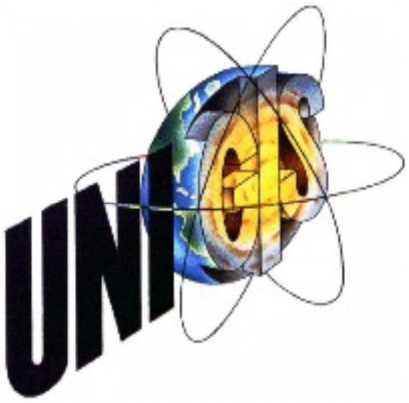


Master Thesis

im Rahmen des
Universitätslehrganges «Geographical Information Science & Systems»
(UNIGIS MSc) am Zentrum für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

«Caching-Techniken für Web Processing Services»

vorgelegt von

Bernd Hellmuth Härtwig
U1419, UNIGIS MSc Jahrgang 2009

Zur Erlangung des Grades
«Master of Science (Geographical Information Science & Systems) – MSc (GIS)»

Gutachter:
Ao. Univ. Prof. Dr. Josef Strobl

Düsseldorf, 30.06.2013

Erklärung

Ich versichere, diese Master Thesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet.

Düsseldorf, den 30. Juni 2013 ..……...

Zusammenfassung

Bis zum heutigen Tage werden Geodaten mehrheitlich mit Desktop-GIS verarbeitet. Mittlerweile sind die Leistungsmerkmale der Netzwerkbandbreiten und Prozessoren geeignet, um auch große Datenmengen performant über das Internet zu übertragen. Daher rücken Web Processing Services immer mehr in den Fokus. Ein WPS stellt über standardisierte Schnittstellen verschiedene Analysefunktionen (z.B. Puffer- oder Verschneidungsfunktionalitäten) zur Verfügung, die bisher ein Hauptmerkmal von Desktop-GIS sind. Mehrere WPS-Implementierungen sind verfügbar. Bei der (zukünftigen) Analyse von Geodaten mit WPS kann man davon ausgehen, dass die gleichen Ausgangsdaten wiederholt von den Ursprungsservern übertragen werden müssen. Eine Reduzierung der Netzwerklast und eine Verbesserung der Performance könnte mit Web-Caching-Techniken erreicht werden. Web-Caching ist eine Technik mit der Daten auf Proxy-Servern zwischengespeichert werden.

In dieser Arbeit werden verschiedene Konzepte für die Nutzung von Web-Caching-Techniken in Verbindung mit WPS erstellt und diskutiert. Mit den Komponenten Zoo-WPS und dem Proxy-Server Squid wird gezeigt, dass die Anbindung eines Web-Caches zum Zwischenspeichern von WFS-Daten möglich ist. Darüber hinaus untersucht die Arbeit die Auswirkungen dieser Implementierung auf die Performance. Es wird gezeigt, dass der Einsatz von Web-Caching in der Mehrzahl der untersuchten Fälle zu einer besseren Performance führt als bei einer Vergleichskonfiguration ohne Cache. In wenigen Konstellationen führt das Web-Caching zu einer geringfügig schlechteren Performance als bei der Vergleichskonfiguration.

Abstract

Currently spatial data are mostly processed with desktop GIS. Meanwhile, the CPU performance and the network bandwidth are adapted to transmit large quantities of data via the Internet with good performance. This is the reason Web Processing Services are increasingly becoming the focus. A WPS provides various analysis functions via standardised interfaces (e.g. buffer or intersection functionalities), that are a major feature of desktop GIS. Several WPS implementations are available. At the (future) analysis of spatial data with WPS can be assumed as from the fact that the same output data must be transferred repeatedly from the origin servers. A reduction of the network load and an improvement of performance could be achieved with web caching techniques. Web caching is a technology to temporarily store data on proxy servers.

In this work, various concepts are created and discussed for the use of web caching techniques in conjunction with WPS. It is demonstrated with the components Zoo-WPS and the proxy server Squid that the connection of a web cache enables to cache WFS data. In addition, this work examines the effects of this implementation on the performance. It is demonstrated, that the use of web caching in the majority of investigated cases leads to better performance than a reference configuration without cache. In a few situations, the web caching leads to a slightly worse performance than a benchmark configuration.

Inhaltsverzeichnis

1 Einleitung ... 1
1.1 Problemstellung ... 1
1.2 Ziele und Hypothesen ... 2
1.3 Struktur der Arbeit ... 4

2 Theoretische Grundlagen ... 6
2.1 Web Processing Service (WPS) ... 6
2.2 Hypertext Transfer Protocol ... 11
2.3 Caching ... 13
2.4 Web-Caching ... 14
2.4.1 Ziele von Web-Caching ... 14
2.4.2 Caching-Strategien ... 15
2.4.3 Arten von Web-Caches ... 16
2.4.4 Funktionsweise von Web-Caching ... 18

3 Konzeption ... 20
3.1 Benachbarte Arbeite ... 20
3.2 WPS-Caching-Varianten ... 21
3.2.1 Grundlegende Überlegungen zum WPS-Caching ... 21
3.2.2 Caching von WPS-Eingangsdaten (Input-Cache) ... 24
3.2.3 Caching von WPS-Ergebnisdaten (Output-Cache) ... 31
3.2.4 Spezifikation ... 36
3.3 Test der Software ... 41
3.4 Performance- und Lasttests ... 42
3.5 Anforderungen an Messwerkzeuge zur Durchführung von Performance- und Lasttests ... 43
3.6 Messwerkzeuge ... 45

4 Implementierung ... 47
4.1 Verwendete Software-Komponenten ... 47
4.1.1 Zoo-WPS ... 48
4.1.2 HTTP-Proxy Squid ... 48
4.2 Anpassung des Zoo-Kernels ... 49

4.3 Konfiguration von Squid 51
4.4 Werkzeug zum Messen von Antwortzeiten 51
4.5 Infrastruktur 56

5 Ergebnisse 59
5.1 Zoo-Kernel 59
5.2 Konzeption der Performance- und Lasttests 61
5.3 Ergebnisse der Performance- und Lasttests 63

6 Diskussion und Schlussfolgerung 66
6.1 Diskussion 66
6.2 Schlussfolgerung 70

7 Referenzen 72

Anhang 76

Abbildungsverzeichnis

Abbildung 1.1: Vorgehensweise 5
Abbildung 2.1: Proxy-Cache 17
Abbildung 2.3: Reverse-Proxy-Cache 18
Abbildung 3.1: Ablauf einer einfachen WPS-Anfrage 22
Abbildung 3.2: Input- und Output-Cache 23
Abbildung 3.3: Ablauf einer WPS-Execute-Operation 28
Abbildung 3.4: Filtern des HTTP-Verkehrs mit einem WCCP-fähigen Router 29
Abbildung 3.5: Ablauf des WPS-File-Caching 30
Abbildung 3.6: Ablauf einer WPS-Anfrage mit Output-Caching 35
Abbildung 3.7: Systemanwendungsfälle für das WPS-Caching 37
Abbildung 3.8: Vorhandensein und Gültigkeit prüfen 39
Abbildung 3.9: Durchführen der WFS-Anfragen 40
Abbildung 3.10: Ablauf der Performance- und Lasttests 44
Abbildung 4.1: Klassenmodell WebServiceBenchmark 53
Abbildung 4.2: Referenzarchitektur 57
Abbildung 5.1: Messergebnis für 20 Puffer-Berechnungen 64
Abbildung 5.2: Messergebnis für 200 Puffer-Berechnungen 64

Tabellenverzeichnis

Tabelle 2.1: HTTP-Header-Anfragen 12

Tabelle 3.1: Anwendungsfall „WFS-Anfrage mit Proxy-Caching“ 26

Tabelle 3.2: Anwendungsfall „WFS-Anfrage mit Input-Caching“..... 32

Tabelle 4.1: Übersicht der am Zoo-Kernel durchgeführten Erweiterungen 50

Tabelle 3.1: Testparameter..... 62

Tabelle 3.2: Testserien, Testläufe und Einzeltests..... 62

Abkürzungsverzeichnis

C	Imperative Programmiersprache
C++	ISO genormte Programmiersprache; Erweiterung von C
C#	C-Sharp, objektorientierte Programmiersprache von Microsoft
GDAL	Geospatial Data Abstraction Library
GDI	Geodaten-Infrastruktur
GEOS	Geometry Engine Open Source
GIS	Geographisches Informationssystem
GML	Geography Markup Language
GNU	General Public License
GPL	GNU General Public License
Hrsg	Herausgeber
HTTP	Hypertext Transfer Protocol
ISO	Internationale Organisation für Normung
JSON	JavaScript Object Notation
KVP	Key Value Pair
OGC	Open Geospatial Consortium
OGR	OGR Simple Feature Library, Teil von GDAL
OWS	OpenGIS Web Services
SOA	Service Oriented Architecture

SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WCCP	Web Cache Communication Protocol
WFS	Web Feature Service
WFS-T	Transactional Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WOA	Web Oriented Architecture
WSRF	Web Service Resource Framework
XML	Extensible Markup Language
ZOO	WPS (Web Processing Service) Open Source Project released under a MIT/X-11 style license

1 Einleitung

1.1 Problemstellung

Web Processing Services (WPS) sind ein Baustein auf dem Weg zu verteilten Geoinformations-Systemen (GIS) in Geodateninfrastrukturen (Brauner, 2008). Hierbei werden Funktionalitäten zum Verarbeiten von Geodaten über standardisierte Schnittstellen zur Verfügung gestellt. Räumliche Analysen, Generalisierungen oder komplexe Bildbearbeitungen werden heute mit Funktionen durchgeführt, die in klassischen GIS oder Datenbanken integriert sind (Brauner, 2008). Die aktuellen Prozessoren und Netzwerkbandbreiten erlauben die Verarbeitung dieser Prozesse mit Web Processing Services, die Teil einer GDI sind (Foerster und Schäffer, 2007). Dabei übermittelt der Client die zu verarbeitenden Daten oder deren Referenzen sowie die Bezeichnung einer vordefinierten Funktionalität mit den entsprechenden Parametern an einen WPS-Server. Der Server führt die Verarbeitung aus und übermittelt das Ergebnis direkt als Datei oder stellt eine Webadresse zum Abrufen zur Verfügung.

Es ist zu vermuten, dass mit der produktiven Einführung und Verbreitung von WPS in immer kürzeren Zeitabständen die gleichen Datensätze verarbeitet werden sollen oder sogar identische Anfragen an den WPS-Server gerichtet werden. Die zu verarbeitenden Daten müssen wiederholt von den Ursprungsservern angefordert werden. Dies könnte zu starken Netzbelastungen und zu längeren Verarbeitungszeiten führen. Werden identische Anfragen an den WPS gerichtet, so müssten dieselben Verarbeitungsschritte wiederholt ausgeführt werden. Ressourcen würden unnötig beansprucht. Längere Verarbeitungszeiten könnten die Folge sein.

Um das mehrfache Übertragen von Daten zu vermeiden könnte man Web-Caching-Techniken einsetzen. Caches dienen zum Zwischenspeichern von Daten und werden in fast allen Bereichen von Computersystemen eingesetzt, sowohl in der Software als auch in der Hardware. In Datenbanken, CPUs, Festplatten und Anwendungen werden Caches mit dem Ziel betrieben, häufig und oft angefragte Ressourcen so zu speichern, dass sie bei Bedarf schneller (als vom Ursprungsserver) zur Verfügung gestellt werden können (Kuhn und Raith, 2013). Unter Web-Caching versteht man eine Technik mit der Daten auf Proxy-Servern zwischengespeichert werden (Wessels, 2001). Die Proxy-Server können im lokalen Netzwerk oder im Internet betrieben werden.

1.2 Ziele und Hypothesen

Grundgedanke dieser Arbeit ist es, die beiden Techniken WPS und Web-Caching zu verbinden. Das übergeordnete Ziel ist, praktische Erkenntnisse über Caching-Techniken für WPS zu gewinnen. Im Einzelnen werden die folgenden Ziele verfolgt:

- Konzeption und Gegenüberstellung verschiedener Caching-Techniken für Web Processing Services,
- Realisierung von Teilaspekten.

Der Einsatz von Web-Caching-Techniken für WPS soll dazu führen, dass häufig benötigte Daten nicht mehrfach von den Ursprungsservern zu übertragen sind. Dies sollte zu einer Reduzierung des Netzverkehrs und zu einer Verbesserung der Performance führen. Die Arbeit untersucht die folgende Hypothese:

- Die Antwortzeiten von WPS-Anfragen mit referenzierten WFS-Daten können durch den Einsatz von Caching-Techniken reduziert werden.

Im Einzelnen werden die folgenden Forschungsfragen untersucht.

- Wie können Web-Caching-Techniken zum Zwischenspeichern eingesetzt werden?
  In der Arbeit wird beschrieben wie und mit welchem Aufwand eine WPS-Installation um Web-Caching-Funktionalitäten erweitert werden kann. Es

werden Unterschiede zwischen einem Input-Cache zum Zwischenspeichern von WFS-Daten und eines Output-Caches zum Puffern von Analysedaten aufgezeigt. Für beide Caches werden Konzepte entwickelt und diskutiert. Die konzipierten Lösungen für das Input-Caching werden mit Zoo-WPS und dem Proxy-Cache Squid technisch realisiert. Dabei wird der HTTP-Verkehr zwischen Zoo-WPS und den WFS-Servern so umgeleitet, dass die Daten in Squid zwischengespeichert werden können. Die Darstellung und Diskussion von Problemen und Unterschieden sind wesentliche Punkte der Untersuchung.

- Welche Auswirkungen hat der Einsatz von Web-Caching-Techniken auf die Performance?
  Mit der Frage wird der Einfluss der softwaretechnischen Anpassungen auf die Performance untersucht. Das Antwortzeitverhalten von drei WPS-Konfigurationen (Web-Caching, integrierter WPS-Cache, ohne Cache) wird systematisch untersucht und diskutiert. Es wird zunächst untersucht, wie die Auswirkungen auf die Performance gemessen werden können. Danach wird ein entsprechendes Konzept und die Durchführung an Hand eines Anwendungsfalls vorgestellt. Die erzielten Ergebnisse werden anschließend diskutiert.

Für die Arbeit werden folgende Einschränkungen getroffen:

- Es werden ausschließlich WPS-Operationen vom Typ *Execute* via HTTP-Post mit referenzierten WFS-Daten untersucht.
- Die für die Studie herangezogenen Anwendungsfälle beschränken sich auf die fundamentalen GIS-Funktionalitäten *Puffer (Buffer)* und *Verschneidung (Intersection)*.
- Die Arbeit soll keine umfassende und abschließende Analyse geben; die gewonnenen Ergebnisse und Erkenntnisse müssen sicherlich vertieft werden.
- Es wird keine Nutzererhebung durchgeführt.

Die Arbeit setzt grundsätzliche Kenntnisse über den Web Processing Service sowie das Netzwerkprotokoll HTTP voraus. Sie richtet sich an GIS-Koordinatoren oder -Fachleute, die von den Forschungsergebnissen profitieren oder einen Web-Cache für WPS einrichten möchten.

1.3 Struktur der Arbeit

Die Arbeit besteht im Wesentlichen aus sechs Kapiteln.

Nach dem einleitenden ersten Kapitel werden im zweiten Kapitel theoretische Grundlagen und Fachbegriffe erläutert. Im ersten Teil des Kapitels wird der Web Processing Service vorgestellt. Der zweite Teil stellt das Hypertext Transfer Protokoll vor und gibt einen Überblick über Caching-Techniken. Danach werden Web-Caching-Techniken vorgestellt.

Aufbauend auf den theoretischen Grundlagen werden im dritten Kapitel verschiedene Caching-Szenarien für WPS entwickelt und spezifiziert. Darüber hinaus werden Anforderungen an Werkzeuge zur Durchführung von Performance- und Lasttests definiert und spezifiziert. Zur Dokumentation werden verschiedene UML-Diagrammarten verwendet.

Die Umsetzung der Spezifikation und eine Referenzarchitektur werden im vierten Kapitel erläutert. Es werden Auswahl und Funktionsweisen der verwendeten Komponenten sowie die softwaretechnischen Anpassungen vorgestellt. Des Weiteren wird die Entwicklung eines objektorientierten Werkzeuges zur Durchführung von Performance- und Lasttests vorgestellt.

