht vorhersehbaren Ausbreitungspfade im Internet kann es zudem zu Latenzen kommen, die es nicht sinnvoll erscheinen lassen, spezielle Performance-Anforderungen an die Software zu stellen. Es muss lediglich ein flüssiges Arbeiten gewährleistet werden.

Ein weiteres wichtiges Ziel ist die Portierbarkeit und Kompatibilität der Software. Dies gilt sowohl für die Systemarchitektur, als auch für die Be-

nutzerschnittstelle. Die Software sollte mittels objektorientierter Programmierung erstellt werden. Weiterhin wird großen Wert auf die Dokumentation der Software und einheitlichen Quellcode gelegt. Um diese Ziele sicherzustellen, wird der Entwickler durch weitere Werkzeuge unterstützt. Das im DLR verwendete Werkzeug ist die Erweiterung Checkstyle für die zu verwendende Entwicklungsumgebung eclipse. Als Grundlage für die Konfiguration von Checkstyle dienen DLR interne Vorgaben. Als weitere Werkzeuge zur Sicherstellung der Softwarequalität werden Unit-Tests und Code-Reviews eingesetzt.

# 7 Entwicklungswerkzeuge

Zur Entwicklung stehen mehrere Werkzeuge, die den kompletten Entwicklungszyklus einer Software abdecken, zur Verfügung.

## 7.1 IDE

Zur Entwicklung wird die integrierte Entwicklungsumgebung eclipse eingesetzt. Für die verwendeten Programmiersprachen und Werkzeuge sind die benötigten Erweiterungen zu installieren. Im Einzelnen sind das:

- Subclipse – Erweiterung zum Zugriff auf die Versionsverwaltung
- Checkstyle – Automatische Überprüfung des Quellcodes im Hinblick auf formale Vorgaben (Kommentare, Klammerung, u. s. w.)

## 7.2 Konfigurationsmanagement

Die Sourceforge-Plattform bietet für dieses Projekt alle benötigten Dienste an. Im Einzelnen werden folgende Dienste verwendet:

- Versionsverwaltung mit Subversion
- Bug-Tracker
- Wiki zur Dokumentation
- Mailinglisten

# 8 Zeitplan

Das Projekt wird voraussichtlich im Oktober 2007 starten und im April 2008 beendet sein. Ein erster Zeitplan sieht wie folgt aus:

| Aufgabe | Dauer |
|---|---|
| Einarbeitung | 2 Wochen |
| Marktstudie (Sprache, Werkzeuge) | 4 Wochen |
| Lastenheft, Pflichtenheft | 8 Wochen |
| Prototyp | 2 Wochen |
| Implementierung | 6 Wochen |
| Test | 4 Wochen |
| Ausarbeitung / Abstract | 8 Wochen |
| Präsentation | 2 Wochen |
| Summe | 36 Wochen |

Tabelle 8.1: Zeitplanung

Die Differenzen zwischen der Summe der tabellarisch aufgelisteten Dauer der einzelnen Aufgaben und der gesamten Bearbeitungsdauer in Bild 8.1 kommen durch die teilweise parallele Abarbeitung von Aufgaben zustande.

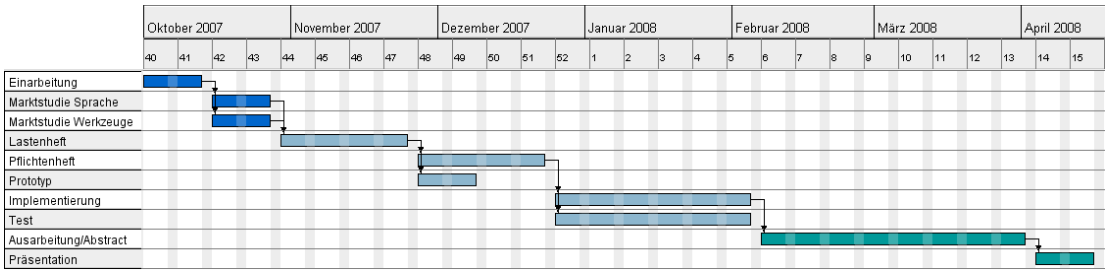


Bild 8.1: Aufgaben des Zeitplans als Gantt-Diagramm

# 9 Ergänzungen

## 9.1 Realisierung

Neben der Entwicklung der erwähnten Software besteht ein Teil der Diplomarbeit in der Untersuchung möglicher bereits vorhandener Werkzeuge auf deren Funktionsumfang und ob diese in geeigneter Weise erweitert werden könnten.

Die zu erstellende Software soll in der Programmiersprache C/C++, Java und/oder Python realisiert werden. Eine Entscheidung für eine dieser Sprachen oder eine Kombination ist im Verlauf der Diplomarbeit zu erarbeiten. Hierbei sind Portabilität und das Angebot an Bibliotheken der jeweiligen Sprache zu berücksichtigen. Weiterhin hat die Klarheit des Programmcodes bei der Entwicklung Vorrang gegenüber Optimierungen, die dem Einsparen von Speicherplatz und/oder Rechenzeit dienen.

Die schriftliche Darstellung der Ergebnisse und ein hausinterner Vortrag bilden den Abschluss der Diplomarbeit. Der Vortrag wird zusätzlich an der Universität Siegen abgehalten.

## 9.2 Open Source

Das Produkt setzt ausschließlich auf Open Source Software auf. Die jeweiligen Produkte stehen unter Copyright ihrer jeweiligen Ersteller. Deshalb wird auch dieses Produkt ein Open Source Produkt, das unter der Apache 2.0 Lizenz [APA07] eingesetzt und erweitert werden kann.

# Literaturverzeichnis

[APA07] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html
*15. Oktober 2007*

[RFC4918] RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)
http://www.ietf.org/rfc/rfc4918.txt
*15. Oktober 2007*

# Glossar

**ACL** Engl. Abkürzung für Access Control List. ACLs sind Listen, die Rechte von Benutzern auf Objekte speichern.

**Bug-Tracker** Software zur Verfolgung und Dokumentation von Fehlern in Software.

**Catacomb** Catacomb ist eine Erweiterung für das WebDAV-Modul des Apache Webservers. Die Daten werden hierbei statt im Dateisystem in einer relationalen Datenbank abgelegt.

Siehe auch: http://catacomb.tigris.org/

**Checkstyle** Checkstyle ist ein Werkzeug für Entwickler mit dem man Quelltext nach formalen Kriterien überprüfen kann.

Siehe auch: http://eclipse-cs.sourceforge.net/

**DASL** Engl. Abkürzung für DAV Searching and Locating. DASL befasst sich mit der serverseitigen Suche in WebDAV-Repositories. Die Arbeitsgruppe zu diesem Standard wurde allerdings mittlerweile aufgelöst.

**DataFinder** DataFinder ist eine Applikation auf Clientseite zur Verwaltung technisch-wissenschaftlicher Daten. Hierbei können die Daten sowie die Metadaten über verschiedenartige Speicherschnittstellen auf einem oder mehreren Servern abgelegt werden.

Siehe auch:
http://www.dlr.de/sc/desktopdefault.aspx/tabid-1189/1650_read-3140/

**DeltaV** Engl. Web Versioning and Configuration Management. Dieses Protokoll ist eine Erweiterung des WebDAV-Protokolls um Versions- und Konfigurationsmanagement.

**DLR** Deutsches Zentrum für Luft- und Raumfahrt.

Siehe auch: http://www.dlr.de

**Eclipse** Eclipse ist ein Framework zur Entwicklung von Anwendungen und ist vergleichbar mit einer IDE, kann aber auch noch wesentlich mehr bieten, da es durch eine entsprechende Architektur einfach erweiterbar ist.

**IDE** Engl. Abkürzung für Intergrated Developement Environment. Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software, das mehrere Komponenten in einer Oberfläche zusammenfasst.

**Loopback** Eine Loopback Netzwerkschnittstelle ist ein lokaler Informationskanal mit nur einem Endpunkt. Sender und Empfänger sind hierbei identisch.

**Mailingliste** Eine Mailingliste ist ein E-Mail-Verteiler für eine Gruppe von Menschen. Einzelne Mailinglisten haben häufig bestimmte Themen und dienen zur Kommunikation und Information.

**Makro** Ein Makro ist in der Programmierung eine vorgegebene Sequenz von Befehlen oder Aktionen. In diesem Zusammenhang bezeichnet ein Makro eine aufgezeichnete Kommunikationssequenz zwischen zwei WebDAV-Applikationen.

**MVC** Engl. Model-View-Controller. MVC ist ein zusammengesetztes Architekturmuster zur Gestaltung von Anwendungen. Um eine bessere Wiederverwendbarkeit und einfachere Erweiterbarkeit zu erhalten erfolgt eine Aufteilung in die Elemente Datenmodell (Model), Präsentation (View) und Steuerung (Controller).

**Open Source** Open Source Software ist quelloffene Software. Das heißt, der Quellcode dieser Software ist frei zugänglich und deren Bearbeitung und Verbreitung erlaubt und erwünscht. Es gibt verschiedene Lizenzmodelle, die eine genaue Definition der Verwendung und Verbreitung der jeweiligen Software regeln.

**Plugin** Ein Plugin ist ein Software- oder Hardwaremodul, welches in eine bestehende Software oder Hardware eingebunden wird und die Funktionalität der Software oder Hardware erweitert.

**Proxy** Ein Proxy ist im Zusammenhang mit Netzwerken ein Dienstprogramm zur Steuerung von Netzwerkverkehr. Ein Proxy besitzt sowohl eine ServerSchnittstelle als

auch eine Client-Schnittstelle. Im einfachsten Falle leitet der Proxy den Netzwerkverkehr unbeeinflusst weiter. In diesem Fall spricht man auch von einem „Relay“.

**RFC** Engl. Requests for Comments. Die RFCs sind eine Sammlung von technischen und organisatorischen Dokumenten zu den im Internet verwendeten Protokollen und Verfahren.

**SC** Abkürzung für die Abteilung Simulations- und Softwaretechnik innerhalb des DLR.

Siehe auch: http://www.dlr.de/sc/

**sourceforge.net** sourceforge.net ist eine Online-Plattform. Hier werden einem oder einer Gruppe von Entwicklern Werkzeuge zur Unterstützung bei der Anwendungsentwicklung und zur Projektverwaltung geboten.

Siehe auch: http://www.sourceforge.net/

**Subclipse** Eclipse-Plugin für die Versionsverwaltung Subversion.

**Subversion** Open Source Software für die Versionsverwaltung von Dateien und Ordnern.

Siehe auch: http://subversion.tigris.org/

**TCP** Engl. Abkürzung für Transmission Control Protocol. TCP ist ein verbindungsorientiertes Transportprotokoll der OSI-Schicht 4.

**UML** Engl. Abkürzung für Unified Modeling Language. Eine von der Object Management Group entwickelte Sprache zur Modellierung von Software.

**Unit-Test** Der Unit-Test wird auch als Modultest bezeichnet und dient zur Überprüfung der Korrektheit von Programmteilen.

**WebDAV** Engl. Abkürzung für Web-based Distributed Authoring and Versioning. WebDAV ist ein offener Standard zur Bereitstellung von Dateien im Internet und ist eine Weiterentwicklung des HTTP-Protokolls.

Siehe auch: http://www.webdav.org/

**Wiki** Ein Wiki ist eine Sammlung von Hypertextseiten, die von Benutzern gelesen, editiert und verknüpft werden können.

# D Marktstudie


Elektrotechnik und Informatik

Diplomarbeit

# Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen

Marktstudie

Revision 0.5

# Inhaltsverzeichnis

# 1 Einleitung

Die vorliegende Marktstudie gliedert sich in zwei Teile:

- Das Kapitel Werkzeuge behandelt bereits existierende Werkzeuge zum Debuggen des WebDAV-Protokolls und/oder von Netzwerkverkehr. Zunächst werden die für dieses Projekt wichtigen Anforderungen an die Produkte definiert. Die einzelnen Werkzeuge werden dann kurz vorgestellt und ihre Besonderheiten hervorgehoben. Abschließend werden die wichtigsten Kriterien noch einmal tabellarisch dargestellt und die gewonnen Erkenntnisse zusammengefasst.
- Das Kapitel Implementierungssprache beschäftigt sich mit der Auswahl der zu verwendenden Programmiersprache. Die Anforderungen ergeben sich hierbei aus der im vorigen Kapitel getroffenen Entscheidung. Zusätzliche Kriterien werden ebenfalls zunächst definiert und dann folgt eine Betrachtung der zur Wahl stehenden Programmiersprachen. Zum Schluss erfolgt eine Zusammenfassung und Bewertung.

# 2 Werkzeuge

## 2.1 Anforderungen

Folgende Forderungen standen bereits zu Beginn des Projektes fest:

- Das fertige Produkt soll unter einer passenden Open Source Lizenz stehen. Das bedeutet für Erweiterungen bereits bestehender Produkte, dass die Lizenzbedingen der Ausgangssoftware dies zulassen müssen.
- Das erstellte Produkt soll Betriebssystemunabhängig sein. Eventuell zu erweiternde Produkte müssen diese Forderung ebenfalls erfüllen.
- Datenverkehr soll zunächst unbeeinflusst weitergeleitet werden und für die Analyse aufbereitet werden. Später soll gegebenenfalls ein Eingriff in den Datenverkehr umgesetzt werden.
- Eine graphische Oberfläche sollte vorhanden sein.
- Optimal wäre bereits eine vorhandene WebDAV-Unterstützung.

## 2.2 Bewertung

Zur Bewertung vorhandener Produkte, die die obigen Forderungen erfüllen, werden jetzt zusätzlich noch folgende Punkte betrachtet:

- Einsatzzweck des Werkzeugs
- Leistungsmerkmale im Allgemeinen
- Aufwand für die Nutzung des Werkzeuges
- Erweiterbarkeit

### 2.2.1 Http Probe

Http Probe ist ein in C# geschriebener HTTP- und WebDAV-Client zum Erstellen und Auswerten von WebDAV- und HTTP-Anfragen und basiert damit auf der .NET Plattform. Entwickelt wurde das Programm von Stefan Delmarco, einem Microsoft Most Valuable Professionals [MVP07]. Die Lizenz unter der das Werkzeug verteilt wird konnte bisher nicht ermittelt werden und ist somit unbekannt. Der Quellcode ist jedoch im Paket der Applikation enthalten.

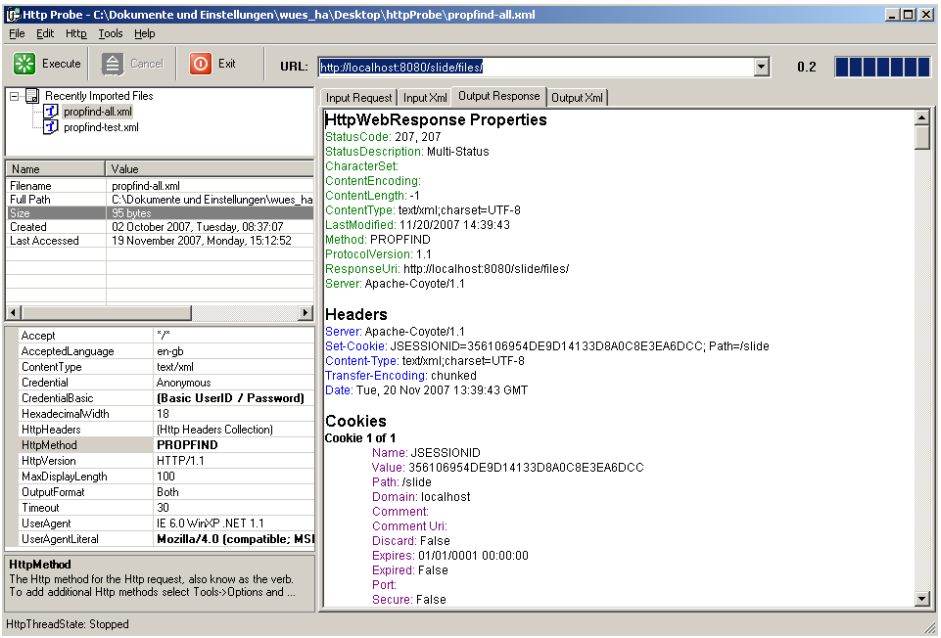


Bild 2.1: Http Probe

Das Werkzeug wird innerhalb des DLR bereits verwendet und zeichnet sich durch gute Übersichtlichkeit und einfache Bedienung aus. Die Benutzerschnittstelle besteht aus drei wichtigen Bereichen, auch dargestellt in Bild 2.1:

- Werkzeugleiste
- Ansichten für Anfrage- und Antwortdaten
- Einstellungen für Kopfdaten und Vorlagendateien für Anfragen

In der Werkzeugleiste kann die gewünschte Zieladresse eingeben und die Ausführung von Anfragen angestoßen werden. Die verschiedenen Ansichten für die Ein- und Ausgabedaten dem Benutzer eine einfache und übersichtliche Auswertung der Kommunikation. Auf der linken Seite können die Einstellungen für die Kopfdaten vorgenommen werden und frei definierbare Vorlagen für Anfragen können hier aus Dateien importiert werden. Was fehlt ist eine Verlaufsansicht der bisher erfolgten Anfragen mit einer entsprechenden Zuordnung der jeweiligen Antworten.

**Positive Merkmale:** Quellcode verfügbar, Sprache C#, GUI, Http-Debugger, WebDAV Unterstützung

**Negative Merkmale:** reiner Client, nur lauffähig unter Windows Betriebssystemen, unbekannte Lizenz

### 2.2.2 Apache JMeter

Bei Apache JMeter [AJM07] handelt es sich um ein Werkzeug für Lasttests von Serveranwendungen. Mit dieser Software kann sowohl die Leistung von statischen, als auch von dynamischen Serverdiensten wie etwa Servlets oder Datenbanken getestet und ausgewertet werden. Das Produkt ist vollständig in Java geschrieben und ist als Teil des Apache Jakarta Projektes [AJP07] unter der Apache 2.0 Lizenz [APA07] erhältlich.

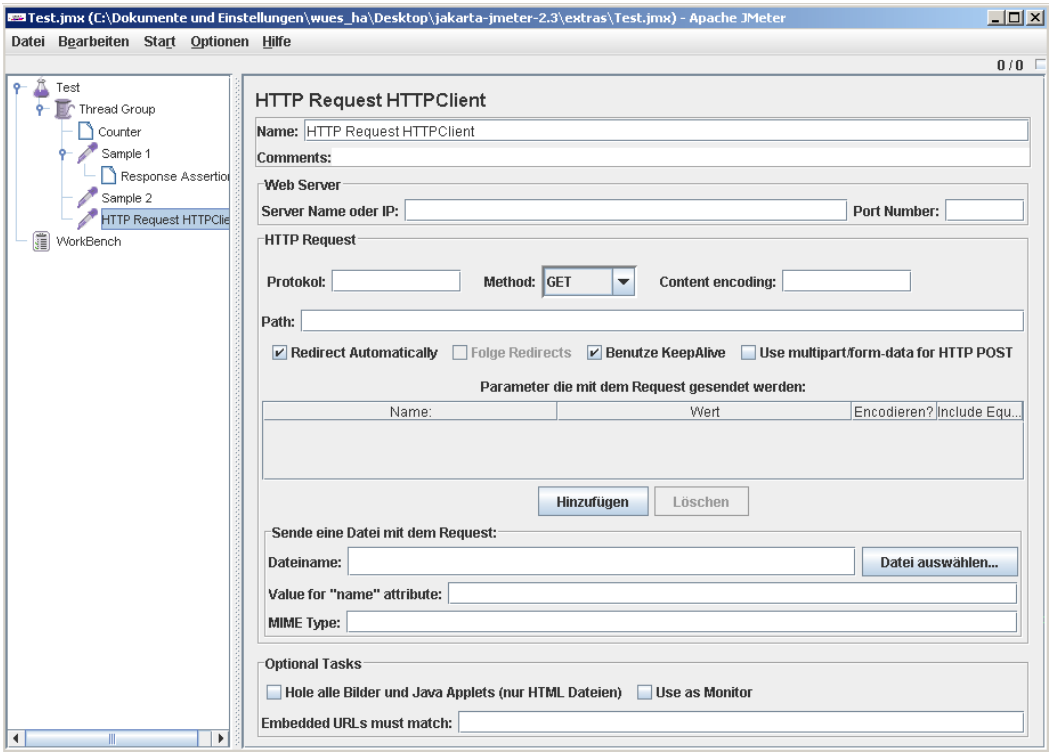


Bild 2.2: Apache JMeter

Wie aus Bild 2.2 ersichtlich können in einer Baumansicht vom Benutzer verschiedene Tests- und Testgruppen angelegt werden. Die Tests können für die unterschiedlichsten Szenarien ausgelegt werden und sowohl für den Test als auch für die Auswertung des Tests ist Logik definierbar. Insgesamt ist Apache JMeter ein sehr umfangreiches Produkt mit sehr vielen Möglichkeiten, jedoch ist das Projekt eindeutig auf fest definierte Lasttests Ausgerichtet und nicht für interaktives Arbeiten.

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache Java, Swing GUI

**Negative Merkmale:** reiner Client, Test von Servern und Netzwerkanwendungen

### 2.2.3 ntop

ntop ist ein auf die Analyse des Netzwerkverkehrs spezialisiertes Werkzeug. Allgemein wird solch eine Software auch als Sniffer bezeichnet. Der komplette Datenverkehr ab der Sicherungsschicht wird aufgezeichnet und kann ausgewertet werden. Die Konfiguration und Analyse ist über eine Webbrowser möglich. Exemplarisch ist in Bild 2.3 das Diagramm des Netzwerkdurchsatzes dargestellt. Unter den Microsoft Betriebssystemen ab Windows 95 ist auch eine grafische Oberfläche verfügbar. Das Programm und der Quellcode stehen unter GNU GPL.

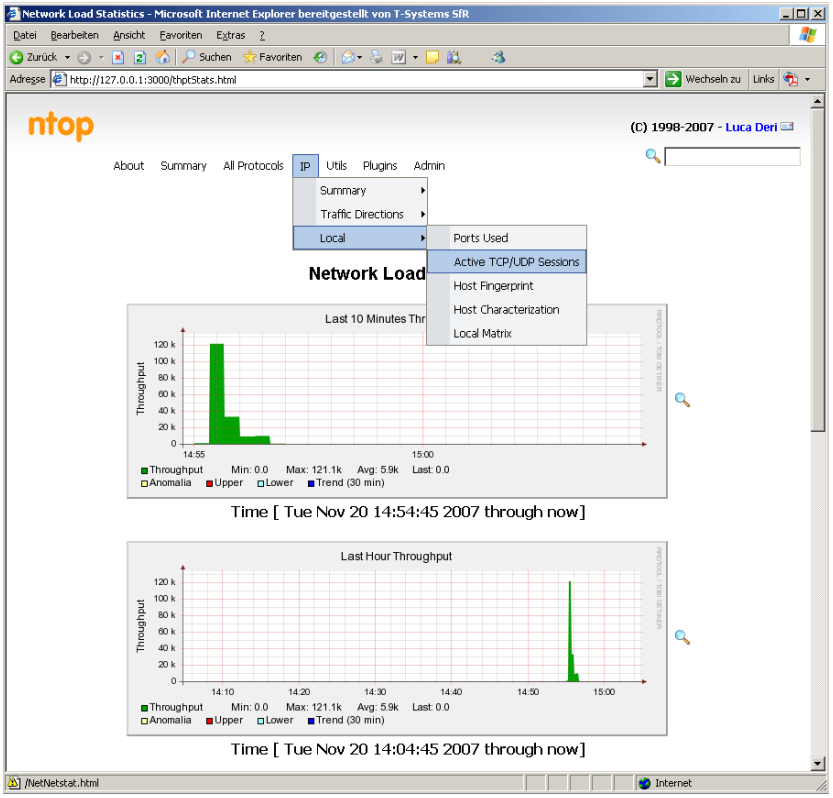


Bild 2.3: ntop Weboberfläche - Durchsatzdiagramm

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache C

**Negative Merkmale:** Netzwerkprotokollanalyse/Sniffer, Weboberfläche

### 2.2.4 Wireshark

Bei Wireshark handelt es sich ebenfalls um eine Software die als Sniffer bezeichnet wird und zur Analyse der Netzwerkkommunikationsverbindungen dient. Wireshark ist ein Folgeprojekt von Ethereal [ERW07].

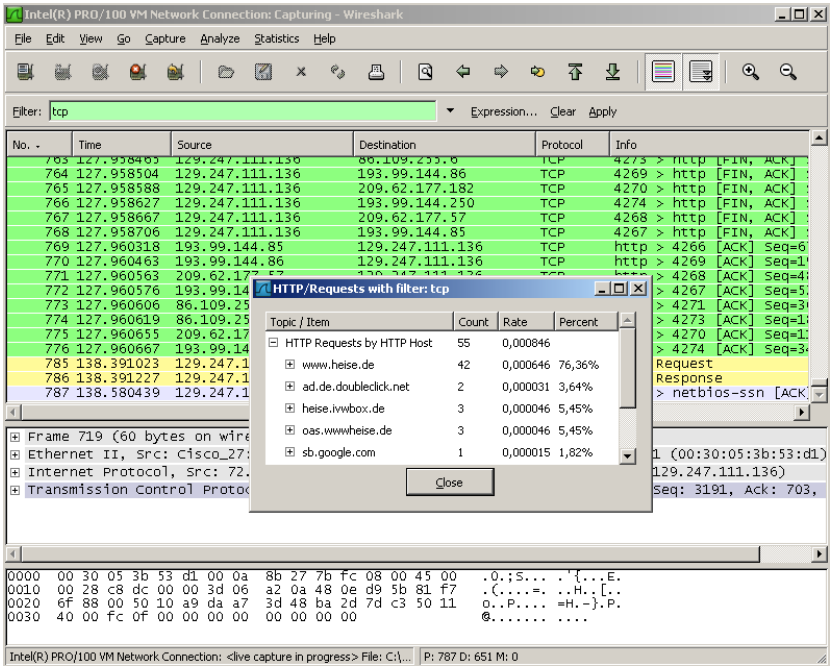


Bild 2.4: Wireshark mit aktivem TCP-Filter

Die Software ist unter der GNU GPL Lizenz [GPL07] erhältlich und stellt einen Quasistandard in dieser Softwarekategorie dar. Die Aufzeichnung des Datenverkehrs ist ab der Sicherungsschicht möglich und kann über sehr viele verfügbare Filtermöglichkeiten [WiS07] ausgewertet werden. Das Programm verfügt über eine grafische Benutzeroberfläche und ist unter Windows, beispielhaft ist eine Auswertung in Bild 2.4 dargestellt, Linux, sowie diversen anderen Betriebssystemen lauffähig.

**Positive Merkmale:** Open Source,Betriebssytemunabhängig, Sprache C, GUI

**Negative Merkmale:** Netzwerkprotokollanalyse/Sniffer

### 2.2.5 Charles

Charles ist ein Werkzeug für Entwickler und arbeitet nach dem Prinzip eines Proxys. Die Software wird als Shareware vertrieben und für eine uneingeschränkte Funktionalität ist ein Kauf des Produktes unumgänglich [Cha07]. Charles nutzt die Programmiersprache Java und ist somit auf vielen Betriebssystemen lauffähig. Der Quellcode der Software ist nicht verfügbar.

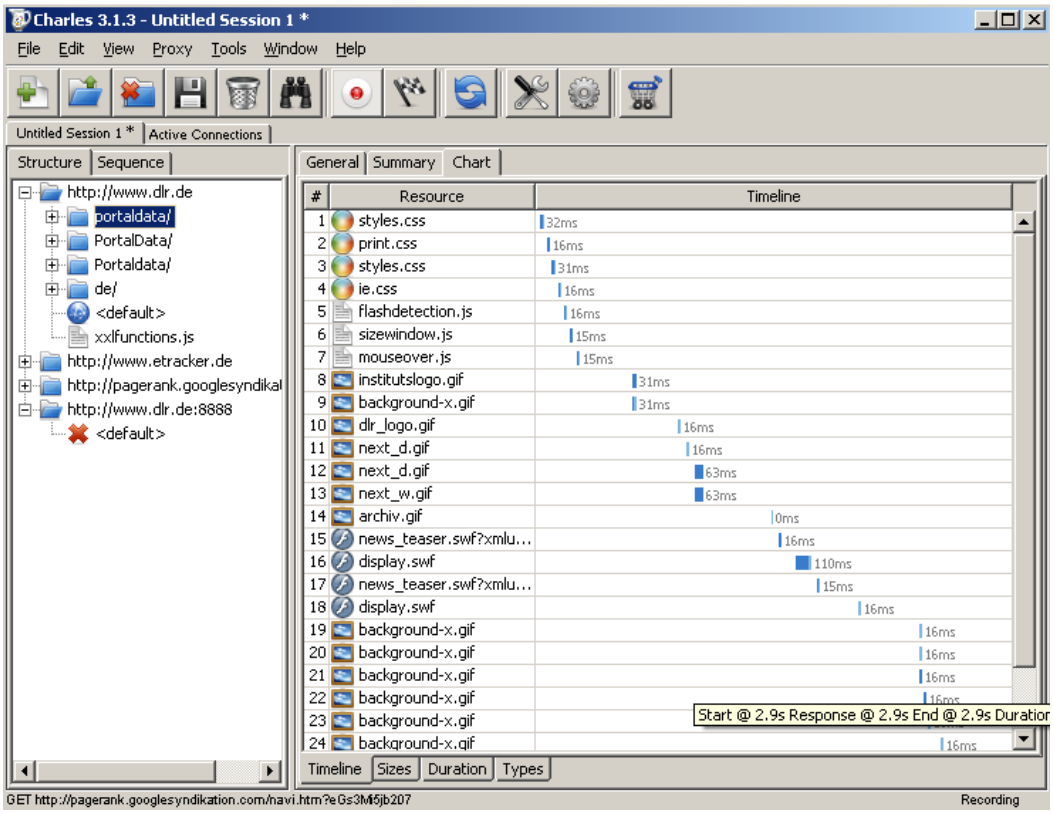


Bild 2.5: Charles Structure Timeline

Bild 2.5 zeigt eine der möglichen Ansichten des von Charles aufgezeichneten Datenverkehrs. Hierbei handelt es sich um eine Darstellung der Ladezeiten für die verschiedenen Elemente einer Webseite, hier beispielhaft die DLR-Startseite. Dies ermöglicht Webentwicklern die Ladezeiten ihre Seiten zu optimieren. Als Besonderheit verfügt Charles über die Möglichkeit die Übertragungsbandbreite zu drosseln um dadurch langsame Internetverbindungen, etwa per Modem, zu simulieren.

**Positive Merkmale:** Betriebssytemunabhängig, Sprache Java, Http-Debugger, GUI

**Negative Merkmale:** Shareware, kein Quellcode verfügbar

### 2.2.6 HttpWatch

HttpWatch ist ein Plugin für den Microsoft Internet Explorer und damit nur für Microsoft Betriebssysteme verfügbar. Durch die Integration in den Webbrowser ist das Programm sehr einfach zu bedienen und zudem übersichtlich gestaltet. HttpWatch wird kommerziell vertrieben daher ist auch kein Quellcode verfügbar [HTW07]. Zielgruppe für die Software sind hauptsächlich Webentwickler, die mit diesem Werkzeug ihre Webseiten analysieren und optimieren können. In Bild 2.6 kann man beispielhaft die verschiedenen Ladezeiten der Http-Anfragen für einen Aufruf der DLR-Portalseite sehen.

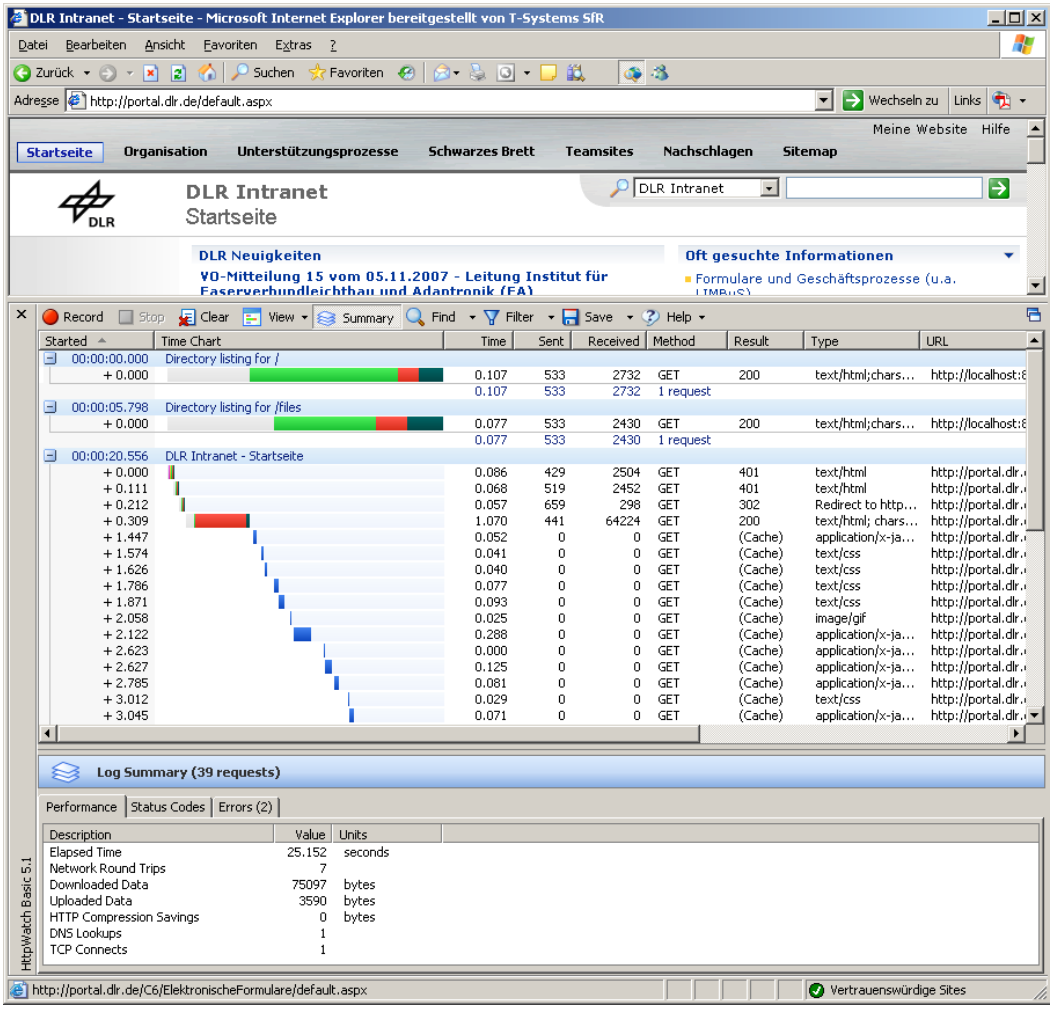


Bild 2.6: HttpWatch im Microsoft Internet Explorer

**Positive Merkmale:** Http-Debugger, Automatisierungsschnittstelle

**Negative Merkmale:** kommerzielles Produkt, nur lauffähig unter Windows Betriebssystemen, Quellcode ist nicht verfügbar, Sprache unbekannt, Browser-Plugin

### 2.2.7 HTTP Debugger

Der Http Debugger ist ein nur für Microsoft Betriebssysteme erhältlicher Http-Proxy. Die Software richtet sich sowohl an Web- und Applikationsentwickler, als auch an Netzwerk- und Systemadministratoren. Wie in Bild 2.7 zu erkennen gliedert sich das Programmfenster von Http Debugger in zwei Teile. Auf der rechten Seite ist wahlweise der Verlauf der bisherigen Http-Anfragen in tabellarischer Form dargestellt oder der Benutzer kann sich die Anfrage/ Antwort in Rohform oder aufbereitet anzeigen lassen. Auf der rechten Seite sind jeweils Details zu den Kopfdaten der aktuell ausgewählten Http-Anfrage dargestellt. Als Besonderheit ist für den Http Debugger ein SDK verfügbar [HTD07].

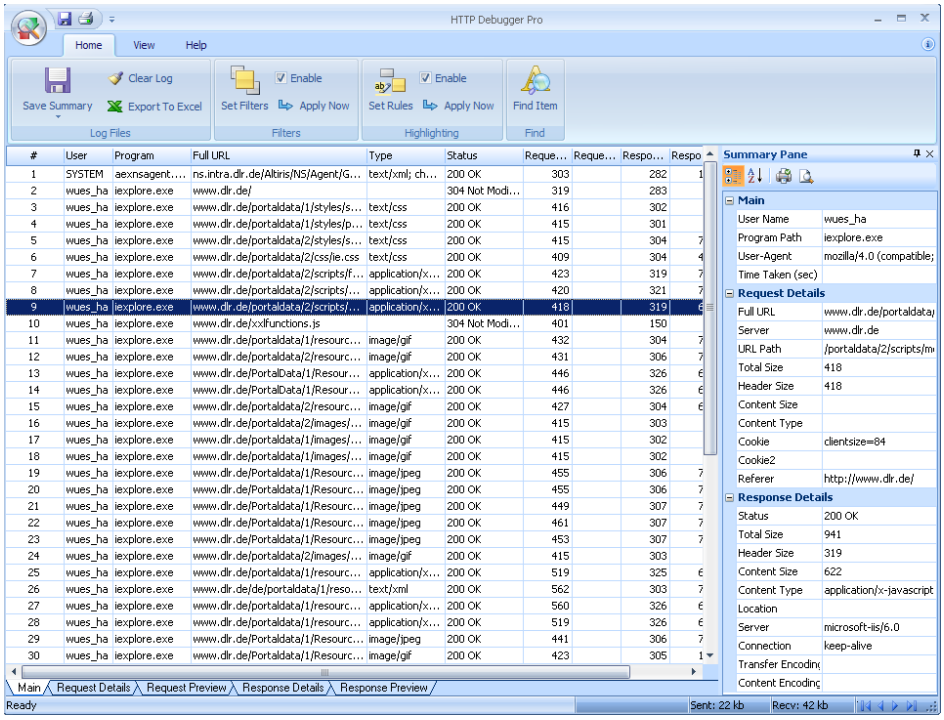


Bild 2.7: Hauptansicht Http Debugger

**Positive Merkmale:** Http-Debugger, SDK verfügbar, GUI

**Negative Merkmale:** kommerzielles Produkt, nur lauffähig unter Windows Betriebssystemen, Quellcode ist nicht verfügbar, Sprache unbekannt

### 2.2.8 Fiddler

Fiddler ist ein Werkzeug der Firma Microsoft zum Debuggen von Anwendungen, die das Http-Protokoll verwenden [Fid07]. Das Programm selber ist Freeware, der Quellcode ist jedoch nicht verfügbar. Der Einsatz von Fiddler ist auf .NET-fähige Plattformen beschränkt. Zu den besonderen Eigenschaften von Fiddler gehört die Erweiterungsmöglichkeit durch Skripte und das Setzen von Haltepunkten. In Bild 2.8 ist auf der rechten Seite exemplarisch die Ansicht der Kopfdaten der aktuell ausgewählten Http-Anfrage und auf der linken Seite der Verlauf der bisherigen Http-Anfragen dargestellt.

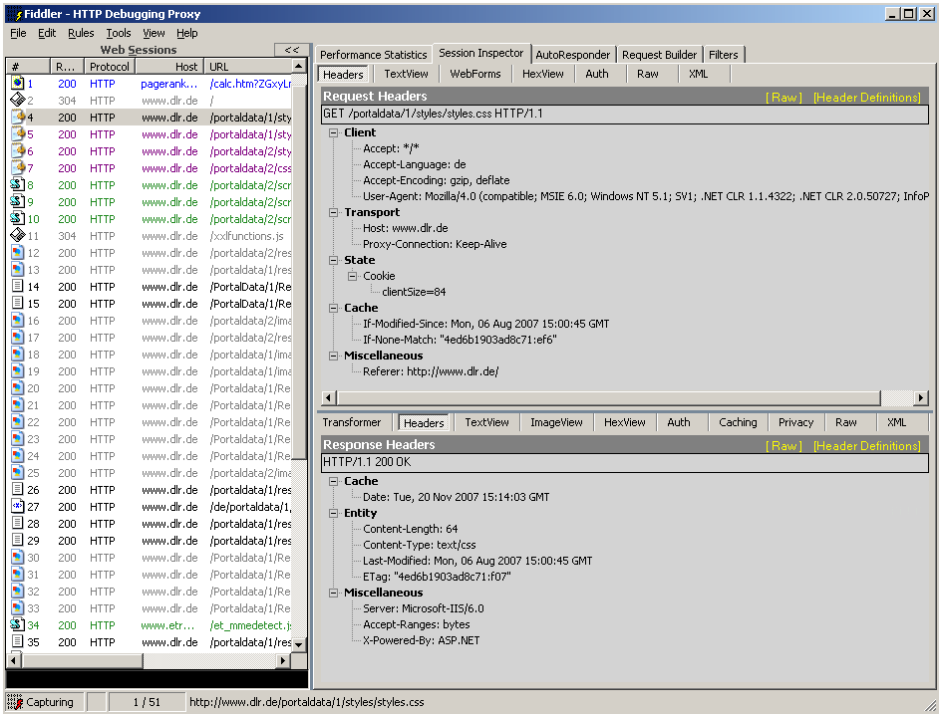


Bild 2.8: Fiddler Http Debugging Proxy - Header

**Positive Merkmale:** Http-Debugger, Automatisierungsschnittstelle, Sprache C#, GUI

**Negative Merkmale:** Freeware, nur lauffähig unter Windows Betriebssystemen, Quellcode ist nicht verfügbar

### 2.2.9 muffin

Bei muffin handelt es ich um ein in der Sprache Java geschriebenen Proxy unter GNU GPL [GPL07]. Hauptanwendungsgebiet von muffin ist das Filtern der übertragenen Daten um den Benutzer vor störenden und gefährlichen Inhalten aus dem Internet zu schützen [Muf07]. Das Programm besitzt eine AWT-Oberfläche, kann aber auch über eine Konsole oder eine Weboberfläche administriert werden.