Im ersten Teil des fünften Kapitels wird das Ergebnis der zuvor durchgeführten softwaretechnischen Anpassungen vorgestellt. Danach wird das Antwortzeitverhalten für drei unterschiedliche Testszenarien, mit Caching-Proxy, mit integriertem File-Caching und ohne Caching systematisch an Hand eines Anwendungsfalls untersucht und dokumentiert. Es werden Konzeption und Durchführung der Performance-Messungen vorgestellt.

Im sechsten Kapitel werden die im vorangegangenen Kapitel festgestellten Ergebnisse diskutiert. Es wird untersucht, ob die Hypothese durch die erzielten Ergebnisse bestätigt oder falsifiziert wird. Mit den gewonnenen Erkenntnissen und Ergebnissen werden die Forschungsfragen beantwortet. Darüber hinaus werden Hinweise zu weiteren Forschungsbedarfen gegeben. Hierbei werden auch benachbarte Arbeiten einbezogen. Das Kapitel endet mit einem Ausblick.

Die Vorgehensweise kann Abbildung 1.1 entnommen werden.

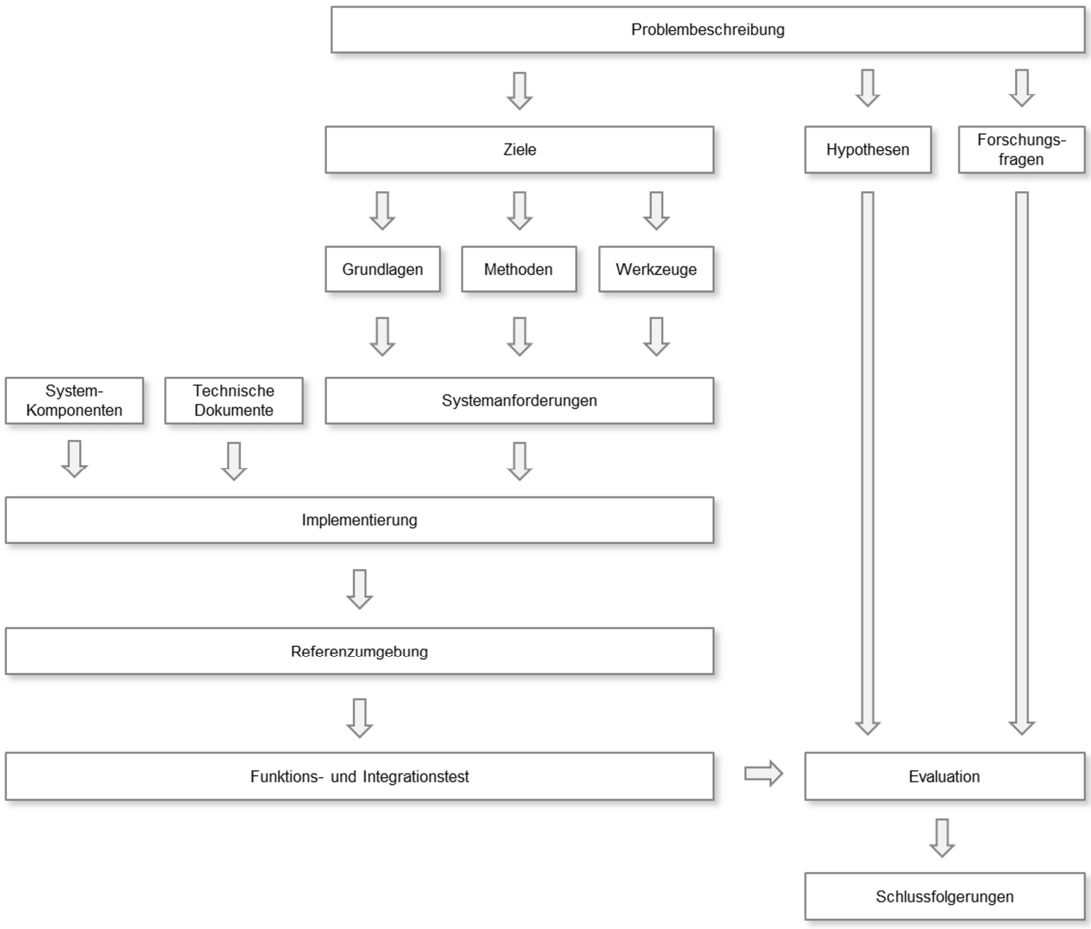


Abbildung 1.1: Vorgehensweise

2 Theoretische Grundlagen

In diesem Kapitel werden Fachbegriffe und theoretische Grundlagen erläutert. Im ersten Teil des Kapitels wird der Web Processing Service vorgestellt. Die theoretischen Ausführungen werden mit Beispielen ergänzt. Der zweite Teil des Kapitels stellt das Hypertext Transfer Protokoll vor. Der dritte Teil gibt zunächst einen allgemeinen Überblick über Caching-Techniken und stellt danach Web-Caching-Techniken vor.

2.1 Web Processing Service (WPS)

Die Web Processing Service-Spezifikation ist einer der neuesten internationalen Standards des Open Geospatial Consortium. Die erste Veröffentlichung erfolgte im Jahr 2005. Die aktuelle Spezifikation wurde im Jahre 2007 als Version 1.0.0 veröffentlicht (OGC 2007). Die generisch ausgelegte Spezifikation definiert eine allgemeine Schnittstelle für Zugriffe auf Funktionalitäten zum Prozessieren von Geodaten. Die Prozessarten sind nicht vorgegeben, diese werden von jeder WPS-Implementierung selbst festgelegt (OGC 2007). In der relativ neuen WPS-Spezifikation wurden auch die im IT-Mainstream bereits etablierten Standards SOAP für die Kommunikation und WSDL (Web Service Description Language) als Skriptsprache berücksichtigt.

Ein Web Processing Service (WPS) stellt Funktionen für räumliche Analysen von Geodaten über das Internet zur Verfügung. Es können sowohl Rasterdaten als auch Vektordaten bearbeitet werden. Als Kommunikationsprotokoll wird HTTP benutzt. Die Umsetzung von Anfragen (Requests) via HTTP-Get ist für alle WPS verpflichtend (OGC 2007).

Operationen

Ein OGC-konformer WPS muss die drei verpflichtenden Operationen GetCapabilities, DescribeProcess und Execute implementieren.

GetCapabilities

Die Operation *GetCapabilities*, die von jedem OGC-konformen Dienst zur Verfügung gestellt werden muss, stellt ein *Capabilities Document* mit wichtigen Metadaten der Server-Instanz sowie den vom Server bereitgestellten Funktionen zur Verfügung. Zu jeder Funktion werden eine eindeutige Bezeichnung, ein Titel und eine kurze Beschreibung ausgegeben. Der Anfrage sind die drei verpflichtenden Parameter

```
Service=WPS
Version=1.0.0
Request=GetCapapilities
```

zu übergeben (OGC 2007). Die Antwort erfolgt im xml-Format. Die Umsetzung von Anfragen via HTTP-Get ist für jeden WPS verpflichtend. Anfragen via HTTP-Post zur Unterstützung des Netzwerkprotokolls SOAP können optional implementiert werden (OGC 2007). Eine beispielhafte Anfrage via HTTP-Get, bei der die Parameter als Schlüssel-Wert-Paar (Key Value Pair, KVP) übergeben werden, sieht wie folgt aus:

```
http://www.beispiel.com/geoserver/wps?
  Service=WPS&
  Version=1.1.0&
  Request=GetCapabilities
```

Der WPS muss die folgenden verpflichtenden Parameter in einer GetCapabilities-Antwort zur Verfügung stellen:

Bezeichnung	***Definition***
Service:	eindeutige Bezeichnung des Dienstes
Version:	Version der Operation
lang:	Sprachidentifikator
Service Identification:	Metadaten über den Server
Service Provider:	Metadaten über den Betreiber
Operations Metadata:	Metadaten über die Operation
Process Offerings:	Liste der angebotenen Funktionen
Languages:	Liste der unterstützten Sprachen