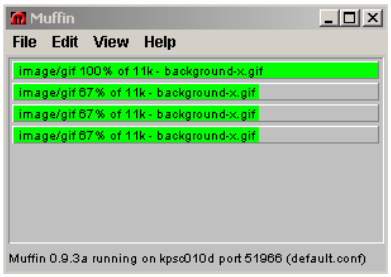


Bild 2.9: muffin Hauptfenster

Die Informationen zu den aktuell gefilterten Daten sind, wie in Bild 2.9 und Bild 2.10 zu erkennen, sehr knapp und in dieser Form schlecht nutzbar. Zudem gibt es keine Möglichkeit den aufgezeichneten Datenverkehr zu analysieren. Weiterhin wird das Projekt nicht mehr gepflegt und weiterentwickelt [MSF07].

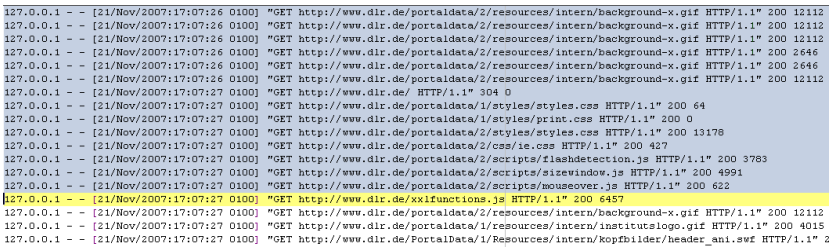


Bild 2.10: muffin Logfile

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache Java

**Negative Merkmale:** Produkt wird nicht mehr gepflegt, Contentfilter/Anonymisierer, reiner Proxy, AWT Oberfläche

### 2.2.10 NetTool

NetTool ist sowohl ein Http-Client als auch TCP-Proxy. Das Programm kann Http-Anfragen absetzen und TCP-Verbindungen weiterleiten. NetTool wurde von Neil O'Toole in der Programmiersprache Java geschrieben und steht unter LGPL, beziehungsweise die benutzten Bibliotheken unter ihren jeweils eigenen Lizenzen. Die Lizenz war zunächst ungeklärt, da auf der Sourceforge-Projektseite keine Angabe zur Lizenz zu finden war [Net07]. Diese lies sich eindeutig nur dem Quelltext entnehmen. Der Quellcode selbst ist bis auf fünf Klassen vollständig vorhanden. Die letzten Änderungen stammen vom 9.10.2002. Zusätzlich wurde bezüglich des Quellcodes eine Anfrage an den Autor per Email gestellt [NEm07].

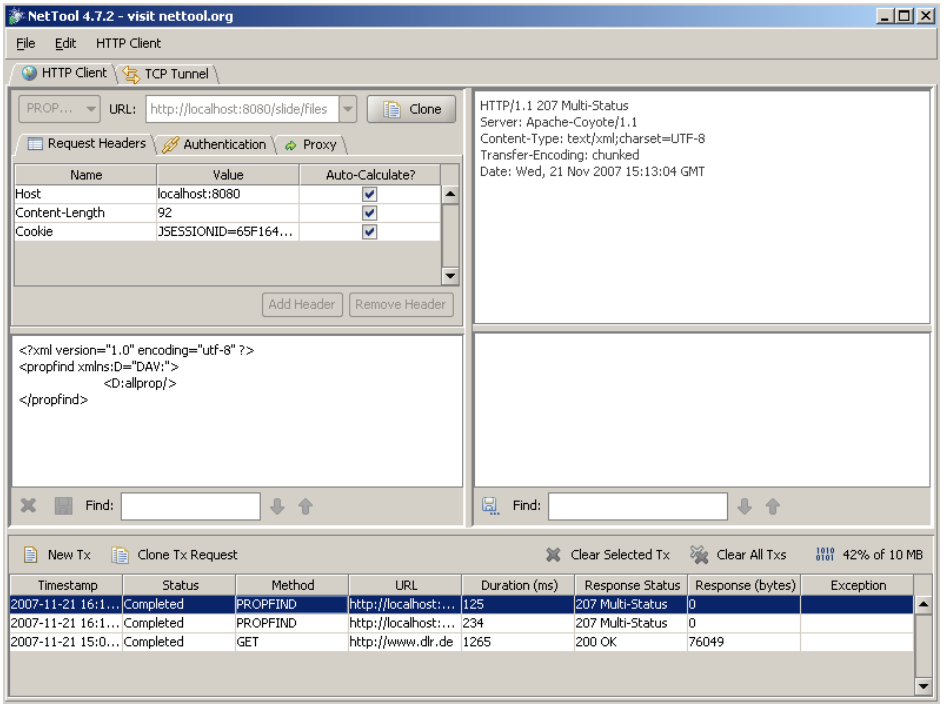


Bild 2.11: NetTool - Client-Modus

NetTool bietet zwei Hauptansichten. In Bild 2.11 ist der Client-Modus von NetTool dargestellt. Auf der linken Seite können in der oberen Hälfte die URL und die Kopfdaten für eine Anfrage bearbeitet werden. In der unteren

Hälfte können Rumpfdaten hinterlegt werden. Auf der rechten Seite findet man von oben nach unten jeweils die Kopfdaten der Antwort und darunter den Rumpf der Antwort. Der unterste Abschnitt des Programmfensters enthält eine Verlaufsansicht der bisherigen Anfragen.

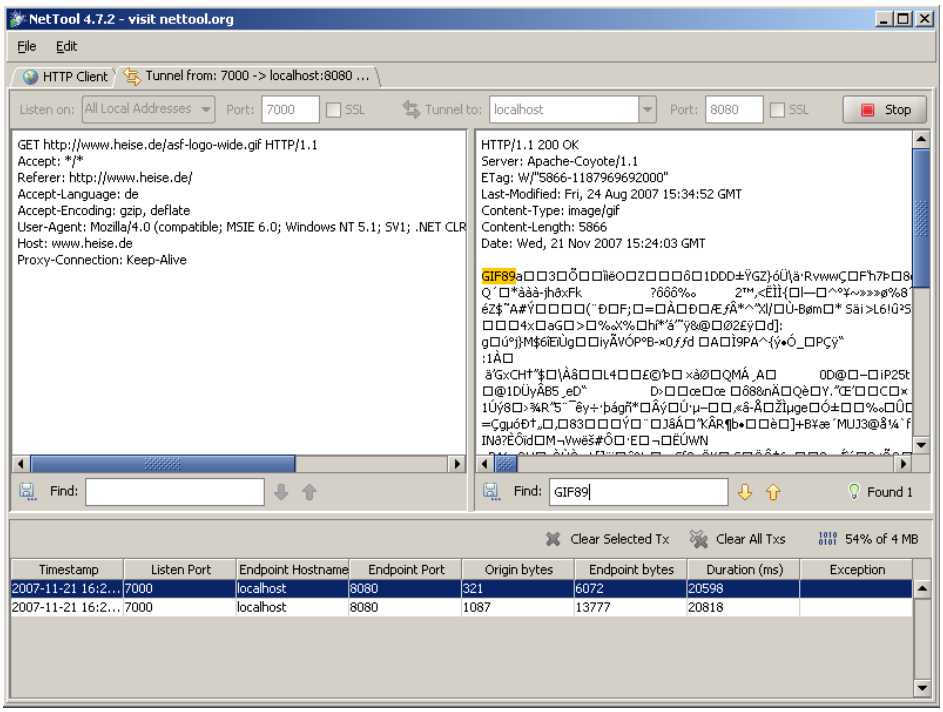


Bild 2.12: NetTool - Tunnelmodus

Bild 2.12 zeigt den Aufbau der Benutzerschnittstelle im Tunnelmodus. In der Kopfzeile kann man hier den Quellport, sowie Zielport und -Adresse setzen. Darunter findet man auf der linken Seite die Anfragen und auf der rechten Seite die Antworten. Am unteren Rand des Programmfensters befindet sich, wie im Client-Modus, wiederum eine Auflistung der bisher durchgeleiteten Anfragen.

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache Java, Http-Debugger, Swing Oberfläche, rudimentäre WebDAV-Unterstützung

**Negative Merkmale:** Der Quellcode ist nur in einer alten Version zugänglich.

### 2.2.11 Proximodo

Proximodo ist ein unter GNU GPL entwickelter Http-Proxy [Pro07]. Das Proximodo Projekt ist eine Weiterführung des Proxomitron Projektes unter einer Open Source Lizenz und mit einer Multiplattform fähigen Oberfläche basierend auf wxWidgets. Das Programm selber ist in C++ geschrieben und befindet sich noch im Alpha-Status [PSF07].


Bild 2.13: Proximodo Hauptfenster

Hauptanwendung von Proximodo ist das Filtern von Webinhalten zum Schutz des Nutzers vor störenden und gefährlichen Inhalten. Die Oberfläche des Programms ist in Bild 2.13 und Bild 2.14 dargestellt. Die Gestaltung ist übersichtlich und einfach.

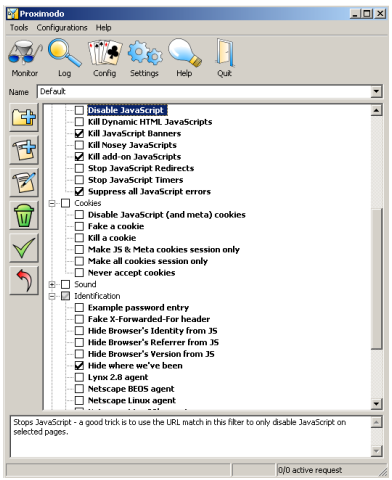


Bild 2.14: Proximodo Konfigurationsfenster

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache C++, GUI

**Negative Merkmale:** Contentfilter/Anonymisierer, reiner Proxy

### 2.2.12 Privoxy

Privoxy ist ebenfalls ein Http-Proxy, verfügbar für verschiedene Plattformen und erhältlich unter GNU GPL [Pri07]. Das Programm lässt sich über eine grafische Oberfläche, eine Konsole oder eine Weboberfläche administrieren. Privoxy ist in der Programmiersprache C geschrieben und sehr ausgereift mit einer größeren Entwicklergemeinde [PrS07]. Bild 2.15 zeigt das Hauptfenster von Privoxy. Auch hier ist das Anwendungsgebiet wieder der Schutz der Privatspähre des Benutzers. Das Programm dient also zum Filtern von schädlichen oder störenden Inhalten in Http-Kommunikation.

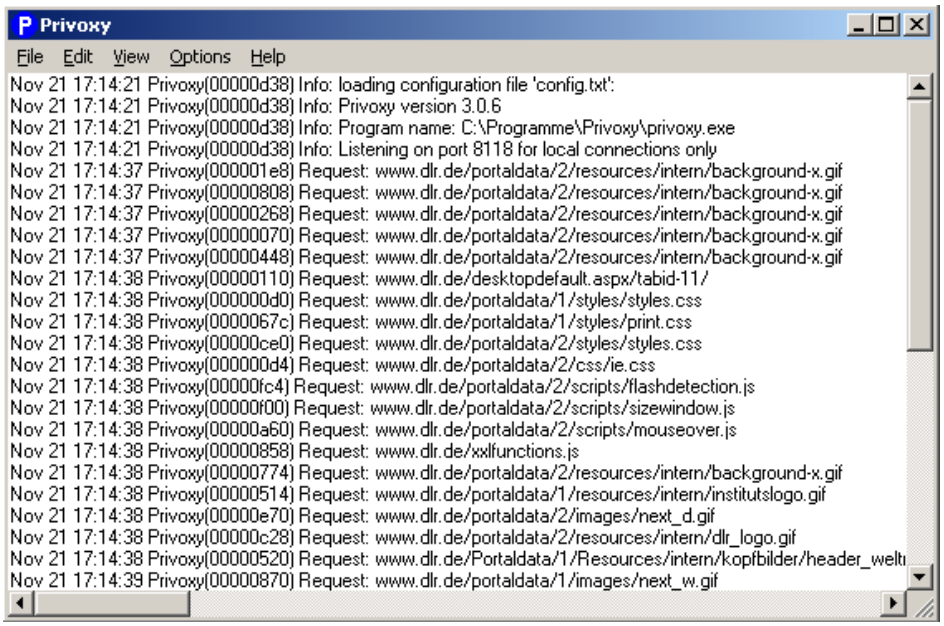


Bild 2.15: Privoxy Hauptfenster

**Positive Merkmale:** Open Source, Betriebssytemunabhängig, Sprache C

**Negative Merkmale:** Weboberfläche, Contentfilter/Anonymisierer, reiner Proxy

## 2.3 Fazit

In der nun folgenden Zusammenfassung werden die wichtigsten Kriterien nochmals erläutert und getroffene Entscheidung diskutiert. Die Menge an angebotenen Produkten ist inzwischen sehr groß. Während der Marktanalyse wurden noch weitere Produkte untersucht. Viele davon schieden wegen ihrer begrenzten Eigenschaften jedoch früh aus. Ausschlusskriterien für weitere Produkte waren insbesondere eine unpassende Lizenz, Beschränkung auf eine Plattform oder ein unpassender Anwendungsfall.

In Tabelle 2.1 sind die wichtigsten Eigenschaften der zuvor besprochenen Produkte noch einmal kompakt dargestellt. Die hierbei berücksichtigten Kriterien sind in den im Folgenden erläuterten Spalten dargestellt:

- **Produkt:** Die Bezeichnung der betrachteten Software steht in der ersten Spalte.
- **Lizenz:** In dieser Spalte wird die Lizenz des Produktes, sofern verfügbar, angegeben und ob der Quellcode für dieses Produkt erhältlich ist.
- **Betriebsystem:** Die Spalte „Betriebsystem“ enthält die Betriebsysteme, beziehungsweise Betriebssystemgruppen, unter denen das Produkt eingesetzt werden kann.
- **Sprache:** Die Programmiersprache, sofern ermittelbar, in der die Software implementiert wurde ist in dieser Spalte hinterlegt.
- **Konzept:** Das Konzept spiegelt die Arbeitsweise des Produktes auf Netzwerkebene wieder.
- **GUI:** In der Spalte „GUI“ ist die verwendete GUI-Technik, sofern diese ermittelt werden konnte, angegeben.
- **WebDAV:** Die Spalte „WebDAV“ gibt die WebDAV-Unterstützung des Produktes an.

- **Einsatzzweck:** Hier ist der Hauptanwendungsfall des Produktes angegeben.
  Die untersuchten Produkte lassen sich generell in vier Kategorien unterteilen:

  1. Http-Debugger
  2. Server-Test
  3. Netzwerkprotokollanalyse
  4. Contentfilter / Anonymisierer

- **Erweiterbar:** In dieser Spalte ist die Erweiterbarkeit des Produktes hinterlegt. Bei allen Open Source Produkten ist dies durch die Verfügbarkeit der Quelltexte gegeben. Bei Verfügbarkeit einer Automatisierungsschnittstelle können Abläufe durch externe Skripte angestoßen und gesteuert werden. Ein SDK unterstützt die Erstellung einer eigenen Erweiterung auf Basis des Produktes, jedoch ohne die Möglichkeit den Kern der Applikation zu verändern.

Das gewünschte Anforderungsprofil umfasst als wichtigste Punkte eine Open Source Lizenz, Plattformunabhängigkeit der Software einschließlich der Oberfläche und Erweiterbarkeit. Von Vorteil ist eine bereits vorhandene WebDAVUnterstützung, ein Einsatzfeld als Http-Debugger und eine Auslegung des Werkzeugs als Client und auch Proxy.

Als weitere Hilfe zur Entscheidungsfindung wurden im Rahmen einer Videokonferenz mit den WebDAVEntwicklern für Client- und Serverseite des DLR die verschiedenen Produkte vorgestellt und erläutert. Hierbei wurden drei Produkte besonders herausgehoben: Http Probe, Fiddler und NetTool. Http Probe wird bereits rudimentär verwendet und überzeugt durch einfach und selbsterklärende Bedienung. Allerdings ist dieses Werkzeug nur ein WebDAVClient und bietet keine ProxyFunktionalität und keine Verlaufsansicht. Fiddler besticht durch einen klaren Aufbau der Oberfläche mittels Registerkarten und eine übersichtliche und strukturierte Darstellung der Kopfdaten.

NetTool kann sowohl als Client, als auch als Relay eingesetzt werden, bietet jedoch im RelayModus keine Möglichkeit, den Datenstrom zu beeinflussen oder zu analysieren. Diese Software ist in Java geschrieben und böte sich für eine Erweiterung an, jedoch konnte die Lizenz der Software nicht eindeutig geklärt werden. Der Quellcode war leider nur in einer sehr alten Version verfügbar und auf Anfragen per Email hat der Autor bisher nicht reagiert. Da aber zeitnah eine Entscheidung zu treffen war und keines der untersuchten Produkte über das gewünschte Profil verfügt oder aber aus verschiedenen Gründen nicht erweiterbar ist, wird eine Eigenentwicklung der Erweiterung eines bestehenden Werkzeugs vorgezogen. Hierbei sollen jedoch die hervorstechenden Merkmale von Http Probe und NetTool als Vorlage dienen. Des Weiteren soll die Kopfdatenaufbereitung von Fiddler als Muster für eine übersichtliche Darstellung genutzt werden.

| Produkt | Lizenz | Betriebssystem | Sprache | Konzept | GUI | WebDAV | Einsatzzweck | Erweiterbar |
|---|---|---|---|---|---|---|---|---|
| Http Probe | keine Angabe | Windows | C# | Client | .NET | Ja | HTTP/ WebDAV-Debugger | Ja, Quellcode verfügbar |
| JMeter | Apache 2 | Windows, OS X, Linux, Unix | Java | Client | Swing | Nein | Server-Tester | Ja, Quellcode verfügbar |
| ntop | GNU GPL | Windows, OS X, Linux, Unix | C | Sniffer | Web | Nein | Netzwerkprotokollanalyse | Ja, Quellcode verfügbar |
| Wireshark | GNU GPL | Windows, OS X, Linux, Unix | C | Sniffer | MSVC, GTK | Nein | Netzwerkprotokollanalyse | Ja, Quellcode verfügbar |
| Charles | Shareware | Windows, OS X, Linux, Unix | Java | Proxy | Java | Nein | HTTP-Debugger | Nein |
| HttpWatch | Kommerziell | Windows | unbekannt | Browser--Plugin | Win32 | Nein | HTTP-Debugger | Nein, Automatisierungsschnittstelle |
| HTTP-Debugger | Kommerziell | Windows | unbekannt | Client | Win32 | Nein | HTTP-Debugger | Ja, SDK |
| Fiddler | Freeware | Windows | C# | Proxy | .NET | Nein | HTTP-Debugger | Nein, Automatisierungsschnittstelle |
| muffin | GNU GPL | Windows, OS X, Linux, Unix | Java | Proxy | AWT | Nein | Contentfilter Anonymisierer | Ja, Quellcode verfügbar |
| NetTool | k.A. / LGPL | Windows, OS X, Linux, Unix | Java | Relay / Client | Swing | teilweise | HTTP-Debugger | Ja, Quellcode verfügbar |
| Proximodo | GNU GPL | Windows, OS X, Linux, Unix | C++ | Proxy | wxWidgets | Nein | Contentfilter Anonymisierer | Ja, Quellcode verfügbar |
| Privoxy | GNU GPL | Windows, OS X, Linux, Unix | C | Proxy | Web | Nein | Contentfilter Anonymisierer | Ja, Quellcode verfügbar |

Tabelle 2.1: Übersicht Werkzeuge

# 3 Implementierungssprache

## 3.1 Anforderungen

Die Anforderungen an die Implementierungssprache ergeben sich zunächst einmal aus den Anforderungen an das Produkt. Für ein Open Source Projekt ist eine breite Nutzer- und Entwicklerbasis erwünscht. Von Vorteil hierfür sind eine verbreitete Programmiersprache und eine einfache Installation des Endprodukts sowie der Entwicklungsumgebung. Weiterhin ist darauf zu achten, dass die gewählte Programmiersprache eine einfache Umsetzung der Forderung nach Portabilität gewährleistet. Weiterhin sind eine gute Unterstützung für die Entwicklung von graphischen Benutzerschnittstellen, Bibliotheken für XML-Verarbeitung und die Netzwerkprogrammierung, hier im speziellen WebDAV gewünscht. Vorgabe war zunächst von den Programmiersprachen Java, Python und C/C++ und deren Kombination auszugehen.

## 3.2 Bewertung

### 3.2.1 Python

Python ist eine plattformunabhängige interpretierte Sprache, die Anfang der 1990er Jahre von Guido van Rossum entwickelt wurde. Python [Mar03] unterstützt verschiedene Programmierparadigmen. So kann der Entwickler objektorientiert, strukturiert oder auch funktional programmieren. Ziel bei der Entwicklung von Python war es eine einfache und übersichtliche Sprache zu schaffen, die trotzdem einen großen Funktionsumfang bietet. Dieses Ziel wird auch durch eine umfangreiche Standard-Bibliothek und

eine einfache Möglichkeit in anderen Programmiersprachen erstellte Module einzubetten untermauert. Vorteile von Python sind die Einfachheit der Sprache und die große Funktionalität der mitgelieferten Bibliotheken. Im DLR wird zur GUI-Entwicklung PyQt [PyQ07] verwendet, eine von der Firma Riverbank Computing Ltd erstellte Python-API zum Zugriff auf die Klassenbibliotheken von Qt der Firma Trolltech [Qt07]. Daraus ergeben sich allerdings auch folgende Nachteile: Eine Entwicklungsumgebung für zwei Sprachen (Python und Qt mit C++) ist komplexer einzurichten und auch das fertige Produkt erfordert mehr Installationsaufwand. Weiterhin sind die Thread-Bibliotheken in Python nach Aussage von Mitarbeitern des DLR und auch nach Recherche im Internet noch nicht so ausgereift wie in anderen Programmiersprachen. Zu guter Letzt gibt es momentan keine aktuelle und gepflegte WebDAV-Bibliothek für Python unter einer passenden Lizenz.