DescribeProcess

Die Operation *DescribeProcess* stellt eine Beschreibung mit den Merkmalen eines spezifischen Prozesses zur Verfügung. Der Name des Prozesses wird in der Anfrage über den verpflichtenden Parameter Identifier spezifiziert. Die Antwort erfolgt im xml-Format. Die Umsetzung von Anfragen via HTTP-Get ist für jeden WPS verpflichtend (OGC 2007). Anfragen via HTTP-Post zur Unterstützung von SOAP können optional implementiert werden (OGC 2007). Eine beispielhafte Anfrage via HTTP-Get für einen Prozess mit der Bezeichnung Buffer sieht wie folgt aus:

```
http://www.beispiel.com/geoserver/wps?
  Service=WPS&
  Version=1.1.0&
  Request=DescribeProcess&
  Identifier=Buffer
```

Der WPS muss die folgenden verpflichtenden Parameter in einer Describe-Process-Antwort zur Verfügung stellen (OGC 2007):

Bezeichnung	***Definition***
Service:	eindeutige Bezeichnung des Dienstes
Request:	Name der Operation
Version:	Version der Operation
Identifier:	eindeutige Bezeichnung des Prozesses

Execute

Mit der dritten verpflichtenden Operation *Exceute* wird eine Anfrage zur Durchführung einer Analyse mit spezifischen Eingabewerten und vorgegebenem Ausgabeformat definiert. Anfragen können via HTTP-Get oder HTTP-Post erfolgen. Aufgrund der Komplexität, die die Struktur einer WPS-Anfrage annehmen kann, erfolgen Anfragen in der Regel via HTTP-Post. Die Namen der spezifischen Parameter und die einzugebenden Werte und Attribute sind abhängig vom angeforderten Prozess.

Das folgende Beispiel zeigt eine WPS-Anfrage via HTTP-Post zur Berechnung eines Puffers mit angegebener Pufferdistanz. Der Name des Prozesses (Buffer) wird im Element Identifier angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute service="WPS" version="1.0.0"
xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_request.xsd">
  <ows:Identifier>Buffer</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>InputPolygon</ows:Identifier>
      <ows:Title>Buffer Request</ows:Title>
      <wps:Reference
schema="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd
"
xlink:href="http://demo.opengeo.org/geoserver/ows?Request=Get
Feature&amp;Service=WFS&amp;Version=1.1.0&amp;Typename=topp:t
asmania_water_bodies&amp;Outputformat=GML3" />
    </wps:Input>
    <wps:Input>
      <ows:Identifier>BufferDistance</ows:Identifier>
      <ows:Title>Buffer</ows:Title>
      <wps:Data>
        <wps:LiteralData>0.02</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="text/xml;
subtype=gml/3.1.1">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

Die zur Berechnung benötigten Geometriedaten werden als Referenz auf den entfernten WFS-Server (http://demo.opengeo.org/geoserver/ows) im URL-Encoding spezifiziert. Die Größe des Puffers (0.02) wird im Element Buffer-Distance angegeben. Die Dimension der Pufferdistanz ist im Beispiel nicht angegeben und richtet sich somit nach der Dimension des Referenzsystems der zugrundeliegenden WFS-Daten. Die WFS-Daten werden als geographische Koordinaten mit dem Referenzsystem EPSG:4326 angefordert. Die Pufferdistanz beträgt somit 0.02 Grad. Das Ausgabeformat (RawData) wird im

Element ResponseForm angegeben. Die aufgeführten Eingabe-Parameter sind grau hinterlegt.

Der WPS muss die folgenden verpflichtenden Parameter in einer Execute-Antwort zur Verfügung stellen:

Bezeichnung	***Definition***
`Service:`	`eindeutige Bezeichnung des Dienstes`
`Request:`	`Name der Operation`
`Version:`	`Version der Operation`
`Identifier:`	`eindeutige Bezeichnung des Prozesses`

Ausführungsmodi

Ein WPS stellt zwei Ausführungsmodi zur Verfügung. Die Verarbeitung kann synchron oder asynchron erfolgen. Synchron bedeutet, dass die Client-Anwendung wie im oben genannten Beispiel auf die Verarbeitung der Daten warten muss. Im Gegensatz dazu muss der Client bei einer asynchronen Verarbeitung nicht auf die Fertigstellung des Ergebnisdatensatzes warten. In beiden Fällen können die Ergebnisdaten optional auf dem Server gespeichert werden. Dem Client wird nach Fertigstellung der Verarbeitung eine entsprechende Referenz mitgeteilt. Der Ausführungsmodus wird über den optionalen Parameter *StoreExecute Response* gesteuert (OGC 2007).

Datentypen

Ein WPS muss die folgenden Datentypen unterstützen, die sowohl bei Anfragen als auch bei Antworten in xml-Dokumenten angewandt werden (OGC 2007).

ComplexData

Eine komplexe, generische Datenstruktur, die neben den Prozessdaten auch Angaben zum Encoding und den verwendeten Schemata enthält.

- LiteralData

  Einfache Struktur für Werte vom Datentyp String, Integer, Double oder Date, die in einen String eingebettet werden. Optional kann ein zusätzliches Attribut eingegeben werden (z.B. die Dimension der Pufferdistanz).

- BoundingBoxData

  Ein umschließendes Rechteck mit einer verpflichtenden Angabe des Referenzsystems. Weitere Referenzsysteme können optional angegeben werden.

Zustandslosigkeit

Die allen OGC-Spezifikationen zugrundeliegende *The OpenGIS Abstract Specification* (OGC 2002) fordert, dass alle OGC-konformen Dienste zustandslos zu modellieren sind. Dies bedeutet, dass Zustände nur vom Beginn der Anfrage bis zur Bereitstellung der Daten auf dem Server abgelegt werden dürfen. Um Abhängigkeiten von Zwischenergebnissen zu vermeiden, sollen nach der Bearbeitung ggf. temporär abgelegte Daten oder andere Spuren wieder vom Server entfernt werden (Papazoglou und Heuvel, 2007). Diese Forderung erschwert das Zwischenspeichern von Daten bei der Anwendung von WPS.

Im Rahmen dieser Arbeit wird die WPS-Operation Execute mit eingebetteten WFS-Daten benutzt.

2.2 Hypertext Transfer Protocol

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll mit dem Daten über ein Netzwerk transportiert werden. Der aktuelle HTTP-1.1-Standard wurde 1999 als Request für Comments Nr. 2616 (RFC 2616) publiziert (URL 2013).

Das HTTP-Protokoll ist ein zustandsloses Protokoll. Alle Anfragen werden unabhängig von vorigen Anfragen behandelt. Es wird zwischen den Nachrichtenarten Anfrage und Antwort unterschieden. Beide Nachrichtenarten bestehen aus einem Nachrichtenkopf (Header) und einem Nachrichtenkörper (Body). Header müssen immer vorhanden sein, Nachrichtenkörper sind optional.

Eine Anfrage, mit der die Ressource index.html vom Server www.beispiel.com angefordert wird, sieht vereinfacht wie folgt aus:

```
GET /index.html HTTP/1.1
Host: www.beispiel.com
```

Nachdem der Server die Anfrage verarbeitet hat, sendet er eine entsprechende Antwort. Eine beispielhafte Antwort sieht wie folgt aus:

```
HTTP/1.1 200 OK
Date: Thu, 01 Jan 2013 10:03:11 GMT
Content-Length: 12345
Content-Type: text/html
```

Der HTTP-1.1-Standard definiert die in Tabelle 2.1 aufgezählten Anfrage-Methoden (URL 2013).

Tabelle 2.1: HTTP-Header-Anfragen

Request-Methode	Beschreibung
Get	gebräuchlichste Methode; die Identifizierung erfolgt über den URI; die Länge des URI ist in der Regel auf 255 Zeichen begrenzt; die Parameter werden als Wertepaare mit vorangestelltem ? an die URL angehängt
Head	identisch mit Get, die Antwort enthält jedoch keinen Nachrichtenkörper; wird z.B. zur Prüfung der Gültigkeit von Cache-Objekten eingesetzt
Post	verarbeitet die im Nachrichtenkörper angegebenen Daten; im Gegensatz zur Methode Get kann eine (unbegrenzte) von den physikalischen Eigenschaften des Servers abhängige Menge an Daten an den Server gesandt werden
Put	speichert die Daten der im Nachrichtenkörper beigefügten Ressource auf dem Server
Trace	Loopback-Methode, die die Anfrage an den Client zurückliefert; wird in erster Linie zum Debugging eingesetzt
Delete	entfernt die in der URI angegebene Ressource auf dem Server
Options	fordert eine Liste der vom Server unterstützten Methoden und Features an
Connect	wird von Proxy-Servern benutzt, die für bestimmte Protokolle eine Tunnel-Verbindung zur Verfügung stellen können

Im Nachrichtenkopf einer Anfrage oder Antwort werden Metadaten ausgetauscht. Bei Anfragen werden im Nachrichtenkopf immer die HTTP-Version und die URI übermittelt, bei Antworten werden die HTTP-Version sowie der Statuscode der angefragten URI mitgesendet. Im Nachrichtenkopf werden auch Metadaten zur Steuerung von Caching-Funktionalitäten ausgetauscht.

2.3 Caching

In der Informatik wird unter dem Begriff Caching das Zwischenspeichern oder Puffern von Inhalten auf schnellen Speichermedien verstanden (Daniel Kuhn und Michael Raith, 2013). Insbesondere innerhalb einer Organisation neigen Anwender nach einer in der Informatik weit verbreiteten Regel dazu, auf bereits verwendete Ressourcen zuzugreifen (Badach et al. 2003). Ziel von Caching-Konzepten ist es, häufig verwendete Inhalte schnell zur Verfügung zu stellen, um diese nicht ständig von den (langsameren) Ursprungsmedien laden zu müssen. Cache ist ein Lehnwort aus dem Englischen. Es geht auf den französischen Begriff cache zurück, der die eigentliche Bedeutung Versteck besitzt.

Caches lassen sich nach der Art des Speichermediums und dessen Zugriffszeiten unterscheiden. Daten, die in einem flüchtigen Cache (Haupt- oder Registerspeicher der CPU) abgelegt werden, stehen nach einem Neustart oder Stromausfall nicht mehr zur Verfügung. Dagegen stehen in einem persistenten Cache (Festplatte oder SSD) gehaltene Daten weiterhin zur Verfügung. Da Caches meist auf schnellen und teuren Speichermedien untergebracht sind, werden diese aus Kostengründen nur für häufig benutzte Daten eingesetzt (Kuhn und Raith, 2013).

Caching-Konzepte werden heute in fast allen Computer- und Netzwerkbereichen eingesetzt (Kuhn und Raith 2013). CPUs verfügen über einen Daten- und einen Instruktionscache, die direkt in den Prozessor integriert sind. In CPUs wird meistens ein mehrstufiges Caching-Konzept verwendet. Die Caches werden von der CPU verwaltet. In Festplatten werden Daten im Laufwerks-Cache gesammelt, damit diese schneller vom Lesekopf zum Schreiben verarbeitet werden können. In optischen Laufwerken werden mit Caches Schwankungen in der Datenübertragung ausgeglichen. Laufwerks-Caches werden vom Betriebssystem verwaltet. Moderne Anwendungen oder Frameworks stellen eigene, teilweise konfigurierbare Caching-Mechanismen zur Verfügung. Caches werden in allen Computersystemen betrieben.

2.4 Web-Caching

In Web-Anwendungen werden Caches in verschiedene Schichten unterteilt. Kuhn und Raith (2013) unterscheiden die folgenden fünf Caching-Schichten:

- Clientschicht: Bei Web-Anwendungen ist dies der Browser des Anwenders. Moderne Browser besitzen eigene Caching-Mechanismen, die über eine Web-Anwendung oder einen Web-Server gesteuert werden (Kuhn und Raith 2013).
- Anwendungsschicht: In der Anwendungsschicht wird das auf Anwendungsfälle bezogene Caching-Verhalten durch die Implementierung der Software bestimmt. Die Caches dienen meist zum Speichern von Daten oder Zuständen auf Festplatten (Kuhn und Raith 2013).
- Caching-Schicht: Zwischen jeder Schicht kann zur Entlastung der Anwendungsschicht und aller dahinterliegenden Schichten eine separate Caching-Schicht eingesetzt werden. Die Caching-Schicht kann auf einem lokalen Server oder auf mehreren Servern verteilt implementiert werden (Kuhn und Raith 2013).
- Datenbankschicht: Moderne Datenbanksysteme besitzen eigene Caches, mit denen beispielsweise Anfragen und deren Ergebnisse gepuffert werden.
- Fremdsystem: Fremdsysteme können eigene Caching-Mechanismen besitzen, die als Black-Box betrachtet werden müssen. Allerdings können Anfragen zu Fremdsystemen durch eine eigene Caching-Schicht reduziert werden (Kuhn und Raith 2013).

Diese Arbeit beschäftigt sich mit der Caching- und der Anwendungsschicht. In den nächsten Kapiteln werden entsprechende Konzepte und Vorgehensweisen vorgestellt.

2.4.1 Ziele von Web-Caching

Mit Web-Caching werden in erster Linie die folgenden zwei Ziele verfolgt:

- Verringerung der Zugriffszeit: Da sich der Cache näher am Client befindet als der Ursprungsserver, können dem Client die angefragten Objekte schneller zur Verfügung gestellt werden. (Wessels 2001).

- Reduzierung der Bandbreite: Da Objekte nur einmal vom Ursprungsserver abgerufen werden, lassen sich die Anforderungen an die Bandbreite reduzieren und Kosten einsparen (Wessels 2001). Befinden sich die angefragten Objekte im Cache, können in der gleichen Zeit in der Regel mehr Daten übertragen werden.

2.4.2 Caching-Strategien

Caching-Konzepte dienen dazu, die Performance einer Anwendung zu verbessern. Die Größe von Caches ist im Verhältnis zur Größe der Ursprungsserver eher klein. Die Speichermedien von Caches sind in der Regel schneller als die der Ursprungsserver (Kuhn und Raith 2013). Daten, die häufig benötigt werden eignen sich besser zum Cachen als Daten, die nur selten angefragt werden. Wird der gleiche Datensatz von einer Organisation häufig benötigt, so ist es sinnvoll die Daten im Cache abzulegen. Im Gegensatz dazu sind stark wechselnde Anfragen, weniger gut zum Cachen geeignet, da die Möglichkeit gering ist, dass eine identische Anfrage ein weiteres Mal gestellt wird. Statische Daten, die sich eher selten ändern, sind besser zum Cachen geeignet als Daten, die sich sehr schnell ändern. Bei der Konzeption von Caching-Strategien müssen nach Kuhn und Raith (2013) folgende Punkte beachtet werden:

- Caching-Dauer: Die Caching-Dauer ist das Zeitfenster, in dem Objekte im Cache zur Verfügung stehen. Sie sollte an Hand der Aktualität und Nutzungsdauer der Daten festgelegt werden. Für statische Daten, die sich kaum ändern, kann eine längere Caching-Dauer festgelegt werden. Darüber hinaus hängt die Caching-Dauer von der Größe der Daten und deren Ladezeiten ab. Zur Verbesserung der Performance können Daten mit langen Ladezeiten länger im Cache vorgehalten werden. Die Caching-Dauer stellt einen Kompromiss zwischen der Aktualität der Daten und deren Ladezeiten dar. Sie beeinflusst die Performance der Anwendung (Kuhn und Raith, 2013).
- Gleichartigkeit: Aus Sicht der Performance ist es sinnvoll gleichartige Daten, die häufig angefragt werden, zu cachen. Bei unterschiedlichen Anfragen entscheidet die Häufigkeit der Zugriffe auf diese einzelnen Anfragen. Ist die Wahrscheinlichkeit gegeben, dass Daten mehrfach abgefragt werden und ist die Generierung der Daten aufwändig, kann sich Cachen für eine bestimmte Dauer anbieten (Kuhn und Raith 2013).