### 3.2.2 C/C++

Die Programmiersprache C wurde in den frühen 1970er Jahren von Dennis Ritchie entwickelt und steht auf fast allen Betriebssystemen zur Verfügung. Sie ist sehr verbreitet und kann für nahezu jedes Problem verwendet werden. In den 1980ern wurde diese Sprache von Bjarne Stroustrup [CPL97] zu C++ weiterentwickelt, die damit dann sowohl systemnahe, als auch abstrakte, wie etwa objektorientierte, Programmierung unterstützt. C/C++ ist weit verbreitet, bietet die Möglichkeit plattformunabhängige Software zu entwickeln und hat eine große Entwicklergemeinde. Weiterhin besticht die Sprache durch sehr schnelle Programme und eine große Auswahl an Bibliotheken. Im gewünschten Umfeld etwa existiert die Bibliothek lib-neon [LNE07], die WebDAV-Unterstützung bietet und von Projekten wie etwa davfs [DaF07] und catacomb [Cat07] verwendet wird. Zur plattformübergreifenden Programmierung von grafischen Oberflächen ließe sich etwa die Bibliothek Qt von Trolltech [Qt07] verwenden. Die Programmiersprache C/C++ bietet allerdings nicht nur Vorteile. So ist eine wirklich plattformunabhängige Programmierung ziemlich Aufwendig und für die Sprache C lassen sich Qualitätssicherungsmaßnahmen wie etwa Unit-Tests nur

umständlich realisieren. Auch ist die native Unterstützung für nebenläufige Programmierung nicht so weit fortgeschritten wie in anderen Sprachen.

### 3.2.3 Java

Die Programmiersprache Java [PIJ02] ist ein Teil der Plattform Java der Firma Sun Microsystems. Diese Plattform besteht aus der Programmiersprache Java selbst und verschiedenen Versionen der Laufzeitumgebung für die in Java geschriebene Programme [Sun07]. Die Laufzeitumgebungen sind mittlerweile für nahezu jedes Betriebssystem verfügbar. Die Sprache Java ist ähnlich weit verbreitet wie C++ und hat bei den Entwicklern eine hohe Akzeptanz erreicht. Gegenüber C++ hat Java unter anderem den Vorteil einer automatischen Speicherbereinigung. Java bietet eine einfache Möglichkeit plattformunabhängige Programme zu entwickeln, da die Programme von der jeweiligen Laufzeitumgebung interpretiert werden. Die Sprache ist vollständig objektorientiert und bietet sehr umfangreiche Bibliotheken für nahezu alle Problemfelder. Im Rahmen dieses Projektes von besonderem Interesse sind etwa Netzwerkprogrammierung, Thread-Programmierung und XML-Verarbeitung, die alle durch die Standardbibliotheken unterstützt werden. Weiterhin soll an dieser Stelle darauf hingewiesen werden, dass die WebDAV-Referenzimplementierung, das Jakarta-Jackrabbit-Projekt [AJR07], ebenfalls in Java entwickelt wird. Als Nachteil können die allgemeinen Nachteile von interpretierten Sprachen gegenüber kompilierten Sprachen betrachtet werden. So ist zum Beispiel die Ablaufgeschwindigkeit eines Java Programms langsamer als die eines C oder C++ Programms.

## 3.3 Fazit

Aufgrund der Entscheidung aus Kapitel 2 und der Bewertungsgrundlagen für die zu verwendende Programmiersprache ist die Entscheidung zu Gunsten der Sprache Java ausgefallen. Die Begründung dieser Entscheidung

wird im Folgenden dargelegt. So verfügt Java über eine weite Verbreitung und Akzeptanz sowohl bei Nutzern als auch Entwicklern. Dies ermöglicht eine unkomplizierte Verbreitung der fertigen Anwendung und eine einfache und kurze Einarbeitung für neue Entwickler. Das Look & Feel einer Java-Anwendung lässt sich zudem problemlos an das jeweilige Betriebsystem anpassen und bietet so eine nahtlose Integration in bestehende Umgebungen. Weiterhin ist die sehr große Verfügbarkeit von Bibliotheken ein großer Pluspunkt für diese Programmiersprache.

# Literaturverzeichnis


[AJM07] Apache JMeter
http://jakarta.apache.org/jmeter
*15. November 2007*

[AJP07] Apache Jakarta Projekt
http://jakarta.apache.org
*22. November 2007*

[AJR07] Apache Jackrabbit Projekt - WebDAV Library
http://jackrabbit.apache.org/doc/components/webdav.html
*3. Dezember 2007*

[APA07] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html
*15. Oktober 2007*

[Cat07] Catacomb WebDAV Server
http://www.webdav.org/catacomb/
*3. Dezember 2007*

[Cha07] Charles
http://www.xk72.com/charles
*15. November 2007*

[CPL97] Bjarne Stroustrup:
The C++ Programming Language
Addison-Wesley Longman, Amsterdam 1997

[DaF07] WebDAV Linux File System davfs2
http://sourceforge.net/projects/dav/
*3. Dezember 2007*

[ERW07] Ethereal is now Wireshark
http://www.wireshark.org/news/20060607.html
*22. November 2007*

[Fid07] Fiddlertool
http://www.fiddlertool.com/fiddler
*15. November 2007*

[GPL07] GNU GPL
http://www.gnu.org/licenses/gpl.html
*22. November 2007*

[HTD07] HTTP Debugger
http://www.httpdebugger.com/index.html
*15. November 2007*

[HTP07] HttpProbe
http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=FEA72F64-458E-4293-8FE7-619CC28A5DD7
*15. November 2007*

[HTW07] HttpWatch
http://www.httpwatch.com
*15. November 2007*

[LNE07] neon
http://www.webdav.org/neon/
*3. Dezember 2007*

[Mar03] Martelli, Alex:
Python in a Nutshell
O'Reilly & Associates Inc., Sebastopol 2003

[MSF07] muffin Projektseite bei Sourceforge
http://sourceforge.net/projects/muffin
*23. November 2007*

[Muf07] Muffin
http://muffin.doit.org
*15. November 2007*

[MVP07] Microsoft MVP
https://mvp.support.microsoft.com/gp/mvpexecsum
*22. November 2007*

[NEm07] Email an den Autor von NetTool
*21. November 2007*

[Net07] NetTool
http://nettool.sourceforge.net
*15. November 2007*

[NTo07] ntop
http://www.ntop.org
*15. November 2007*

[PIJ02] Jobst, Fritz:
Programmieren in Java
Hanser, München/Wien 2002

[Pri07] Privoxy
http://www.privoxy.org
*20. November 2007*

[Pro07] Proximodo
http://sourceforge.net/projects/proximodo
*20. November 2007*

[PrS07] Privoxy Projektseite bei Sourceforge
http://sourceforge.net/projects/ijbswa
*23. November 2007*

[PSF07] Proximodo Projektseite bei Sourceforge
http://sourceforge.net/projects/proximodo
*23. November 2007*

[PyQ07] PyQt - Python Qt-Bibliothek
http://www.riverbankcomputing.co.uk/pyqt/index.php
*3. Dezember 2007*

[PWi07] Proxomitron und Proximodo in Wikipedia
http://de.wikipedia.org/wiki/Proxomitron
*23. November 2007*

[Qt07] Trolltech Qt Framework
http://trolltech.com/products/qt
*3. Dezember 2007*

[Sun07] Sun Java Technology
http://www.sun.com/java/about/
*3. Dezember 2007*

[WiS07] Wireshark
http://www.wireshark.org
*19. November 2007*

# Glossar

**ACL** Engl. Abkürzung für Access Control List. ACLs sind Listen, die Rechte von Benutzern auf Objekte speichern.

**Bug-Tracker** Software zur Verfolgung und Dokumentation von Fehlern in Software.

**Catacomb** Catacomb ist eine Erweiterung für das WebDAV-Modul des Apache Webservers. Die Daten werden hierbei statt im Dateisystem in einer relationalen Datenbank abgelegt.

Siehe auch: http://catacomb.tigris.org/

**Checkstyle** Checkstyle ist ein Werkzeug für Entwickler mit dem man Quelltext nach formalen Kriterien überprüfen kann.

Siehe auch: http://eclipse-cs.sourceforge.net/

**DASL** Engl. Abkürzung für DAV Searching and Locating. DASL befasst sich mit der serverseitigen Suche in WebDAV-Repositories. Die Arbeitsgruppe zu diesem Standard wurde allerdings mittlerweile aufgelöst.

**DataFinder** DataFinder ist eine Applikation auf Clientseite zur Verwaltung technisch-wissenschaftlicher Daten. Hierbei können die Daten sowie die Metadaten über verschiedenartige Speicherschnittstellen auf einem oder mehreren Servern abgelegt werden.

Siehe auch:
http://www.dlr.de/sc/desktopdefault.aspx/tabid-1189/1650_read-3140/

**DeltaV** Engl. Web Versioning and Configuration Management. Dieses Protokoll ist eine Erweiterung des WebDAV-Protokolls um Versions- und Konfigurationsmanagement.

**DLR** Deutsches Zentrum für Luft- und Raumfahrt.

Siehe auch: http://www.dlr.de

**Eclipse** Eclipse ist ein Framework zur Entwicklung von Anwendungen und ist vergleichbar mit einer IDE, kann aber auch noch wesentlich mehr bieten, da es durch eine entsprechende Architektur einfach erweiterbar ist.

**IDE** Engl. Abkürzung für Intergrated Developement Environment. Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software, das mehrere Komponenten in einer Oberfläche zusammenfasst.

**Loopback** Eine Loopback Netzwerkschnittstelle ist ein lokaler Informationskanal mit nur einem Endpunkt. Sender und Empfänger sind hierbei identisch.

**Mailingliste** Eine Mailingliste ist ein E-Mail-Verteiler für eine Gruppe von Menschen. Einzelne Mailinglisten haben häufig bestimmte Themen und dienen zur Kommunikation und Information.

**Makro** Ein Makro ist in der Programmierung eine vorgegebene Sequenz von Befehlen oder Aktionen. In diesem Zusammenhang bezeichnet ein Makro eine aufgezeichnete Kommunikationssequenz zwischen zwei WebDAV-Applikationen.

**MVC** Engl. Model-View-Controller. MVC ist ein zusammengesetztes Architekturmuster zur Gestaltung von Anwendungen. Um eine bessere Wiederverwendbarkeit und einfachere Erweiterbarkeit zu erhalten erfolgt eine Aufteilung in die Elemente Datenmodell (Model), Präsentation (View) und Steuerung (Controller).

**Open Source** Open Source Software ist quelloffene Software. Das heißt, der Quellcode dieser Software ist frei zugänglich und deren Bearbeitung und Verbreitung erlaubt und erwünscht. Es gibt verschiedene Lizenzmodelle, die eine genaue Definition der Verwendung und Verbreitung der jeweiligen Software regeln.

**Plugin** Ein Plugin ist ein Software- oder Hardwaremodul, welches in eine bestehende Software oder Hardware eingebunden wird und die Funktionalität der Software oder Hardware erweitert.

**Proxy** Ein Proxy ist im Zusammenhang mit Netzwerken ein Dienstprogramm zur Steuerung von Netzwerkverkehr. Ein Proxy besitzt sowohl eine ServerSchnittstelle als

auch eine Client-Schnittstelle. Im einfachsten Falle leitet der Proxy den Netzwerkverkehr unbeeinflusst weiter. In diesem Fall spricht man auch von einem „Relay“.

**RFC** Engl. Requests for Comments. Die RFCs sind eine Sammlung von technischen und organisatorischen Dokumenten zu den im Internet verwendeten Protokollen und Verfahren.

**SC** Abkürzung für die Abteilung Simulations- und Softwaretechnik innerhalb des DLR.

Siehe auch: http://www.dlr.de/sc/

**sourceforge.net** sourceforge.net ist eine Online-Plattform. Hier werden einem oder einer Gruppe von Entwicklern Werkzeuge zur Unterstützung bei der Anwendungsentwicklung und zur Projektverwaltung geboten.

Siehe auch: http://www.sourceforge.net/

**Subclipse** Eclipse-Plugin für die Versionsverwaltung Subversion.

**Subversion** Open Source Software für die Versionsverwaltung von Dateien und Ordnern.

Siehe auch: http://subversion.tigris.org/

**TCP** Engl. Abkürzung für Transmission Control Protocol. TCP ist ein verbindungsorientiertes Transportprotokoll der OSI-Schicht 4.

**UML** Engl. Abkürzung für Unified Modeling Language. Eine von der Object Management Group entwickelte Sprache zur Modellierung von Software.

**Unit-Test** Der Unit-Test wird auch als Modultest bezeichnet und dient zur Überprüfung der Korrektheit von Programmteilen.

**WebDAV** Engl. Abkürzung für Web-based Distributed Authoring and Versioning. WebDAV ist ein offener Standard zur Bereitstellung von Dateien im Internet und ist eine Weiterentwicklung des HTTP-Protokolls.

Siehe auch: http://www.webdav.org/

**Wiki** Ein Wiki ist eine Sammlung von Hypertextseiten, die von Benutzern gelesen, editiert und verknüpft werden können.

# E Pflichtenheft


Elektrotechnik und Informatik

Diplomarbeit

# Entwicklung eines grafikunterstützten Debugging-Werkzeugs zur Implementierung von WebDAV-Applikationen

Pflichtenheft

Revision 0.6

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Das deutsche Zentrum für Luft- und Raumfahrt ist eine Forschungseinrichtung der Bundesrepublik Deutschland mit den Schwerpunkten Luftfahrt, Raumfahrt, Energie und Verkehr. Das Spektrum der Forschungen reicht von der Grundlagenforschung bis hin zu fertigen Produkten und Anwendungen. Hierbei werden sowohl nationale, internationale als auch industrielle Kooperationen umgesetzt.

Wissenschaftliche Einrichtungen müssen in der Regel eine große Menge von Daten verwalten. Hierbei sind üblicherweise sowohl die Menge der Daten als auch die vielen verschiedenen Formate und Prozesszugehörigkeiten der Daten problematisch. Zu einem Experiment gehören zum Beispiel Dokumentationen der verschiedenen Arbeitsschritte, Eingabe- oder Messdaten, Modelle sowie Ergebnisdaten. Zusätzlich können die Daten noch in verschiedenen Versionen vorliegen. Für das Management dieser Daten wird im DLR auf Clientseite die Anwendung DataFinder auf Grundlage des WebDAV-Protokolls eingesetzt. Als Server kann dafür jede Server-Software eingesetzt werden, welche die WebDAV-Spezifikation [RFC4918] hinreichend implementiert. Neben kommerziellen Servern (z. B. Tamino der Software AG oder SharePoint von Microsoft) werden zunehmend Open Source Produkte eingesetzt. Im DLR wird dafür am Open Source WebDAV-Server Catacomb mitentwickelt. Für das Entwickeln des Servers und des Clients sind genaue und automatisierte Tests eine wichtige Hilfe.

Im Rahmen der Diplomarbeit soll ein Open Source Werkzeug realisiert werden, mit dessen Hilfe WebDAV-Anwendungen grafikunterstützt getestet und ausgewertet werden können. Dabei soll das Werkzeug als Proxy

zwischen Client und Server fungieren und somit den Datenverkehr auswerten, um eine Analyse der Kommunikation zwischen Server und Client zu ermöglichen. Weiterhin sollen verschiedene Berichte generiert sowie vorher geplante Tests durchgeführt werden können.

Die gestellte Aufgabe ist innerhalb einer sechsmonatigen Diplomarbeit im Rahmen des Studiums „Angewandte Informatik“ mit Anwendungsfach Elektrotechnik an der Universität Siegen zu bearbeiten. Betreut wird die Arbeit auf universitärer Seite durch die Fachgruppe Technische Informatik, geleitet von Uni.-Prof. Hans Wojtkowiak. Ansprechpartner ist Dr.-Ing. Bernd Klose. Ansprechpartner in der Einrichtung „SC – Verteilte Systeme und Komponentensoftware“ des DLR sind Abteilungsleiter Andreas Schreiber und Dipl.-Inform. Markus Litz.

## 1.2 Inhaltsübersicht

In Kapitel 2 wird definiert, welche Funktionalitäten das Produkt bieten soll, welche optional und welche nicht gefordert sind. Kapitel 3 beschreibt die Einsatzumgebung des fertigen Produktes. In Kapitel 4 werden die Kernfunktionen des Produktes kurz erläutert und in Kapitel 5 erfolgt dann eine ausführliche Darstellung der geforderten Leistungsmerkmale. Kapitel 6 beschreibt die zum Produkt gehörenden Daten und Datenströme. Kapitel 7 erläutert die Anforderungen an die grafische Schnittstelle des Produktes und deren wichtigste Elemente. Kapitel 8 enthält die nichtfunktionalen Anforderungen. Kapitel 9 legt die Qualitätskriterien für das Produkt fest. Kapitel 10 beschreibt die zur Entwicklung verwendeten Werkzeuge und Hilfsmittel. Kapitel 11 beinhaltet ergänzende Bemerkungen zu dem Produkt und dessen Entwicklung. In Kapitel 12 erfolgt eine Aufteilung der Entwicklung in verschiedene Arbeitspakete und eine grobe zeitliche Planung. Den Abschluss des Pflichtenheftes bilden das Literaturverzeichnis und Glossar.

# 2 Zielbestimmung

## 2.1 Musskriterien

In diesem Kapitel werden die Funktionalitäten beschrieben, die die Software bieten soll. Der Benutzer soll den Datenverkehr unverändert weiterleiten können und somit nur die Rolle eines Beobachters einnehmen können. Der Benutzer soll aber auch die Möglichkeit haben, in den Datenverkehr einzugreifen und Veränderungen vorzunehmen. Weiterhin soll der Datenverkehr aufgezeichnet werden können. Aus diesen Aufzeichnungen können sogenannten Makros erstellt werden. Makros sollen jedoch auch manuell editiert werden können.

Der Datenverkehr zwischen WebDAV-Anwendungen soll durch Farbgebung, Trennung von Anfrage (Request) und Antwort (Reply) übersichtlich dargestellt werden. Weiterhin soll eine getrennte Anzeige von Kopf- und Rumpfdaten (Head- und Bodydaten) die Übersicht verbessern. Um den Ablauf der Kommunikation sichtbar zu machen, soll eine Verlaufsansicht des aufgezeichneten Datenverkehrs genutzt werden. Die Anwendung soll so ausgelegt sein, dass eine spätere Erweiterung problemlos möglich ist. Im Besonderen ist eine Plugin-Schnittstelle vorzusehen, die eine Erweiterung der Anzeige- und Editiermöglichkeiten ohne Änderungen am Quellcode des Produktes erlaubt

## 2.2 Wunschkriterien

Es wäre wünschenswert, wenn die Software diese Kriterien erfüllt. Dies ist abhängig davon, ob noch genügend Zeit innerhalb des Zeitrahmens der Diplomarbeit zur Verfügung steht. So wäre ein Assistent zum geführten Zusammenstellen von Makros - und damit WebDAV-Kommandos

- eine große Erleichterung für den Benutzer. Auch weitere Plugins zur Kommunikationsanalyse für z. B. ACL, DASL oder DeltaV sind gewünscht.

## 2.3 Abgrenzungskriterien

Die Abgrenzungskriterien dienen zur Verdeutlichung der Funktionen, die das Produkt nicht leisten soll. So ist kein automatisierter Test von WebDAV-Software gefordert. Ebenso soll keine Analyse des WebDAV-Protokolls und dessen Erweiterungen stattfinden.

# 3 Produkteinsatz

## 3.1 Anwendungsbereiche

Für die Entwicklung WebDAV-fähiger Softwareprodukte ist es wichtig mit wenig Aufwand die erstellten Produkte zu testen um Fehler einfach finden und gegebenenfalls reproduzieren zu können. Das zu entwickelnde Werkzeug soll genau diese Möglichkeiten bieten und von Entwicklern und Testern dazu eingesetzt werden, die Kommunikation eines Clients und Servers über das WebDAV-Protokoll zu speichern, auszuwerten und gegebenenfalls wiederholen zu können.

Weiterhin soll die Software in der Lage sein die Gegenseite des jeweiligen Kommunikationspartners durch manuell zu erstellende oder abgespeicherte Kommunikationssequenzen (im Folgenden als Makros bezeichnet) zu emulieren. Hierbei soll zunächst der Schwerpunkt bei der Emulation des Clients liegen.