- Zugriffshäufigkeit: Die Notwendigkeit zum Cachen von Daten steigt mit der Häufigkeit der Anfragen. Abhängig von der Geschwindigkeit des Caches, kann sich das Zwischenspeichern bereits beim zweiten Zugriff lohnen (Kuhn und Raith 2013).
- Invalidierungsstrategie: Bei der Invalidierung werden einzelne Objekte aus dem Cache entfernt, um den Speicherplatz für weitere Verwendungen freizugeben. Dies kann automatisch durch Verdrängungsstrategien oder gezielt erfolgen, da die Daten nicht mehr benötigt werden (Kuhn und Raith 2013).
- Größe des Caches: Zur Bemessung der richtigen Größe des Caches, sollte die Größe der zu speichernden Daten bekannt sein. Darüber hinaus sollte die Größe des Caches an die Caching-Dauer der Daten und deren Aktualität angepasst werden. Ist der Cache zu klein bemessen, werden meist die ältesten Daten auf Grund von Verdrängungsstrategien entfernt (Kuhn und Raith 2013).

In dieser Arbeit sind die Festlegung der Caching-Dauer und die Prüfung der Gleichartigkeit von Daten bzw. der zugrundeliegenden Anfragen von besonderer Bedeutung.

2.4.3 Arten von Web-Caches

Web-Caches können nach ihren Einsatzmöglichkeiten unterschieden werden.

- Browser-Cache: Browser-Caches gehören zu den clientseitigen Caches. Beim Aufruf einer Webseite wird bei modernen Browsern eine Kopie der Inhalte auf der lokalen Festplatte abgelegt. Bei einem erneuten Aufruf dieser Webseite können die Inhalte aus dem Cache geladen werden. Sind die Inhalte im Cache nicht verfügbar, werden sie vom Server nachgeladen. Ein Browser-Cache ist auf einen Anwender beschränkt (Wessels, 2001). Bei wiederholtem Zugriff werden Inhalte schneller dargestellt, der Netzwerkverkehr wird deutlich reduziert. (Kuhn und Raith, 2013). Diese Caching-Möglichkeit ist in allen modernen Browsern vorhanden und lässt sich über HTTP-Header steuern. Die Lösung eignet sich jedoch nur für statische Daten. (Kuhn und Raith, 2013).
- Proxy-Cache (Forward-Proxy): Proxy-Caches gehören zu den serverseitigen Caches, die Anfragen von Clients aus dem eigenen lokalen Netzwerk entge-

gennehmen. Im Gegensatz zu Browser-Caches können Proxy-Caches mehrere Anwender bedienen, wodurch die Effizienz gesteigert wird (Wessels 2004). Proxy-Caches haben höhere Trefferquoten als Browser-Caches (Wessels, 2001). Die Trefferquote steigt mit zunehmender Benutzerzahl (Wessels, 2001).

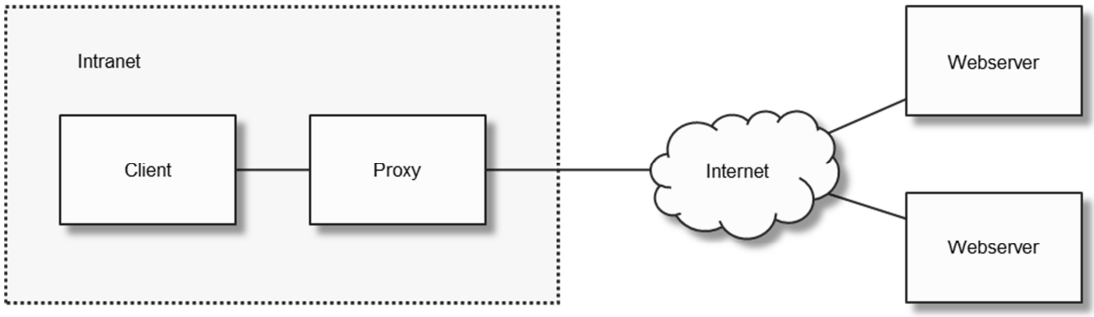


Abbildung 2.1: Proxy-Cache; der Proxy-Cache nimmt Anfragen aus dem lokalen Netzwerk entgegen; (eigene Darstellung)

Caching-Proxys werden meist in der Nähe von Gateways oder Routern angeordnet, über die die Verbindung mit dem Internet hergestellt wird (Wessels, 2001). Moderne Browser können so konfiguriert werden, dass der HTTP-Verkehr über einen Proxy umgeleitet werden kann (Wessels, 2001). Sollen Anfragen einer Web-Anwendung, die keine Möglichkeit zur Umleitung des HTTP-Verkehrs bietet, über einen Proxy umgeleitet werden, so muss die Anwendung softwaretechnisch angepasst werden.

- Reverse Proxy Cache: Reverse Proxy Caches sind serverseitige Caches, die Anfragen aus dem Internet entgegen nehmen und diese an die Server im eigenen Netzwerk weiterleiten. Sie werden auch als HTTP-Accelerator oder Surrogate Proxy bezeichnet. Ein Reverse Proxy Cache puffert Ressourcen, die häufig aus dem Internet abgerufen werden. Die Clients stellen eine Verbindung zum Proxy Server her; die dahinterliegenden Server bleiben den Clients in der Regel verborgen. Reverse Proxy Caches können die Performance von Web-Anwendungen stark verbessern, sofern deren Inhalte sich nicht häu-

fig ändern (Kuhn und Raith, 2013). Reverse Proxy Caches werden häufig von Internet Service Providern betrieben (Wessels, 2001).

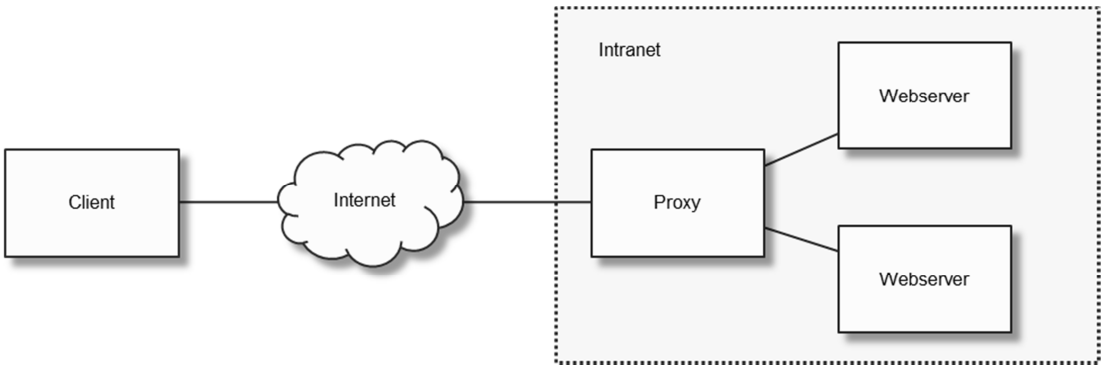


Abbildung 2.3: Reverse-Proxy-Cache; der Reverse Proxy nimmt Anfragen aus dem Internet entgegen; (eigene Darstellung)

Ein Proxy oder Stellvertreter ist eine Kommunikationsschnittstelle in einem Netzwerk oder Internet. Der Proxy-Server stellt stellvertretend für einen Client eine Anfrage an einen Server, nimmt die Antwort des Servers entgegen und leitet die Antwort schließlich an den Client zurück. Es besteht keine direkte Verbindung zwischen dem Client und dem Server. Die Verbindung zwischen Client und Server wird vollkommen getrennt (Dithardt 2007).

In dieser Arbeit werden Strategien für Proxy-Server betrachtet bei denen der WPS quasi als Client fungiert.

2.4.4 Funktionsweise von Web-Caching

Caching basiert auf Regeln, nach denen der Cache bestimmt, ob ein Objekt ausgeliefert wird, sofern es vorhanden ist. Die Datenübertragung zwischen einem Client und einem Cache oder Server erfolgt über das Hypertext Transfer Protocol (HTTP). Es werden zwei Nachrichtenarten unterschieden:

- Anfrage (Request) vom Client an den Server
- Antwort (Response) vom Server an den Client

Server und Cache teilen sich über Metadaten in den Anfragen oder Antworten Anweisungen zum Caching mit (Kuhn und Raith, 2013). Die Regeln werden in den HTTP-Protokollen oder mittels Konfiguration der Proxy-Server festgelegt.
Auf Grund der Regeln kann ein Server einem Client oder Cache beispielsweise das Zwischenspeichern untersagen. Freshness (Frische) und Validation (Gültigkeit) sind zwei wichtige HTTP-Mechanismen um Caches zu kontrollieren. Ein frisches Objekt kann ohne Nachfrage beim Ursprungsserver ausgeliefert werden. Über eine Nachfrage beim Ursprungsserver kann die Gültigkeit eines Objektes validiert werden.
Die Freshness wird über den Expires-HTTP-Header gesteuert, der eine HTTP-Zeitangabe enthält (URL 2013). Mit diesem Header teilt der Webserver dem Cache den Verfallzeitpunkt der Ressource mit. Der Cache speichert die vom Client angeforderten Daten mit dem Verfallzeitpunkt. Bei weiteren Anfragen vor dem Verfallzeitpunkt, liefert der Cache das angeforderte Objekt ohne Nachfrage beim Ursprungsserver aus. Nach dem Verfallzeitpunkt fordert der Cache das Objekt vom Ursprungsserver erneut an. Die Header werden serverseitig aktiviert und in der Regel für statische Objekte mit einer langen Lebenszeit eingesetzt (kuhn und Raith 2013).

Mit Cache-Control-HTTP-Headern, die mit dem HTTP-1.1-Standard eingeführt wurden, kann der Server weitere Metadaten an den Cache übermitteln. Mit den neu hinzugekommenen Headern wurden die Beschränkungen des Expires-Header aufgehoben. Beispiele für Cache-Control-Header sind:

max-age:	gibt die maximale Zeitdauer an, in der ein Objekt als frisch angesehen werden kann
s-maxage:	bezieht sich im Gegensatz zu max-age nur auf gemeinsame Caches
no-cache:	der Cache muss die Anfrage jedes Mal beim Ursprungsserver validieren

Mit Validatoren kann geprüft werden, ob eine Ressource noch frisch ist. Ein häufig genutzter Validator ist der Last-Modified-Header mit dem der Server mitteilt, wann ein Objekt zuletzt geändert wurde. Der Cache kann diesen Header nutzen und seinerseits eine If-Modified-Since-Anfrage an den Ursprungsserver richten. Hierbei fragt der Cache den Server, ob das gespeicherte Objekt noch gültig ist. Der Server antwortet mit Status 304, wenn das Objekt nicht verändert wurde (Kuhn und Raith 2013).

ETag ist ein weiterer Header mit dem Caches gesteuert werden können. Mit Etags werden die ausgelieferten Dateien im Server markiert (Kuhn und Raith, 2013).

3 Konzeption

Im ersten Teil des Kapitels werden aufbauend auf den theoretischen Grundlagen Anforderungen für verschiedene WPS-Caching-Techniken vorgestellt. Zur Beschreibung und Dokumentation der Anforderungen werden Anwendungsfälle, Sequenzdiagramme und Komponentendiagramme der Unified Modeling Language (UML) herangezogen. Im zweiten Teil des Kapitels werden die Anforderungen spezifiziert und in Form von Aktivitätsdiagrammen der UML aufbereitet. Im dritten Teil des Kapitels werden Anforderungen für die Entwicklung eines Messwerkzeugs zur Durchführung von Lasttests vorgestellt.

3.1 Benachbarte Arbeite

Die folgende Auflistung zeigt Forschungsvorhaben, die sich mit OGC Web Processing Services oder Web-Caching-Szenarien für OGC Services beschäftigen und Bezug zu dieser Arbeit haben:

- Loechel und Schmid (2012) stellen Grundlagen für Web-Caching basierend auf Geo-Tile-Caching, Reverse-Proxy-Caching und Web Application Acceleration für OGC Web Map Services vor.
- Brauner (2008) hat die Möglichkeit klassische GIS-Funktionalitäten über einen WPS-Wrapper zur Verfügung zu stellen untersucht.

- Béjar et al (2012) haben den Aufbau eines WFS-Caches über ETL-Prozesse (Extract, Transform, Load) zur Verbesserung der Performance von WPS im EuroGeoSource System untersucht.

3.2 WPS-Caching-Varianten

3.2.1 Grundlegende Überlegungen zum WPS-Caching

Ein WPS stellt Funktionen für räumliche Analysen von Geodaten über standardisierte Schnittstellen zur Verfügung. Der Client übergibt in einer Execute-Operation die Eingabeparameter an den WPS. Der WPS fordert die zur Durchführung der Analyse benötigten Daten von den WFS-Servern an. Die WFS-Server können im lokalen Netzwerk oder entfernt im Internet stehen. Danach erfolgt die Verarbeitung durch den WPS, d.h. es wird beispielsweise eine Puffer-Berechnung durchgeführt. Das Verarbeitungsergebnis wird dem Client im angeforderten Format zur Verfügung gestellt. Der generelle Ablauf einer WPS-Execute-Operation kann Abbildung 3.1 in Form eines Sequenzdiagramms entnommen werden.

Ein Sequenzdiagramm ist eine der Diagrammarten in der UML. Es zeigt den Informationsaustausch zwischen beliebigen Kommunikationspartnern eines bestimmten begrenzen Kontextes unter Betonung der zeitlichen Abfolge (Rupp et al., 2007). Zu den Grundelementen dieser Diagrammart gehören die Kommunikationspartner und die Nachrichten, die zwischen den Partnern ausgetauscht werden. Die Kommunikationspartner werden durch senkrecht verlaufende, gestrichelte sogenannte Lebenslinien repräsentiert. Die Nachrichten werden durch eine durchgezogene Linie mit Pfeil, Antworten als gestrichelte Linie mit Pfeil repräsentiert.

Aus dem Sequenzdiagramm lässt sich erkennen, dass der Gesamtprozess aus zwei Anfragen besteht. Der Client richtet die eigentliche WPS-Anfrage an den WPS. Der WPS extrahiert die Client-Anfrage (in Abbildung 3.1 nicht dargestellt) und richtet seinerseits eine GetFeature-Anfrage an den WFS-Server. Der WFS liefert dem WPS den angeforderten Datensatz. Der WPS führt mit diesem Datensatz die vom Client angeforderte Analyse durch und stellt dem Client das Ergebnis zur Verfügung.

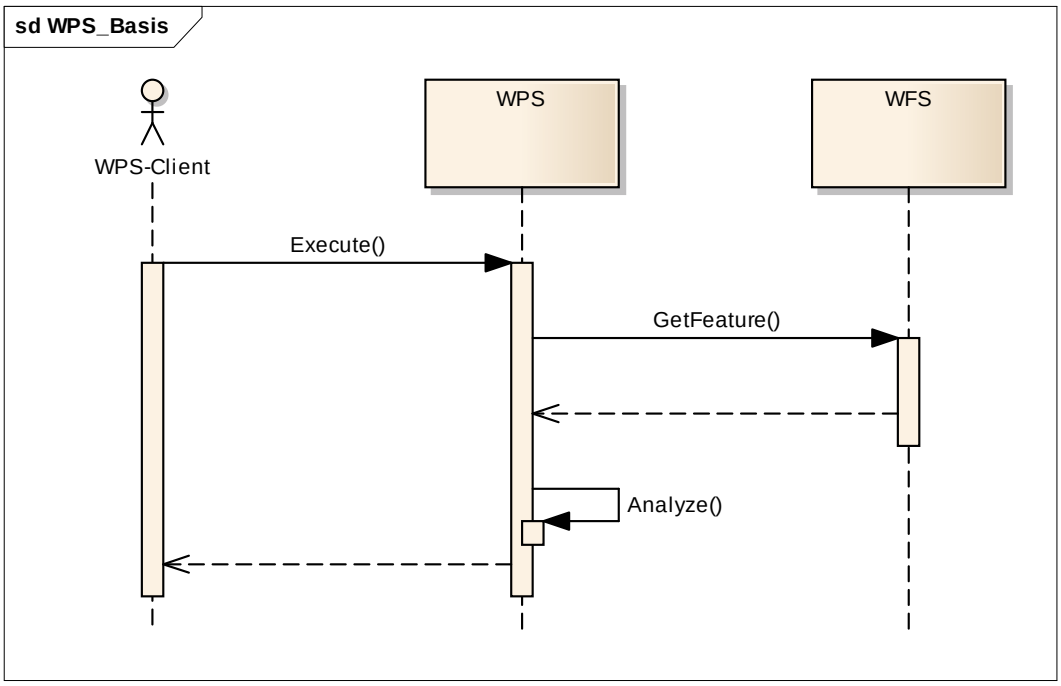


Abbildung 3.1: Ablauf einer einfachen WPS-Anfrage
1. Client stellt WPS-Anfrage; 2. WPS stellt *GetFeature*-Anfrage; 3. Server liefert WFS-Daten; 4. WPS führt Analyse durch; 5. WPS liefert Analyse-Ergebnis an Client; (eigene Darstellung)