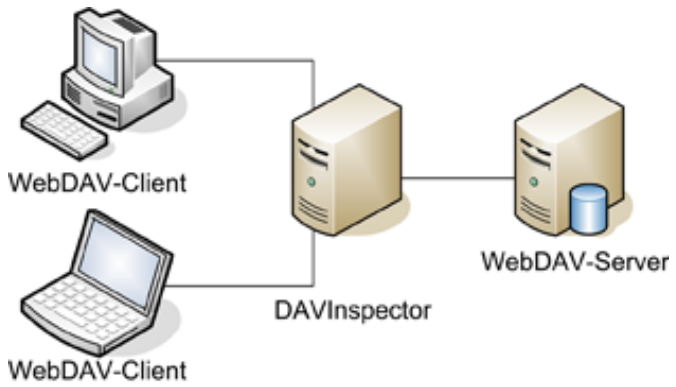


Bild 3.1: Einsatzszenario des DAVInspectors

In Bild 3.1 ist die Einsatzumgebung des zu erstellenden Werkzeugs mit dem Namen DAVInspector dargestellt. Die Kommunikation zwischen den

WebDAV-Clients und dem WebDAV-Server wird hierbei über den DAVInspector geleitet und kann dort nach Wunsch manipuliert und ausgewertet werden. Hierbei handelt es sich allerdings nur um eine schematische Darstellung. In der Praxis können die Komponenten alle auf einem physikalischen Rechner installiert sein oder aber auch auf zwei oder drei Rechner verteilt sein.

## 3.2 Zielgruppe

Zielgruppe für die Anwendung sind WebDAV-Applikationsentwickler. Dies umfasst sowohl Entwickler von Client-, als auch Serverapplikationen. Das Produkt ist nicht für den Einsatz in produktiven Umgebungen gedacht, sondern ist ein Werkzeug für Entwickler. Es soll das Testen und die Suche, sowie die Nachvollziehbarkeit von Fehlern vereinfachen.

# 4 Produktübersicht

In diesem Kapitel werden die Vorgaben beschrieben, an denen sich die Realisierung der Software orientieren wird. Um eine gute Architektur und damit Wiederverwendbarkeit der Software gewährleisten zu können, wird das Gesamtsystem partitioniert. Die Aufteilung erfolgt dabei zunächst dem Model-View-Controller-Muster (MVC-Pattern). Dieses Muster sorgt für eine Trennung in Datenmodell, Steuerung und Darstellung und vereinfacht spätere Änderungen oder Erweiterungen der Software. Das Datenmodell repräsentiert die darzustellenden Daten der Applikation und kann zusätzlich auch Logik enthalten. Das Modell ist von den beiden übrigen Komponenten unabhängig. Änderung in der Darstellung nimmt die Steuerung entgegen und ruft die passenden Funktionen im Modell auf um die gewünschten Änderungen durchzuführen. Die Steuerung kann ebenfalls Einfluss auf die Darstellung nehmen. Die Darstellung ist die Präsentation der Daten aus dem Modell in einer für den Benutzer sinnvollen Form. Interaktionen des Benutzers werden von der Darstellung an die Steuerung weitergeben.

Bild 4.1 zeigt den Entwurf der benötigten Klassen und Schnittstellen in einem UML-Diagramm. Hierbei ist zu beachten, dass, um eine besser Übersicht zu bieten, dieses Diagramm nicht vollständig ist. Einzelne Klassen, zum Beispiel Hilfsklassen für die Darstellung, fehlen. Auch sind keine privaten Attribute oder Operationen aufgeführt. Als zentrale Komponente lässt sich unschwer das „RelayModel“ ausmachen. Diese Klasse bildet zusammen mit den Klassen „MainController“ und „MainView“ das oben aufgeführte MVC-Muster ab. Weitere wichtige Klassen sind die Klasse „Connection“, die die Daten auf Netzwerkebene verarbeitet, die Klasse „PluginManager“ zur Verwaltung und Nutzung der Plugins, sowie die Klasse „MessageHistory“, die für die Speicherung des Verlaufs des Nachrichten-

austausches und die weitere Auswertung einzelner Nachrichten zuständig ist.

Eine weitere Strukturierung der Software wird durch eine Aufteilung der Klassen des Projektes in verschiedene Pakete erreicht. Diese Maßnahme ermöglicht ein übersichtlicheres Design und bietet die Grundvoraussetzung für eine einfache Wiederverwendbarkeit von einzelnen Komponenten.

In Bild 4.2 ist die für den DAVInspector vorgesehene Partitionierung in verschiedene Pakete dargestellt. Im Paket „UI“ werden alle Klassen abgelegt, die für die grafische Benutzerschnittstelle benötigt werden. In dem Paket „Relay“ liegen die Klassen, die die Kernfunktionalität der Anwendung bilden. Von mehreren Teilen gemeinsam genutzte Klassen befinden sich im Paket „Common“. Klassen, die für die Konfiguration der Anwendung benötigt werden, befinden sich im Paket „Config“. Das Paket „Plugin“ enthält alle Pakete, die zur Verwaltung und Nutzung von Plugins benötigt werden. Im Paket „History“ befinden sich alle Klassen um die Verlaufsanzeige zur Visualisierung der Kommunikationssequenzen zu realisieren. Das Paket „plugins“ wiederum enthält konkrete Implementierungen von Plugins. Exemplarisch sind hier die drei Plugins „XMLView“, „XMLTree“, „RawEdit“, „Recording“ und „HeaderView“ abgebildet. Diese befinden sich wiederum in jeweils eigenen Paketen. Das Paket „HTTP“ enthält die für den eingesetzten HTTP-/WebDAV-Parser benötigten Klassen und Ressourcen.

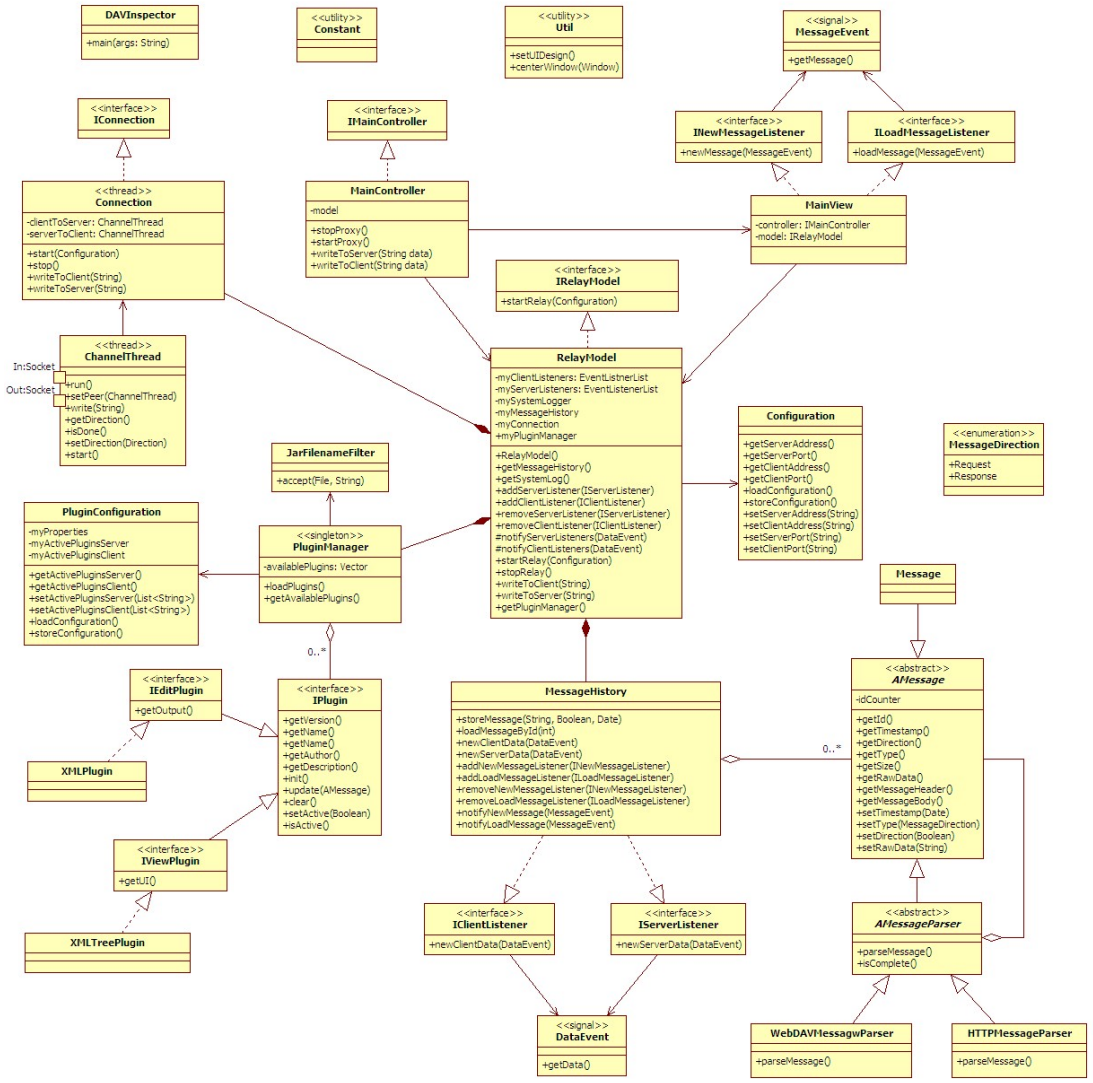


Bild 4.1: UML-Diagramm DAVInspector

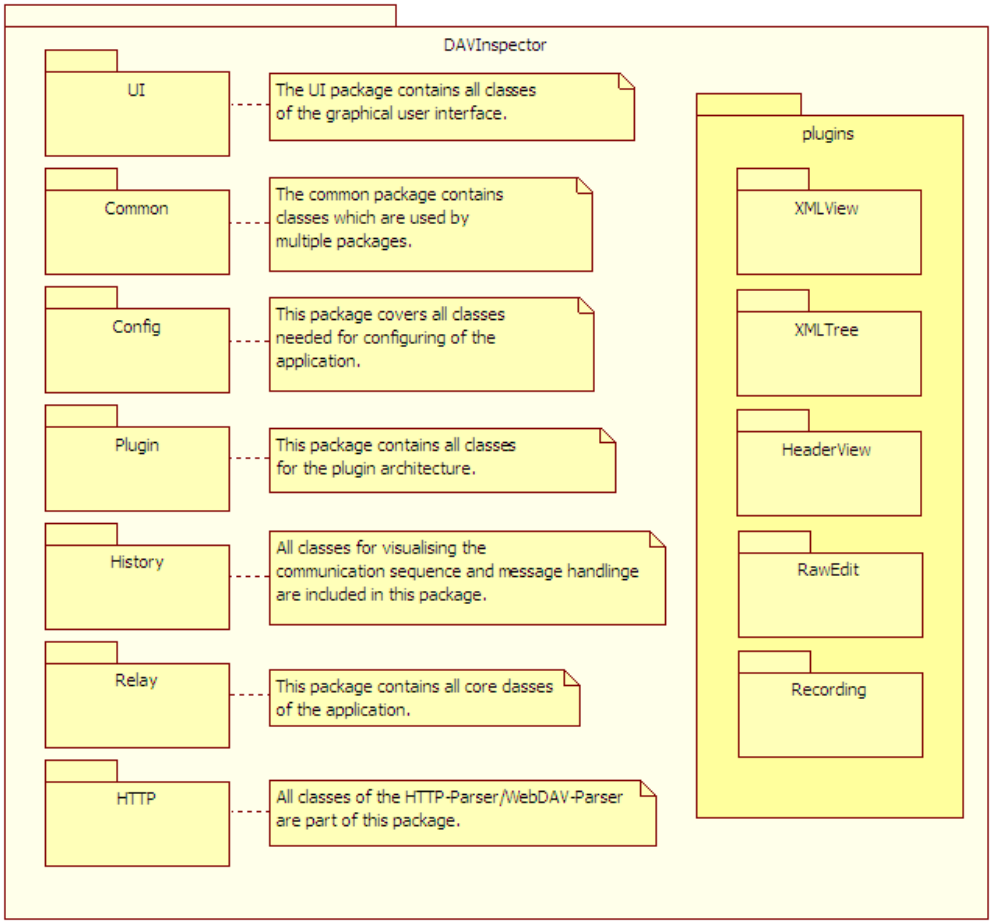


Bild 4.2: Paketstruktur DAVInspector

# 5 Produktfunktionen

In diesem Kapitel werden die zu implementierenden Funktionen detailliert erläutert.

## 5.1 /PF010/ – Abspeichern von Datenverkehr

Der von der Applikation weitergeleite Datenverkehr soll in einer Datei abgespeichert werden. Diese Datei kann später zum Erstellen eines Makros dienen oder gegebenenfalls zu weiteren Analysen durch andere Werkzeuge herangezogen werden.

## 5.2 /PF020/ – Weiterleitung von Datenverkehr

Wie bereits zuvor angedeutet ist eine wichtige Funktion der Anwendung die unveränderte Weiterleitung des Datenverkehrs. Diese Weiterleitung geschieht nach dem Prinzip eines Relais auf Schicht vier des OSI-Referenzmodells, also auf Ebene des TCP-Protokolls. Die einzelnen Datenpakete müssen hierbei durch manuellen Eingriff des Nutzers zum jeweiligen Kommunikationspartner weitergeleitet werden.

In Bild 5.1 und Bild 5.3 ist das Zustandsdiagramm für den Ablauf bei der Durchleitung von Netzwerkverkehr zu sehen. Die blau hinterlegten Elemente stellen die Verbindung zum Client dar, die orange hinterlegten verdeutlichen die serverseitige Verbindung. Das Diagramm zeigt die Abfolge der Zustände für eine Verbindung. Der Datenfluss wird dabei durch die grauen Pfeile symbolisiert.

Entgegen des eigentlichen Ziels die Datenpakete unverändert weiterzuleiten, ist es bei bestimmten Übertragungseinstellungen nötig die erhaltenen Daten zu verändern. Dieser Fall tritt vor allem bei Antworten des Servers, die auf mehrere Datenpakete aufgeteilt sind, auf. Der Eingriff in diese Daten ist unumgänglich um die im nächsten Abschnitt geforderte Funktionalität realisieren zu können. Dabei werden die empfangenen Daten gepuffert und anschließend in einem Paket an den Client weitergeleitet. Dies ist natürlich eine Unterbrechung des Datenstroms, der je nach Dauer der Unterbrechung zu einer Zeitüberschreitung bei einem der Teilnehmer führen kann.

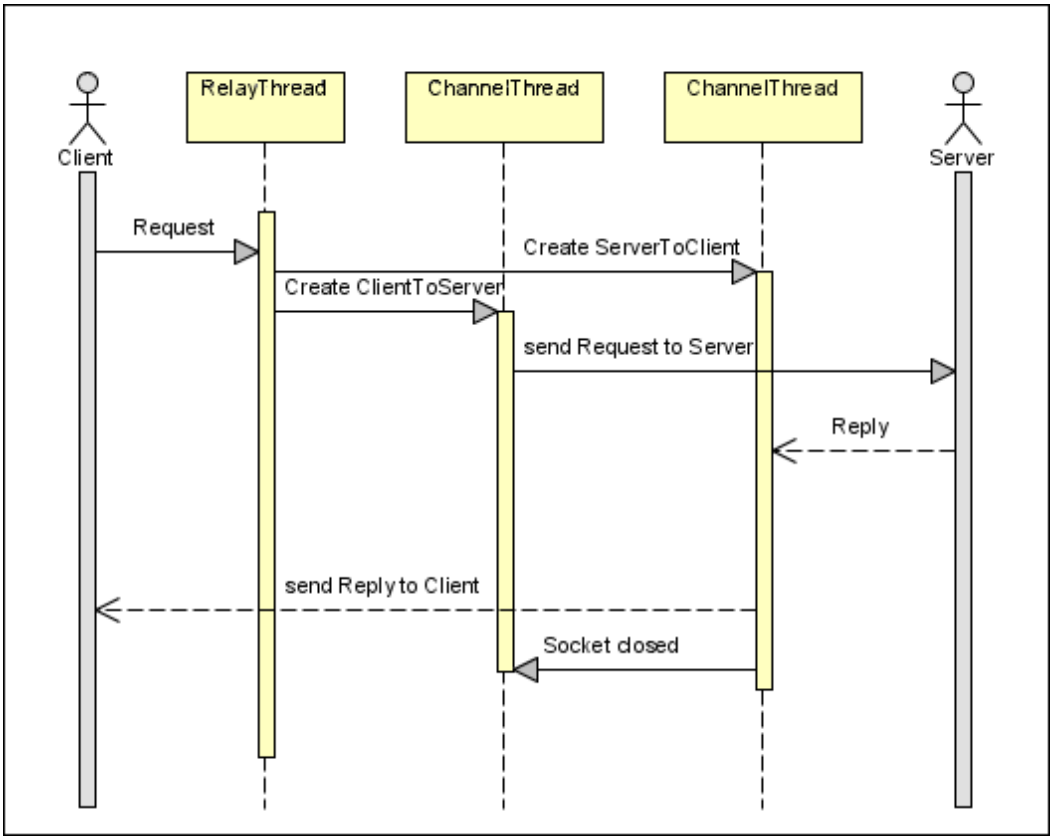


Bild 5.1: Weiterleitung von Datenverkehr, Sequenzdiagramm

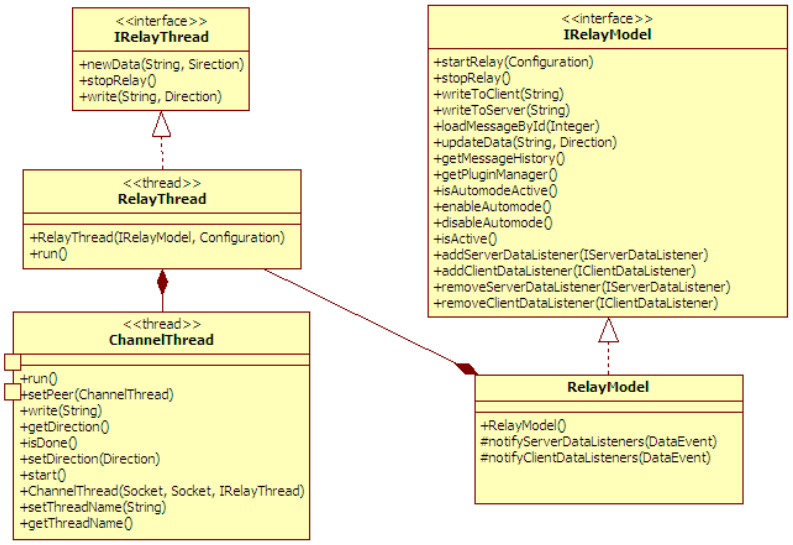


Bild 5.2: Klassendiagramm Relay

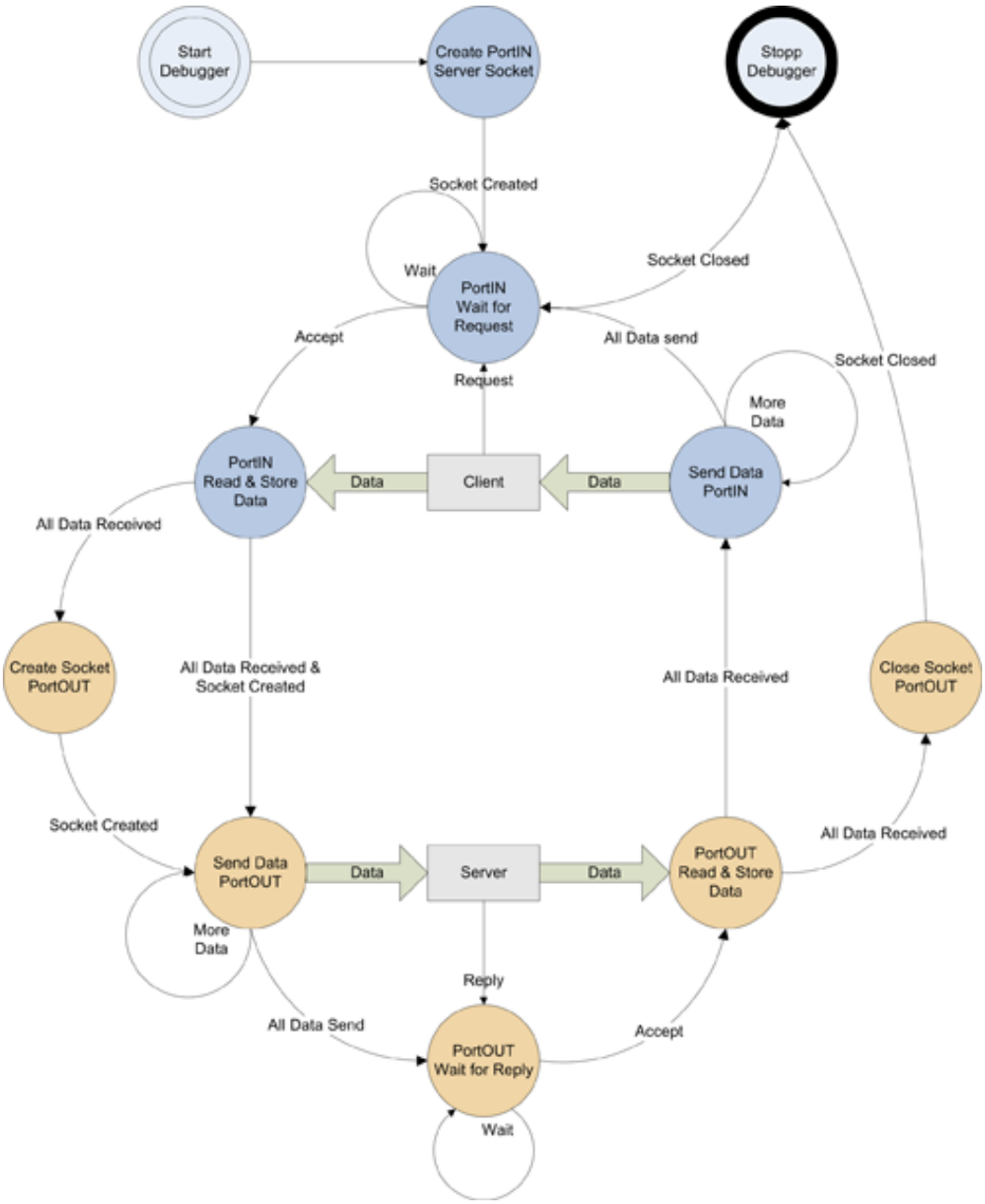


Bild 5.3: Weiterleitung von Datenverkehr, Zustandsdiagramm

## 5.3 /PF030/ – Eingriff in den Datenverkehr

Ein wichtiges Ziel der Entwicklung ist die Möglichkeit den Datenverkehr zur Laufzeit verändern zu können. Die Abfolge der dabei nötigen Zustände ist in Bild 5.4 dargestellt. Hierbei gelten dieselben Anmerkungen für die farbliche Kennzeichnung von Client- und Serverseite wie für Bild 5.3, also die Zustände für die Verbindung zum Client in blau und die Zustände für die Verbindung zum Server in orange. Der Datenfluss wird durch die grauen Pfeile symbolisiert.

Im Vergleich zu Bild 5.3 sind zwei Zustände hinzugekommen. Diese beiden Zustände markieren die Stellen, an denen in den Datenverkehr eingegriffen wird. Hier befinden sich die Schnittstellen zu den verschiedenen Anzeige- und Manipulationsmöglichkeiten, die in der Software vorhanden sein sollen oder per Plugin nachgerüstet werden können.

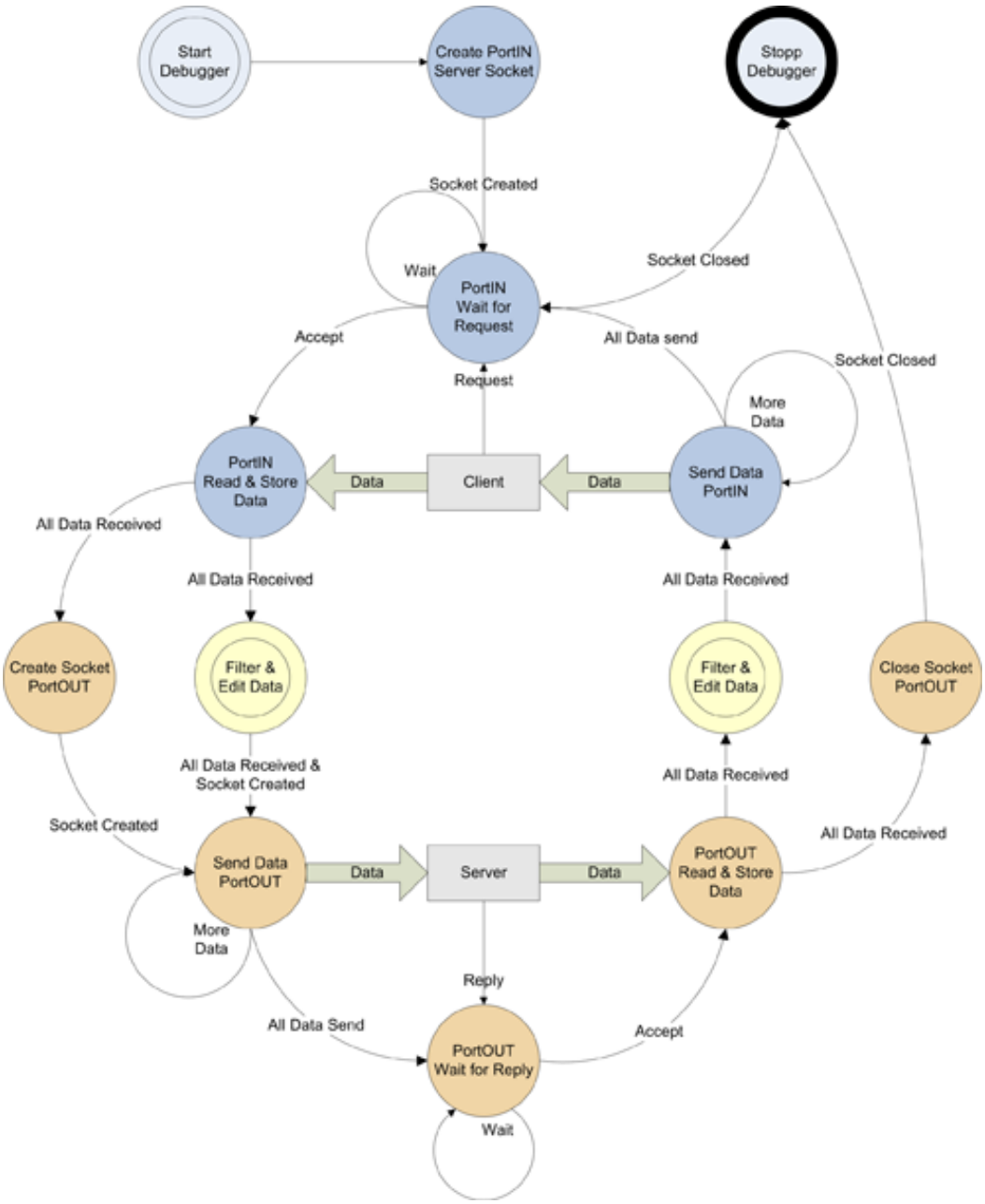


Bild 5.4: Eingriff in den Datenverkehr

## 5.4 /PF040/ – Unterscheidung von Anfrage und Antwort

Eine übersichtliche Darstellung des Datenverkehrs erfordert eine getrennte Anzeige von Anfrage (Request - Clientseite) und Antwort (Reply - Serverseite). Die Unterscheidung erfolgt durch die Definition der Nachrichtentypen nach RFC 2616 [RFC2616, Kapitel 4.1]. Ebenfalls in RFC 2616 [RFC2616, Kapitel 1.3] sind die Rollen von Client und Server definiert. So ist ein Client demnach ein Anwendungsprogramm, das Netzwerkverbindungen aufbaut um Anfragen abzusetzen. Ein Server hingegen ist eine Applikation, die Netzwerkverbindungen annimmt und gestellte Anfragen beantwortet. Eine Software kann sowohl die Client- als auch Server-Rolle übernehmen. Die jeweilige Rolle hängt dann von der konkreten Verbindung ab.

Daten, die vom Client zum Server gesendet werden, werden demnach als Anfragen bezeichnet. Daten, die in umgekehrter Richtung, also vom Server zum Client, versendet werden, werden als Antwort bezeichnet.

## 5.5 /PF050/ – Unterscheidung von Kopf- und Rumpfdaten

Für eine weitere Auswertung des Datenverkehrs ist es auch gewünscht die Kopf- (Head) und Rumpfdaten (Body) getrennt voneinander betrachten zu können. Die Daten des Rumpfes werden dabei durch eine Leerzeile von denen des Kopfes getrennt. Dies ist für HTTP- und WebDAV-Nachrichten in RFC 2616 [RFC2616, Kapitel 4.1] für HTTP und RFC 4918 [RFC4918] für WebDAV definiert. In Listing 5.1 ist der allgemeine Aufbau einer HTTP- oder WebDAV-Nachricht dargestellt. Diese besteht aus einem Kopfteil, der mindestens eine Anfragezeile oder Statuszeile enthält. Optional können mehrere weitere Zeilen folgen, die zusätzliche Kopfdaten

enthalten. Der Kopf wird dann vom Rumpf durch eine leere Zeile mit einem Zeilenumbruch getrennt. Danach folgt der ebenfalls optionale Rumpfteil.

```
1 Message = Request-Line | Status-Line
2           *(Message-Header CRLF)
3           CRLF
4           [ Message-Body ]
```

Listing 5.1: Allgemeine HTTP-/WebDAV-Nachricht nach RFC 2616

## 5.6 /PF070/ – Makro manuell Erstellen

Ein Makro soll manuell erstellt und bearbeitet werden können. Dabei sollten die XML-Daten des Rumpfes, sofern vorhanden, vor dem abspeichern des Makros validiert werden. Vor dem Abspeichern ist vom Benutzer noch eine Bezeichnung für das Makro zu vergeben und ein Zuordnung zur Client- oder Serverseite des Datenverkehrs.

## 5.7 /PF080/ – Makro aus Datenverkehr erstellen

Bei einem aus Datenverkehr zu erstellendem Makro werden die Daten einer Datenrichtung in dem dafür vorgesehenen Format abgespeichert. So kann der Nutzer einen Kommunikationspartner, Client oder Server, auswählen und dessen bis zu diesem Zeitpunkt aufgezeichneten Datenverkehr abspeichern. Diese Funktionalität wird über das Kontextmenü der Rohdatenansichten von Server und Client angeboten. Hierbei kann der Benutzer noch einen Namen für das abzuspeichernde Makro angeben oder den Standard-Dateinamen verwenden. Dieser besteht Richtung der Nachricht und aktuellem Datum und Uhrzeit (Beipiel: *msg_client_2008-04-09_12-12-10.rec*). Das Datenformat wird in Kapitel 6.3 auf Seite 27 erläutert.

## 5.8 /PF090/ – Simulation von Kommunikation

Durch aufgezeichnete Kommunikation kann einer der beiden Kommunikationspartner simuliert werden. Beispielsweise können die aufgezeichneten Kommunikationssequenzen eines Clients dazu benutzt werden, einem Server wiederholt mit derselben Anfrage zu testen. Bei der Wiedergabe finden zunächst keine Aktualisierungen der in der Anfrage vorhandenen Daten, wie etwa Zeitangaben und ähnliches, statt. Dies kann in späteren Versionen ergänzt werden und ist nicht Teil dieser Arbeit.

## 5.9 /PF100/ – Makro mit Assistenten erstellen

Zur Vereinfachung der Erstellung von Makros dient ein Assistent zum geführten Zusammenstellen. Hierbei wird der Nutzer Schritt für Schritt bei der Erstellung einer Anfrage oder einer Antwort durch Vorschläge und Hilfestellungen von der Software unterstützt.

## 5.10 /PF110/ – Erweiterbarkeit durch Filter/Plugins

Da im Zuge dieser Arbeit nur Basisauswertungen erstellt werden können, ist für die zu erstellende Software eine Architektur vorzusehen, die die Erweiterbarkeit der Kommunikationsanalyse durch Plugins erlaubt. Das Klassendiagramm dieser Architektur ist in Bild 5.5 dargestellt. Die Plugins sind unterteilt in zwei verschiedene Typen:

- Auswertende Plugins: Hierbei wird eine Kopie des Datenverkehrs an das jeweilige Plugin weitergeben, welches dann unabhängig von anderen Plugins die Daten auswerten, filtern oder anderweitig verarbeiten kann.
- Manipulierende Plugins: Hierbei wird der eigentliche Datenstrom durch dieses Plugin geleitet und bietet somit die Möglichkeit die Daten vor dem weitersenden zu verändern. Sind mehrere Plugins dieses

Typs vorhanden, werden die Daten nacheinander von Plugin zu Plugin geleitet. Die Reihenfolge kann dabei nicht durch den Benutzer beeinflusst werden.

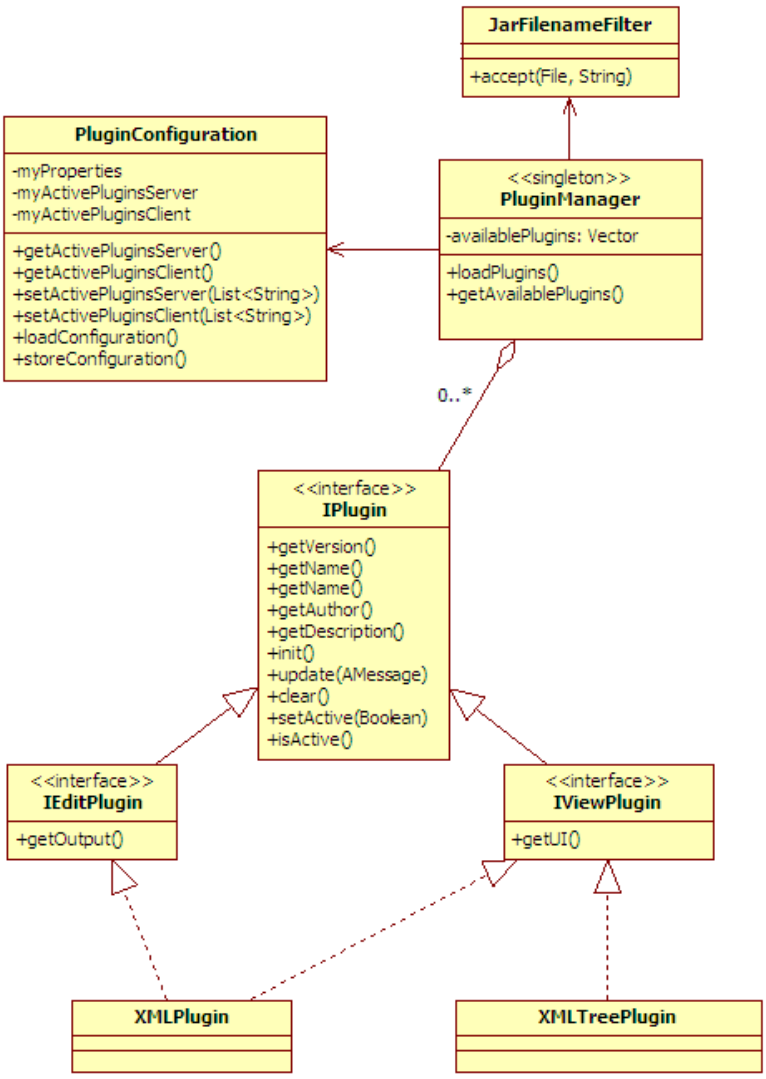


Bild 5.5: Klassendiagramm Plugin-Manager

## 5.11 /PF120/ – Automodus

Anders als in Kapitel 5.2 beschrieben ist in dieser Betriebsart kein Eingriff des Benutzers für die Weiterversendung der Datenpakete nötig. Ein einfacher, selbtsentwickelter HTTP-/WebDAV-Parser versucht vollständige Pakete zu erkennen und diese an den entsprechenden Kommunikationsendpunkt weiterzuleiten.

## 5.12 /PF130/ – Haltepunkt

Die Funktion Haltepunkt (engl. Breakpoint) ist nur in Verbindung mit dem Automodus sinnvoll einsetzbar. Sie soll es dem Benutzer ermöglichen eine Zeichenkette zu spezifizieren, die das Programm veranlasst bei Auftreten dieser Zeichenkette den Automodus zu verlassen. Hierbei wird auf manuellen Betrieb, beschrieben in Kapitel 5.2, umgeschaltet. Eine mögliche Realisierung dieser Funktionalität ist als ein erweitertes Plugin vorzusehen.

## 5.13 /PF140/ – Internationalisierung

Mit den in Java vorhandenen Bibliotheken „java.util.Local“ und „java.util.ResourceBundle“ soll die Basis für eine mögliche Internationalisierung der Applikation geschaffen werden. Exemplarisch soll dies für die deutsche Sprache durchgeführt werden. Hierbei wählt die Bibkiothek „java.util.Local“ automatisch die richtige Sprachdatei aus, soweit diese vorhanden ist. Ansonsten wird die Standard-Sprachdatei verwendet. Das Datenformat dieser Dateien ist in Kapitel 6.5 auf Seite 30 beschrieben. Der Zugriff auf die entsprechende Übersetzung erfolgt im Quellcode dann über die entsprechenden Schlüsselwörter. In Listing 5.2 wird exemplarisch anhand eines JLabels die Benutzung der Internationalisierung verdeutlicht. Statt direkt den Text des Labels zu setzen wird die Methode „getTranslation(String key)“ der Hilfsklasse „Internationalization“ genutzt. Diese Me-

thode greift auf die von „java.util.Local“ geladene Sprachdatei zurück und liefert die entsprechende Übersetzung als String zurück.

```
1 jLabel = new JLabel();
2 jLabel.setText(Internationalization.getTranslation("lb_client_port"));
```

Listing 5.2: Beispiel für Internationalisierung

## 5.14 /PF150/ – Kommunikationseinstellungen

Die genutzten Ports für die Kommunikation sollen frei konfigurierbar sein. Dies ist eine Voraussetzung um die Anwendung in verschiedenen Umgebungen nutzen zu können. Durch die flexible Anpassung der Kommunikationseinstellungen kann das Produkt auch in restriktiven Netzwerken genutzt werden.

## 5.15 /PF160/ – Persistenz

Die bei der Benutzung der Software anfallenden Daten und Konfigurationen sollen dauerhaft auf einem Festspeicher gespeichert werden können. Dies wird durch die Speicherung in Dateien für die Konfiguration, die aufgezeichneten Daten und die erstellten Makros auf der lokalen Festplatte des jeweiligen Rechners realisiert.

## 5.16 /PF170/ – Verlaufsansicht

Um die gewünschte Verlaufsansicht, die auch in Kapitel 6.2 auf Seite 25 und Kapitel 7.6 auf Seite 38 näher beschrieben wird, realisieren zu können, sind weitere Funktionen nötig. Diese sollen hier erläutert werden. Zunächst ist es erforderlich, dass überhaupt Nachrichten gespeichert werden können. Dazu wird bei jeder neuen Nachricht ein Nachrichtenobjekt erzeugt und dieses in einer entsprechenden Datenstruktur abgelegt. Die Nachrichtenobjekte selbst müssen zumindest Zugriff auf die wichtigsten

Parameter einer Nachricht, wie etwa Nachrichtentyp, Größe und ähnliches bieten. Weiterhin ist es notwendig für spätere Verarbeitung und die Aktualisierung der Ansicht auf einzelne Nachrichten gezielt zugreifen zu können. Um dieser Forderung gerecht zu werden erhält jede Nachricht eine eindeutige Nummer. Des Weiteren sind Funktionen vorhanden, die es erlauben, Nachrichten einer Seite oder beider Seiten in eine Protokolldatei zu exportieren. Bild 5.6 zeigt das entsprechende Klassendiagramm dieser Funktion.

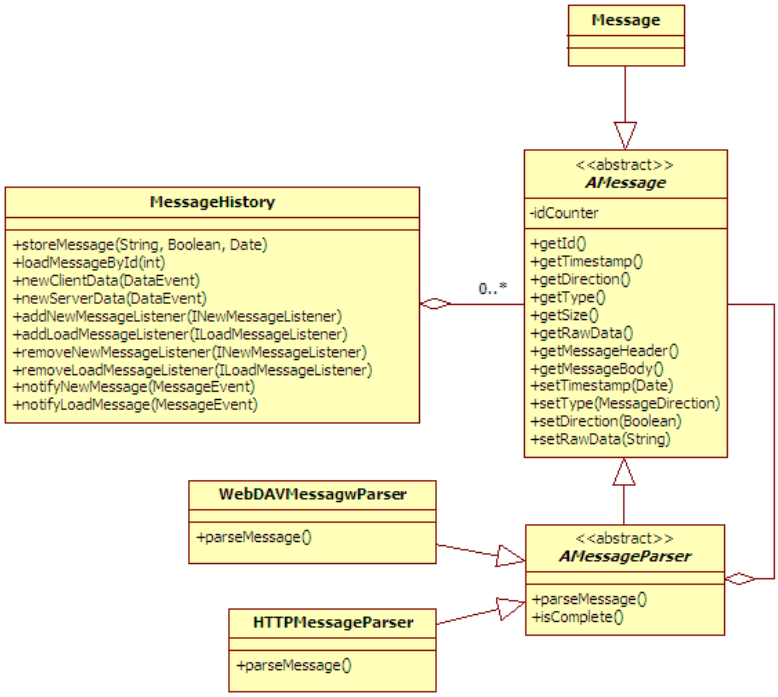


Bild 5.6: Klassendiagramm Verlaufsansicht

# 6 Produktdaten

In den folgenden Abschnitten werden die verschiedenen Daten beschrieben, die bei der Verwendung des Produktes auftreten können.

## 6.1 /PD010/ – Konfigurationsdaten der Software

Die Java-Bibliothek „java.util.Properties“ wird genutzt um die Konfigurationsdaten der Software zu speichern und wieder einzulesen. Die Speicherung der Konfigurationsdaten erfolgt hierbei in einer einfachen Textdatei.

## 6.2 /PD020/ – Protokolldatei des Datenverkehrs

Um den aufgezeichneten Datenverkehr auch mit anderen Applikation auswerten zu können oder später als Sequenz nochmal verwenden zu können, kann man die gespeicherten Nachrichten exportieren. Die Nachrichten werden hierbei in der Reihenfolge ihres Eintreffens und jeweils durch drei Zeilenumbrüche getrennt in einer Datei gespeichert. Der Dateiname der Protokolldatei setzt sich aus dem Datum und der Uhrzeit zusammen. Alternative kann der Nutzer den Namen frei wählen. Listing 6.1 zeigt ein Beispiel einer solchen Datei.

```
1 OPTIONS /slide/files/ HTTP/1.1
2 Host: localhost:9999
3 Connection: Keep-Alive, TE
```

```
4  TE: trailers, deflate, gzip, compress
5  User-Agent: UCI DAV Explorer/0.91 RPT-HTTPClient/0.3-3E
6  Translate: f
7  Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
8
9
10
11 PROPFIND /slide/files/ HTTP/1.1
12 Host: localhost:9999
13 Connection: TE
14 TE: trailers, deflate, gzip, compress
15 User-Agent: UCI DAV Explorer/0.91 RPT-HTTPClient/0.3-3E
16 Depth: 1
17 Translate: f
18 Cookie: JSESSIONID=0CF4C38817DE5B08D685847B2014D4F6
19 Cookie2: $Version="1"
20 Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
21 Content-type: text/xml
22 Content-length: 345
23
24 <?xml version="1.0"?>
25 <A:propfind xmlns:A="DAV:">
26     <A:prop>
27         <A:displayname/>
28         <A:resourcetype/>
29         <A:getcontenttype/>
30         <A:getcontentlength/>
31         <A:getlastmodified/>
32         <A:lockdiscovery/>
33         <A:checked-in/>
34         <A:checked-out/>
35         <A:version-name/>
36     </A:prop>
37 </A:propfind>
```

Listing 6.1: Beispiel Protokolldatei Datenverkehr

## 6.3 /PD030/ – Datenformat für gespeicherte Makros

Die erstellten Makros werden zunächst in ASCII-Dateien abgespeichert. In späteren Versionen der Anwendung kann dies allerdings nicht mehr ausreichend sein. Dann wäre eine Speicherung in einer XML-Struktur zweckmäßig. Um bestimmte Befehle oder Befehlskombinationen mehrmalig zu verwenden kann es dann nötig sein, Platzhalter zu benutzen, die bestimmte Werte in einer Anfrage oder Antwort aktualisieren. Ein Beispiel für solch einen Platzhalter wäre etwa das aktuelle Datum und die aktuelle Uhrzeit. Diese Funktionalität ist jedoch ausdrücklich kein Bestandteil dieser Arbeit. Eine mögliche Struktur einer solchen erweiterten Version ist in Listing 6.2 dargestellt.

```
1  <?xml version="1.0" encoding="windows-1250"?>
2  <makro>
3    <request repeat="2">
4      <![CDATA[
5      OPTIONS /slide/files/ HTTP/1.1
6      Host: localhost:9999
7      ]]>
8    </request>
9    <request>
10     <![CDATA[
11     PROPFIND /slide/files/ HTTP/1.1
12     Host: localhost:9999
13     Content-type: text/xml
14     Content-length: 345
15
16     <?xml version="1.0"?>
17     <A:propfind xmlns:A="DAV:">
18         <A:prop>
19             <A:displayname/>
20             <A:resourcetype/>
21             <A:getcontenttype/>
22             <A:getcontentlength/>
23             <A:getlastmodified/>
24             <A:lockdiscovery/>
25             <A:checked-in/>
```

```
26              <A:checked-out/>
27              <A:version-name/>
28          </A:prop>
29      </A:propfind>
30      ]]>
31    </request>
32  </makro>
```

Listing 6.2: Einfache XML-Version eines Makros

## 6.4 /PD040/ – Datenformat des Datenverkehrs

Das Datenformat des Datenverkehrs zwischen zwei WebDAV-Applikationen ist standardisiert. Die Definitionen der zu berücksichtigenden Standards sollen an dieser Stelle aufgeführt werden:

- RFC 2518 und RFC 4918 (WebDAV)
- RFC 3253 (DeltaV)
- RFC 3648 (Ordered Collections Protocol)
- RFC 3744 (Access Control Protocol)
- DAV Searching and Locating (DASL)

Ausnahmen und Abweichungen von den obigen Standards müssen explizit aufgeführt und dokumentiert werden. Als Beispiel für das Format des Datenverkehrs sollen im folgenden zwei Beispiele dienen. Das erste Beispiel 6.3 stellt eine Client-Anfrage dar. Beispiel 6.4 ist eine mögliche Antwort eines Servers.

```
1 PROPFIND /slide/files/netze_logo.gif HTTP/1.1
2 Host: localhost:9999
3 Connection: TE
4 TE: trailers, deflate, gzip, compress
5 User-Agent: UCI DAV Explorer/0.91 RPT-HTTPClient/0.3-3E
6 Depth: 0
```

```
7  Translate: f
8  Cookie: JSESSIONID=0CF4C38817DE5B08D685847B2014D4F6
9  Cookie2: $Version="1"
10 Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
11 Content-type: text/xml
12 Content-length: 85
13
14 <?xml version="1.0"?>
15 <A:propfind xmlns:A="DAV:">
16     <A:allprop/>
17 </A:propfind>
```

Listing 6.3: WebDAV-Anfrage

```
1  HTTP/1.1 207 Multi-Status
2  Server: Apache-Coyote/1.1
3  Transfer-Encoding: chunked
4  Date: Wed, 09 Apr 2008 07:54:57 GMT
5
6  1ce
7  <?xml version="1.0" encoding="UTF-8"?>
8  <D:multistatus xmlns:D="DAV:">
9      <D:response>
10         <D:href>/slide/files/ntop-overview.pdf</D:href>
11         <D:propstat>
12             <D:status>HTTP/1.1 403 Forbidden</D:status>
13             <D:responsedescription>
14                 <D:error>
15                     <D:supported-report />
16                 </D:error>
17             </D:responsedescription>
18         </D:propstat>
19     </D:response>
20 </D:multistatus>
21
22
23 0
```

Listing 6.4: WebDAV-Antwort

## 6.5 /PD050/ – Ressourcen Internationalisierung

Um die Ressourcen für die gewünschte Internationalisierung der Anwendung zu speichern werden die in Java vorhandenen Möglichkeiten der Klassen „java.util.Local“ und „java.util.ResourceBundle“ genutzt. Die Arbeitsweise der genutzten Funktionen für die Internationalisierung wird in Kapitel 5.13 auf Seite 21 erläutert. Das Datenformat der Resourcendateien für die jeweiligen Sprachen wird an dieser Stelle kurz erläutert. Als Basis dient eine Datei in der Default-Sprache. Im folgenden Beispiel wird diese Datei mit „BeispielBundle.properties“ bezeichnet. Der entsprechende Dateiname für eine Übersetzung in eine andere Sprache besteht dann aus dem Dateinamen der Default-Sprache und der entsprechenden Sprach- und Landesabkürzung. Beipielsweise lautet der Dateiname für eine deutsche Sprachdatei dann „BeispielBundle_de_DE.properties“.

Der Inhalt der Dateien setzt sich aus mehreren Schlüssel-Werte-Paaren zusammen. Hierbei steht der jeweilige Schlüssel links von einem Gleichheitszeichen und der Wert des Schlüssels rechts davon. Die Bezeichnungen der Schlüssel müssen zwingend für jede Sprachdatei indentisch sein, die Werte enthalten die jeweils gewünschte Übersetzung. Die Listings 6.5 und 6.6 zeigen jeweils einen kleinen Ausschnitt aus einer englischen und deutschen Sprachdatei.

```
1 #Actions
2 ac_exit=Exit
3 ac_exit_description=Exit the program
4
5 #AboutDialog
6 dlg_about_title=About DAVInspector
7 dlg_about_close=Close
```

Listing 6.5: Inhalt der Datei BeispielBundle.properties (Englisch)

```
1 #Actions
2 ac_exit=Beenden
3 ac_exit_description=Das Programm schließen
```

```
4
5 #AboutDialog
6 dlg_about_title=Über DAVInspector
7 dlg_about_close=Schließen
```

Listing 6.6: Inhalt der Datei BeispielBundle_de_DE.properties (Deutsch)

# 7 Benutzeroberfläche

Die Benutzeroberfläche stellt die eigentliche Schnittstelle zwischen Produkt und Anwender dar. Sie soll dem Benutzer ermöglichen das Produkt zu konfigurieren, Makros zu erstellen und zu verwenden, den Nachrichtenaustausch zwischen WebDAV-Applikationen zu betrachten und zu beeinflussen, sowie den Verlauf der Kommunikation nachzuvollziehen.

## 7.1 /PG010/ - Anzeige der Daten

Die bei einer Kommunikation anfallenden Daten sollen für Clientseite (Request) und Serverseite (Reply) in optisch voneinander ausreichend getrennten Bereichen angezeigt werden, wie etwa in 7.1 dargestellt. Diese Trennung kann auch durch entsprechende Farbgebung unterstützt werden. In der Basisansicht werden die Daten unverändert angezeigt. Diese Ansicht wird auch als „Rohansicht“ (engl. raw view) bezeichnet. Weitere Sichtweisen, die durch Plugins zu realisieren sind, werden in jeweils eigenen Registerkarten pro Seite angezeigt. Die Anforderung /LF60/ aus dem Lastenheft (Verbergen und Anzeigen von Details) wird hierbei durch die verschiedenen Sichtweisen der Plugins umgesetzt. In der Rohdatenansicht steht zusätzlich ein Kontextmenü zur Verfügung. Dieses ist in Bild 7.2 dargestellt. Es erlaubt das Kopieren, Einfügen und Ausschneiden, sowie den Export von markiertem Text in eine Datei (Siehe auch Kapitel 5.7 Seite 18).

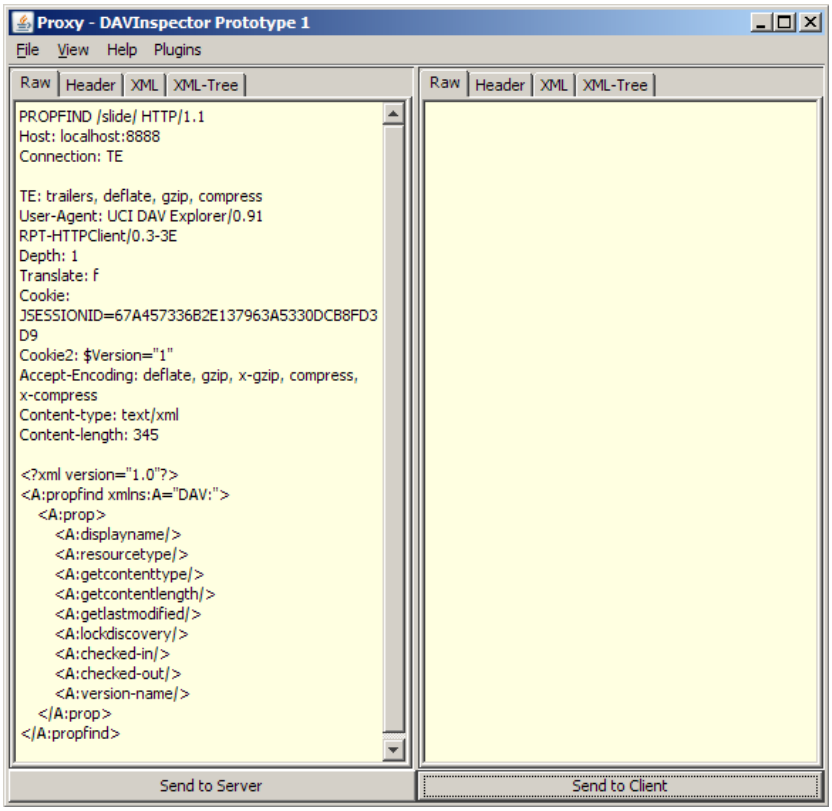


Bild 7.1: Rohdatenansicht DAVInspector

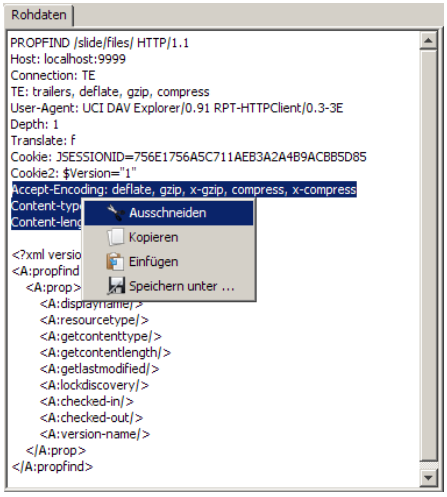


Bild 7.2: Kontextmenü Rohdatenansicht DAVInspector

## 7.2 /PG020/ - XML-Ansicht mit Syntaxhervorhebung

Da bei WebDAV-Protokoll die Daten im Rumpf in einem XML-Format übertragen werden, ist es sehr hilfreich wenn diese Daten farblich und durch Einrückungen aufbereitet werden. Diese Ansicht ist als ein Plugin zu realisieren. Einige WebDAV-Applikationen liefern die Daten zwar strukturiert aus, andere aber nicht. Dieses Plugin stellt somit eine einheitliche Ansicht der XML-Daten zur Verfügung. Zudem findet gleichzeitig eine Syntaxüberprüfung statt. Eine mögliche Realisierung ist in Bild 7.3 dargestellt. Bei der Syntaxüberprüfung auftretende Fehler werden im unteren Teil des Fensters angezeigt.


Bild 7.3: XML-Ansicht mit Syntaxhervorhebung

## 7.3 /PG030/ - XML-Baumansicht

Eine weitere Möglichkeit die XML-Daten des Nachrichtenrumpfes übersichtlich aufzubereiten bietet eine Baumdarstellung. Diese Sicht ist ebenfalls als Plugin zu realisieren. Eine mögliche Umsetzung zeigt Bild 7.4.

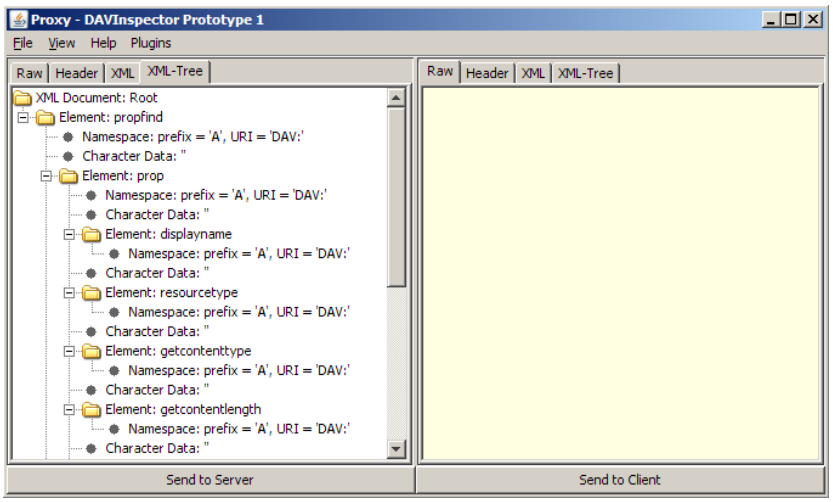


Bild 7.4: XML-Ansicht in einer Baumstruktur

## 7.4 /PG040/ - Ansicht der Kopfdaten

Die Kopfdaten von Anfragen und Antworten sollen in zwei verschiedenen Ansichten dargestellt werden. Zunächst soll eine Rohansicht der Kopfdaten wie in Bild 7.5 verfügbar sein. Hier werden die Kopfdaten unverändert angezeigt. Zusätzlich soll eine strukturierte Ansicht der Kopfdaten wie in Bild 7.6 realisiert werden. Um die Übersichtlichkeit zu verbessern sind die einzelnen Kopfdaten hierbei in geeigneten Kategorien zusammengefasst.

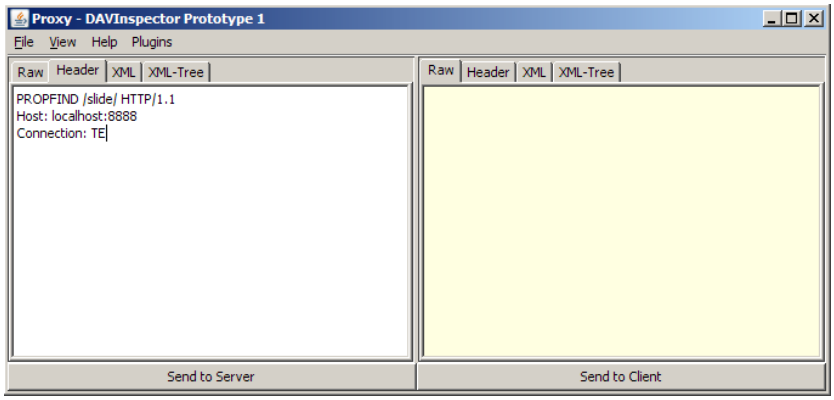


Bild 7.5: Ansicht der Kopfdaten - einfach

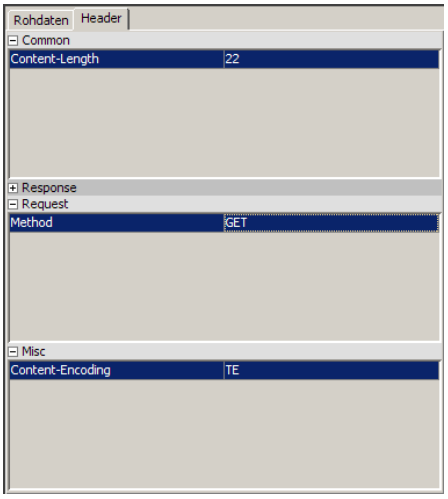


Bild 7.6: Ansicht Kopfdaten - strukturiert

## 7.5 /PG050/ - Plugin mit Editiermodus

Plugins, die die Möglichkeit zur Manipulation der übertragenen Daten bieten, sollen extra gekennzeichnet sein. Dies soll dem Anwender verdeutlichen, an welchen Stellen die Kommunikation beeinflusst werden kann.

## 7.6 /PG060/ - Verlaufsansicht GUI

Um die Übersichtlichkeit bei großen Datenmengen, beziehungsweise komplexeren und umfangreicheren Kommunikationsabläufen, weiter zu verbessern, soll dem Benutzer eine Verlaufsansicht zur Verfügung gestellt werden. In einer Tabelle wird hierbei pro Anfrage und Antwort ein Eintrag erzeugt. Diese Einträge werden mit eins beginnend ab dem Start der Software durchnummeriert und spiegeln die Reihenfolge des Empfangs der Nachrichten wider. Hierbei sollte dem Nutzer bewusst sein, dass eine direkte Zuordnung von einer Antwort auf eine vorausgegangene Anfrage nicht immer richtig ist, vor allem wenn gleichzeitig mehrere Verbindungen bestehen. In Bild 7.7 ist eine beispielhafte Umsetzung der beschriebenen Funktionalität zu sehen. Die Auswahl einer entsprechenden Zeile in der Tabelle soll die Ansichten mit den entsprechenden Daten aktualisieren.

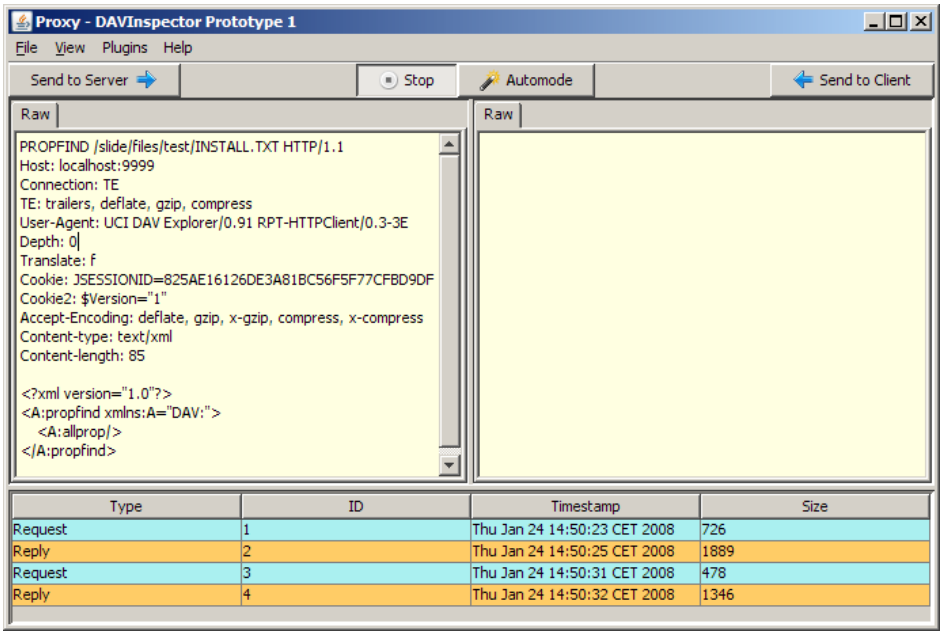


Bild 7.7: Verlaufsansicht

## 7.7 /PG070/ - Konifguration des Produktes

In Bild 7.8 ist der Konfigurationsdialog des DAVInspectors abgebildet. Hier soll die Netzwerkkonfiguration des Produktes einstellbar sein. So können jeweils Adresse und Anschluss für Client- und Serverseite geändert werden. Weitere für das Programm wichtige Einstellungen werden bei Bedarf hier ergänzt.

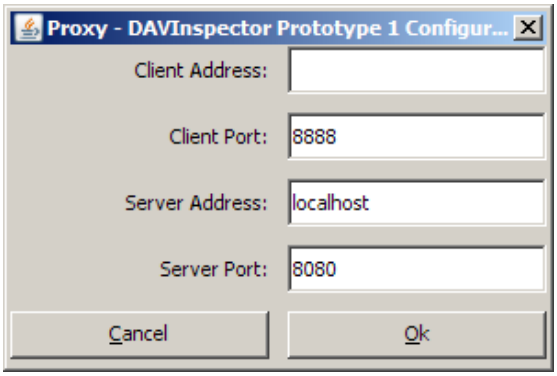


Bild 7.8: Konfigurationsdialog

## 7.8 /PG080/ - Konfiguration der Plugins

Zur komfortablen Verwaltung der aktivierten Plugins auf Client- und Serverseite dient ein Dialog bei dem der Nutzer für Client und Server getrennt die jeweilig zu nutzenden Plugins auswählen kann. Hierbei soll ebenfalls zwischen zwei Typen von Plugins unterschieden werden:

- Auswertende Plugins (View-Plugins): Diese Plugins erhalten eine Kopie des Datenverkehrs und können diesen unabhängig von anderen Plugins auswerten.
- Manipulierende Plugins (Edit-Plugins): Diese Plugins ermöglichen die Veränderung des Datenverkehrs. Hierbei werden die Daten von einem Plugin dieses Typs zum nächsten weitergeleitet. Die Reihenfolge der Plugins ergibt sich durch die Reihenfolge der Registrierung der Plugins und ist nicht durch den Benutzer veränderbar.

In Bild 7.9 ist eine exemplarische Umsetzung des Plugin-Konfigurationsdialoges dargestellt.

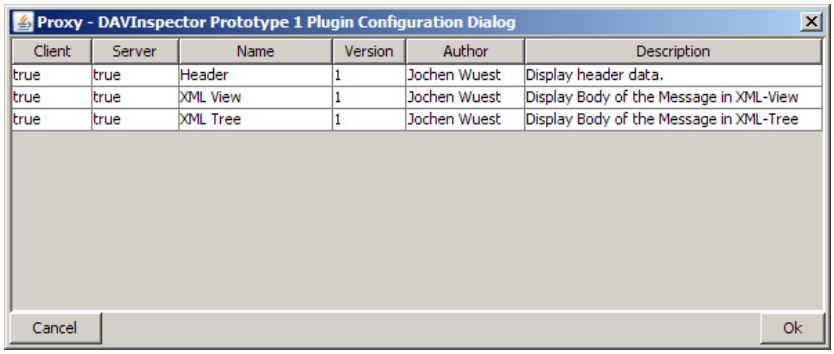


Bild 7.9: Konfigurationsdialog für Plugins

## 7.9 /PG090/ – Exportdialog

Um die Anforderung aus Kapitel 5.1 Seite 11 umzusetzen, wird ein Dialog realisiert, der dem Benutzer erlaubt den Nachrichtentyp und den gewünschten Dateinamen für den Export der Daten auszuwählen. In Bild 7.10 ist eine mögliche Ausführung dieses Dialoges dargestellt.

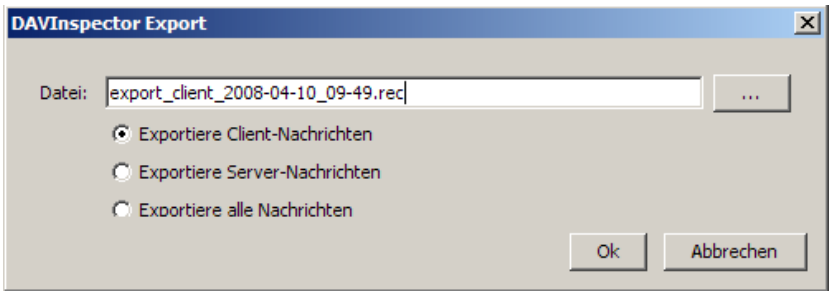


Bild 7.10: Exportdialog für den Datenverkehr

# 8 Nichtfunktionale Anforderungen

## 8.1 /PN010/ – Lauffähigkeit

Das Produkt soll auf allen Rechnern mit mindestens einer Netzwerkschnittstelle lauffähig sein. Das heißt um die Anwendung sinnvoll nutzen zu können ist mindestens eine Netzwerkverbindung erforderlich. Dabei darf es sich jedoch auch um eine Loopback-Schnittstelle handeln. Weiterhin muss eine Java-Laufzeitumgebung vorhanden und nutzbar sein. Für die Anwendung und die bei der Verwendung anfallenden Daten muss genug Speicherplatz auf einem lokalen Datenträger zur Verfügung stehen.

## 8.2 /PN020/ – Plattformunabhängigkeit

Das Produkt soll plattformunabhängig sein, was durch die Wahl der Programmiersprache Java bereits gewährleistet ist. So sind für nahezu alle Plattformen Java-Laufzeitumgebungen vorhanden. Bei der persistenten Speicherung von Daten, wie etwa dem aufgezeichneten Datenverkehr, sollte ebenfalls auf ein plattformunabhängiges Format zurückgegriffen werden. Bei der Auswahl der benötigten Bibliotheken muss auch auf deren Plattformunabhängigkeit geachtet werden. Bei der geplanten Verwendung der Bibliothek Apache log4j [AL4J08] für die Protokollierung ist dies gegeben.

# 9 Qualitätsanforderungen

## 9.1 /PQ010/ – Ergonomie

Da es sich bei der zu entwickelnden Software um ein Werkzeug für Entwickler handelt, sind Ergonomie und intuitive Bedienbarkeit kein Hauptziel der Qualitätsbetrachtung. Trotzdem sollte auf eine übersichtliche und vor allem konsistente Gestaltung der Benutzerschnittstelle geachtet werden.

## 9.2 /PQ020/ – Zuverlässigkeit

Wichtig ist die Zuverlässigkeit der Software, da sie als Werkzeug zum Entwickeln und Testen anderer Software dienen soll. Hierbei ist die Zuverlässigkeit unter dem Aspekt zu betrachten, dass die Software keine Fehler der zu analysierenden Software verdeckt und auch keine Fehler hinzufügt. An dieser Stelle soll auch deutlich darauf hingewiesen werden, dass es sich bei der Software um ein Testwerkzeug handelt, das nicht für den Einsatz in einer Produktivumgebung gedacht ist. Unter einer Produktivumgebung ist in diesem Zusammenhang von der Produktivumgebung der zu testenden Software auszugehen, also der eigentlichen WebDAV-Applikation.

## 9.3 /PQ030/ – Portierbarkeit und Kompatibilität

Ein weiteres wichtiges Ziel ist die Portierbarkeit und Kompatibilität der Software. Dies gilt sowohl für die Systemarchitektur, als auch für die Be-

nutzerschnittstelle. Die Software sollte mittels objektorientierter Programmierung erstellt werden.

## 9.4 /PQ040/ – Aufteilung in Komponenten und Module

Das Produkt soll komponentenbasiert entwickelt werden, um Wartbarkeit und Erweiterbarkeit sicherzustellen.

## 9.5 /PQ050/ – Dokumentation

Weiterhin wird großen Wert auf die Dokumentation der Software und einheitlichen Quellcode gelegt. Um diese Ziele sicherzustellen, wird der Entwickler durch weitere Werkzeuge unterstützt. Das im DLR verwendete Werkzeug ist die Erweiterung Checkstyle für die zu verwendende Entwicklungsumgebung eclipse. Als Grundlage für die Konfiguration von Checkstyle dienen DLR interne Vorgaben. Als weitere Werkzeuge zur Sicherstellung der Softwarequalität werden Unit-Tests und Code-Reviews eingesetzt.

## 9.6 /PQ060/ – Performance

Für hinreichend schnelle Antwortzeiten und Auswertung der Kommunikation wird der Einsatz von aktuellen Rechnersystemen vorausgesetzt. Zunächst handelt es sich bei dem Http-/WebDAV-Protokoll um ein zustandsloses Protokoll. Das bedeutet, nach jeder erfolgreichen Datenübertragung kann die Verbindung geschlossen werden. Sollen erneut Daten übertragen werden, so kann eine neue Verbindung erstellt werden. Aufgrund der unterschiedlichen und nicht vorhersehbaren Ausbreitungspfade im Internet kann es zudem zu Latenzen kommen, die es technischen nicht sinnvoll erscheinen lassen, spezielle Performance-Anforderungen an die Software zu stellen. Es muss lediglich ein unter ergonomischen Gesichtspunkten

„flüssiges Arbeiten“ gewährleistet werden. Die Antwortzeiten der Software sollten dabei im allgemeinen unter einer Sekunde liegen. Die Obergrenze liegt bei drei bis vier Sekunden (siehe auch [NIE04], [ERGO08] und [ISO9241-11]).

# 10 Entwicklungswerkzeuge

Zur Entwicklung stehen mehrere Werkzeuge, die den kompletten Entwicklungszyklus einer Software abdecken, zur Verfügung.

## 10.1 IDE

Zur Entwicklung wird die integrierte Entwicklungsumgebung eclipse eingesetzt. Für die verwendeten Programmiersprachen und Werkzeuge sind die benötigten Erweiterungen zu installieren. Im Einzelnen sind das:

- Subclipse – Erweiterung zum Zugriff auf die Versionsverwaltung
- Checkstyle – Automatische Überprüfung des Quellcodes im Hinblick auf formale Vorgaben (Kommentare, Klammerung, u. s. w.)

## 10.2 Konfigurationsmanagement

Die Sourceforge-Plattform bietet für dieses Projekt alle benötigten Dienste an. Im Einzelnen werden folgende Dienste verwendet:

- Versionsverwaltung mit Subversion
- Bug-Tracker
- Wiki zur Dokumentation
- Mailinglisten

# 11 Ergänzungen

## 11.1 Realisierung

Die Klarheit des Programmcodes bei der Entwicklung hat Vorrang gegenüber Optimierungen, die dem Einsparen von Speicherplatz und/oder Rechenzeit dienen. Die schriftliche Darstellung der Ergebnisse und ein hausinterner Vortrag bilden den Abschluss der Diplomarbeit. Der Vortrag wird zusätzlich an der Universität Siegen abgehalten.

## 11.2 Open Source

Das Produkt setzt ausschließlich auf Open Source Software auf. Die jeweiligen Produkte stehen unter Copyright ihrer jeweiligen Ersteller. Auch dieses Produkt wird ein Open Source Produkt, das unter der Apache 2.0 Lizenz [APA07] eingesetzt und erweitert werden kann.

# 12 Gliederung der Realisierung

Die Realisierung des DAVInspector Prototyps gliedert sich in drei Ausbaustufen:

1. Basissystem (Das System muss in diesem Umfang realisiert werden)
2. Erweitertes System (Die zusätzlichen Funktionen diese Systems sollen realisiert werden.)
3. Optionales System (Steht noch Zeit zur Verfügung können diese Funktionen realisiert werden.)

## 12.1 Basissystem

Das Basissystem soll das minimale Ergebnis dieser Arbeit sein und besteht aus folgenden zu realisierenden Funktionen:

- /PF010/ – Abspeichern von Datenverkehr
- /PF020/ – Weiterleitung von Datenverkehr
- /PF030/ – Eingriff in den Datenverkehr
- /PF040/ – Unterscheidung von Anfrage und Antwort
- /PF050/ – Unterscheidung von Kopf- und Rumpfdaten
- /PF060/ – Verbergen und Anzeigen von Details
- /PF070/ – Makro manuell Erstellen
- /PF110/ – Erweiterbarkeit durch Filter/Plugins
- /PF150/ – Kommunikationseinstellungen konfigurierbar
- /PF160/ – Persistenz

- /PD010/ – Konfigurationsdaten der Software
- /PD020/ – Protokolldatei des Datenverkehrs
- /PD030/ – Datenformat für gespeicherte Makros
- /PD040/ – Datenformat des Datenverkehrs
- /PG010/ – Anzeige der Daten
- /PG020/ – XML-Ansicht mit Syntaxhervorhebung
- /PG030/ – XML-Baumansicht
- /PG040/ – Ansicht der Kopfdaten
- /PG050/ – Plugin mit Editiermodus
- /PG070/ – Konfiguration des Produktes
- /PG090/ – Exportdialog
- /PN010/ – Lauffähigkeit
- /PN020/ – Plattformunabhängigkeit
- /PQ010/ – Ergonomie
- /PQ020/ – Zuverlässigkeit
- /PQ040/ – Aufteilung in Komponenten und Module
- /PQ050/ – Dokumentation
- /PQ060/ – Performance

## 12.2 Erweitertes System

Das erweiterte System besteht aus dem Basissystem und zusätzlich werden, sofern der Zeitrahmen dies zulässt, folgende Funktionen umgesetzt:

- /PF080/ – Makro aus Datenverkehr erstellen
- /PF120/ – Automodus

- /PF170/ – Verlaufsansicht
- /PG060/ – Verlaufsansicht GUI
- /PG080/ – Konfiguration der Plugins
- /PQ030/ – Portierbarkeit und Kompatibilität

## 12.3 Optionales System

Die Funktionen dieses Systems können realisiert werden, wenn das erweiterte System umgesetzt wurde und noch Zeit zur Verfügung steht.

- /PF090/ – Simulation von Kommunikation
- /PF100/ – Makro mit Assistenten erstellen
- /PF130/ – Haltepunkt
- /PF140/ – Internationalisierung
- /PD050/ – Ressourcen Internationalisierung

## 12.4 Zeitplanung

Das Projekt wird voraussichtlich im Oktober 2007 starten und im April 2008 beendet sein. Ein erster Zeitplan sieht wie folgt aus:

| Aufgabe | Dauer |
|---|---|
| Einarbeitung | 2 Wochen |
| Marktstudie (Sprache,Werkzeuge) | 4 Wochen |
| Lastenheft, Pflichtenheft | 8 Wochen |
| Prototyp | 2 Wochen |
| Implementierung | 8 Wochen |
| Test | 2 Wochen |
| Ausarbeitung / Abstract | 8 Wochen |
| Präsentation | 2 Wochen |
| Summe | 36 Wochen |

Tabelle 12.1: Zeitplanung

Insgesamt stehen acht Wochen für die Implementierung und zwei Wochen für Tests zur Verfügung. Diese Zeit verteilt sich folgendermaßen auf das Basissystem und das erweiterte System:

| System | Dauer |
|---|---|
| Basissystem | 8 Wochen |
| Erweitertes System | 2 Wochen |
| Summe | 10 Wochen |

Tabelle 12.2: Zeitplanung Systeme

Das optionale System wurde in dieser Zeitplanung nicht berücksichtigt, da diese Funktionen nur realisiert werden können, wenn sowohl Basissystem als auch erweitertes System bereits fertig gestellt sind und noch Zeit zur Verfügung steht.

# Literaturverzeichnis


[AL4J08] Apache Logging Services – Apache log4j 1.2
http://logging.apache.org/log4j/1.2/index.html
*7. April 2008*

[APA07] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html
*15. Oktober 2007*

[ERGO08] Gesellschaft Arbeit und Ergonomie – online e.V.
http://www.ergo-online.de/site.aspx?url=html/software/ergonomische_gestaltung_von_w/grafiken_multimedia.htm
*7. April 2008*

[ISO9241-11] EN ISO 9241-11:
Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten
Teil 11: Anforderungen an die Gebrauchstauglichkeit – Leitsätze, 1998

[NIE04] Nielsen, Jakob:
Designing Web Usability
Pearson Education Deutschland, 2004

[RFC2616] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt
*15. Oktober 2007*

[RFC4918] RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)
http://www.ietf.org/rfc/rfc4918.txt
*15. Oktober 2007*

# Glossar

**ACL** Engl. Abkürzung für Access Control List. ACLs sind Listen, die Rechte von Benutzern auf Objekte speichern.

**Bug-Tracker** Software zur Verfolgung und Dokumentation von Fehlern in Software.

**Catacomb** Catacomb ist eine Erweiterung für das WebDAV-Modul des Apache Webservers. Die Daten werden hierbei statt im Dateisystem in einer relationalen Datenbank abgelegt.

Siehe auch: http://catacomb.tigris.org/

**Checkstyle** Checkstyle ist ein Werkzeug für Entwickler mit dem man Quelltext nach formalen Kriterien überprüfen kann.

Siehe auch: http://eclipse-cs.sourceforge.net/

**DASL** Engl. Abkürzung für DAV Searching and Locating. DASL befasst sich mit der serverseitigen Suche in WebDAV-Repositories. Die Arbeitsgruppe zu diesem Standard wurde allerdings mittlerweile aufgelöst.

**DataFinder** DataFinder ist eine Applikation auf Clientseite zur Verwaltung technisch-wissenschaftlicher Daten. Hierbei können die Daten sowie die Metadaten über verschiedenartige Speicherschnittstellen auf einem oder mehreren Servern abgelegt werden.

Siehe auch:
http://www.dlr.de/sc/desktopdefault.aspx/tabid-1189/1650_read-3140/

**DeltaV** Engl. Web Versioning and Configuration Management. Dieses Protokoll ist eine Erweiterung des WebDAV-Protokolls um Versions- und Konfigurationsmanagement.

**DLR** Deutsches Zentrum für Luft- und Raumfahrt.

Siehe auch: http://www.dlr.de

**Eclipse** Eclipse ist ein Framework zur Entwicklung von Anwendungen und ist vergleichbar mit einer IDE, kann aber auch noch wesentlich mehr bieten, da es durch eine entsprechende Architektur einfach erweiterbar ist.

**IDE** Engl. Abkürzung für Intergrated Developement Environment. Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software, das mehrere Komponenten in einer Oberfläche zusammenfasst.

**Loopback** Eine Loopback Netzwerkschnittstelle ist ein lokaler Informationskanal mit nur einem Endpunkt. Sender und Empfänger sind hierbei identisch.

**Mailingliste** Eine Mailingliste ist ein E-Mail-Verteiler für eine Gruppe von Menschen. Einzelne Mailinglisten haben häufig bestimmte Themen und dienen zur Kommunikation und Information.

**Makro** Ein Makro ist in der Programmierung eine vorgegebene Sequenz von Befehlen oder Aktionen. In diesem Zusammenhang bezeichnet ein Makro eine aufgezeichnete Kommunikationssequenz zwischen zwei WebDAV-Applikationen.

**MVC** Engl. Model-View-Controller. MVC ist ein zusammengesetztes Architekturmuster zur Gestaltung von Anwendungen. Um eine bessere Wiederverwendbarkeit und einfachere Erweiterbarkeit zu erhalten erfolgt eine Aufteilung in die Elemente Datenmodell (Model), Präsentation (View) und Steuerung (Controller).

**Open Source** Open Source Software ist quelloffene Software. Das heißt, der Quellcode dieser Software ist frei zugänglich und deren Bearbeitung und Verbreitung erlaubt und erwünscht. Es gibt verschiedene Lizenzmodelle, die eine genaue Definition der Verwendung und Verbreitung der jeweiligen Software regeln.

**Plugin** Ein Plugin ist ein Software- oder Hardwaremodul, welches in eine bestehende Software oder Hardware eingebunden wird und die Funktionalität der Software oder Hardware erweitert.

**Proxy** Ein Proxy ist im Zusammenhang mit Netzwerken ein Dienstprogramm zur Steuerung von Netzwerkverkehr. Ein Proxy besitzt sowohl eine ServerSchnittstelle als

| | |
|---|---|
| | auch eine Client-Schnittstelle. Im einfachsten Falle leitet der Proxy den Netzwerkverkehr unbeeinflusst weiter. In diesem Fall spricht man auch von einem „Relay“. |
| **RFC** | Engl. Requests for Comments. Die RFCs sind eine Sammlung von technischen und organisatorischen Dokumenten zu den im Internet verwendeten Protokollen und Verfahren. |
| **SC** | Abkürzung für die Abteilung Simulations- und Softwaretechnik innerhalb des DLR.<br>Siehe auch: `http://www.dlr.de/sc/` |
| **sourceforge.net** | sourceforge.net ist eine Online-Plattform. Hier werden einem oder einer Gruppe von Entwicklern Werkzeuge zur Unterstützung bei der Anwendungsentwicklung und zur Projektverwaltung geboten.<br>Siehe auch: `http://www.sourceforge.net/` |
| **Subclipse** | Eclipse-Plugin für die Versionsverwaltung Subversion. |
| **Subversion** | Open Source Software für die Versionsverwaltung von Dateien und Ordnern.<br>Siehe auch: `http://subversion.tigris.org/` |
| **TCP** | Engl. Abkürzung für Transmission Control Protocol. TCP ist ein verbindungsorientiertes Transportprotokoll der OSI-Schicht 4. |
| **UML** | Engl. Abkürzung für Unified Modeling Language. Eine von der Object Management Group entwickelte Sprache zur Modellierung von Software. |

**Unit-Test** Der Unit-Test wird auch als Modultest bezeichnet und dient zur Überprüfung der Korrektheit von Programmteilen.

**WebDAV** Engl. Abkürzung für Web-based Distributed Authoring and Versioning. WebDAV ist ein offener Standard zur Bereitstellung von Dateien im Internet und ist eine Weiterentwicklung des HTTP-Protokolls.

Siehe auch: http://www.webdav.org/

**Wiki** Ein Wiki ist eine Sammlung von Hypertextseiten, die von Benutzern gelesen, editiert und verknüpft werden können.