In den folgenden Abschnitten werden Lösungsansätze zum Zwischenspeichern der WFS-Daten und der WPS-Analyseergebnisse gesucht. Es können folgende Caching-Varianten unterschieden werden:

1. Caching von WFS-Daten
   Hierunter ist das Zwischenspeichern von WFS-Daten zu verstehen, die zur Durchführung einer vom Client angeforderten WPS-Analyse benötigt werden. Die WFS-Daten werden im Nachrichtenkörper der HTTP-Post-Anfrage als Referenz übergeben. Werden die gleichen WFS-Daten - mit identischen Parametern - erneut als Referenz in einer WPS-Anfrage übergeben, können die Daten aus dem Cache zur Verfügung gestellt werden. Für diesen Zwischenspeicher wird die Bezeichnung Input-Cache eingeführt. Der WPS nimmt gegenüber dem WFS-Server die Funktion eines Clients ein.
2. Caching von WPS-Ergebnisdaten
   Hierunter ist das Zwischenspeichern der vom WPS durchgeführten Analyseergebnisse zu verstehen. Wird die gleiche Analyse - mit identischen Parametern - erneut angefordert, muss die Berechnung nicht wiederholt werden.

Die Daten können aus dem Cache zur Verfügung gestellt werden. Für diesen Zwischenspeicher wird die Bezeichnung Output-Cache eingeführt.

Die Funktionsweise von Input- und Output-Cache ist in Abbildung 3.2 als Komponentendiagramm der UML dargestellt. Ein Komponentendiagramm ist ein Strukturdiagramm, das die verschiedenen Bestandteile eines Systems als Komponenten darstellt (Rupp et al, 2007). Es stellt die Organisation der Komponenten mit ihren Schnittstellen oder Ports zur Laufzeit mit den sich daraus ergebenden Abhängigkeiten dar. Input- und Output-Cache werden in den folgenden Abschnitten separat behandelt; sie können auch in Kombination angewandt werden.

Abbildung 3.2: Input- und Output-Cache; der Input-Cache puffert die WFS-Daten, er befindet sich zwischen dem WPS und dem (entfernten) WFS-Server; der Output-Cache puffert die Analyse-Ergebnisse und befindet sich zwischen Client und WPS; (eigene Darstellung)

Wie im zweiten Kapitel ausgeführt muss ein OGC-konformer WPS-Dienst zustandslos modelliert sein. Standardmäßig werden die Analyseergebnisse in der angeforderten Kodierung an den Client ausgeliefert. Die WPS-Spezifikation lässt jedoch eine Ausnahme zu. Die Analyseergebnisse zeitaufwändiger oder verketteter Prozesse können zum späteren Abruf auf dem Server abgelegt werden. Diese Möglichkeit kann mit dem Parameter *storeSupported* angefordert werden (OGC 2007).

Einerseits soll der WPS alle Spuren nach der Verarbeitung beseitigen. Andererseits möchte man aus Performancegründen redundante Datenübertragungen zwischen WPS und WFS vermeiden. Dieser Widerspruch lässt sich überwinden, wenn verkettete WPS-Anfragen in einem eigenen, temporären Workspace ausgeführt werden. Werden identische Datensätze innerhalb der Prozesskette wiederholt benötigt, ließe sich der Aufwand für das wiederholte Im- und Exportieren sowie das Parsen der Datensätze vermeiden (Brauner 2008). Die der WPS-Anfrage zugrundeliegenden Daten werden nur einmal vom WFS-Server angefragt und im Workspace zwischengespeichert. Wird der identische WFS-Datensatz im Verlauf ein weiteres Mal benötigt, so

kann er aus dem Workspace abgerufen werden. Nach Brauner (2008) wird durch die Wiederverwendung von Datensätzen die WPS-Spezifikation und somit die Interoperabilität verletzt.

3.2.2 Caching von WPS-Eingangsdaten (Input-Cache)

In diesem Abschnitt werden Möglichkeiten für das Zwischenspeichern von WFS-Daten aufgezeigt. Bei den Überlegungen werden Caching-Varianten betrachtet, die serverseitig implementiert werden. Mit den oben dargelegten Grundüberlegungen und den im vorangegangenen Kapital vorgestellten Grundlagen sind für die WPS-Eingangsdaten drei Caching-Varianten denkbar:

1. Puffern von WFS-Daten mit einem Standard-Proxy,
2. Puffern von WFS-Daten mit einem transparenten Proxy sowie einem Router oder Switch zur Umleitung des HTTP-Verkehrs,
3. WPS implementiert einen eigenen File-Cache.

Die Caching-Varianten werden in den folgenden Abschnitten vorgestellt.

3.2.2.1 Variante 1: Standard Proxy

Proxy-Server wurden eingeführt, um stellvertretend für einen Client, Anfragen an einen Server durchzuführen. Der Client fragt den Proxy. Der Proxy fragt den Web-Server und bekommt seine Antwort direkt vom Web-Server. Der Proxy leitet die Antwort an den Client weiter. Dabei besteht keine direkte Verbindung zwischen dem Client und dem Web-Server. Neben diesem Sicherheitsaspekt bieten Proxy-Server eine Caching-Funktionalität. Der Proxy-Server behält eine Kopie des angeforderten Objektes in seinem Zwischenspeicher (Cache). Wird das Objekt ein weiteres Mal angefordert, kann der Proxy das Objekt aus seinem Cache zur Verfügung stellen ohne eine Anfrage an den entfernten Web-Server zu stellen.

Überträgt man diese nach Wessels (2004) seit Jahren etablierte Caching-Technik auf das Zwischenspeichern von WPS-Eingangsdaten, so muss der HTTP-Verkehr zwischen dem WPS und dem WFS-Server über einen Caching-Proxy umgeleitet werden. Der WPS fungiert gegenüber den WFS-Servern als Client. Dies entspricht der in

Browsern implementierten Möglichkeit, den HTTP-Verkehr über einen Proxy-Server umzuleiten. In den gängigen Browsern Microsoft Internet Explorer, Mozilla Firefox und Google Chrome kann die Netzwerkadresse eines zwischengeschalteten Proxy-Servers zur Umleitung des HTTP-Verkehrs angegeben werden. Der generische WPS der Firma deegree stellt diese Möglichkeit ebenso zur Verfügung (URL 2013). Bietet der WPS keine Möglichkeit zur Konfiguration, muss der Programmcode des WPS softwaretechnisch angepasst werden. Der Ablauf dieser Variante ist in Tabelle 3.1 als Anwendungsfall dargestellt.

Anwendungsfälle sind seit einigen Jahren fester Bestandteil der Systemanalyse. Sie beschreiben das Zusammenspiel zwischen Akteuren (Benutzern) und den Systemkomponenten sowie die bereitzustellenden Funktionalitäten. Rupp et al. (2007) definieren einen Anwendungsfall als eine Menge von Aktionen, die schrittweise ausgeführt, ein spezielles Verhalten formen. Dabei wird der Anwendungsfall stets von einem Akteur oder Stakeholder ausgelöst und führt zu einem fachlichen Ergebnis. Es soll die Sicht des Akteurs und nicht die Sicht des Systems widergegeben werden.

Anwendungsfälle können in natürlicher Sprache formuliert oder graphisch aufbereitet werden. Cockburn (2003) wie auch Balzert (2000) empfehlen bei der Beschreibung von Anwendungsfällen auf graphische Hilfsmittel zu verzichten. Anwendungsfälle sollten nach Cockburn (2003) so formuliert werden, dass sie ohne Spezialkenntnisse zu verstehen sind. Die richtige Interpretation einer graphischen Aufbereitung setzt ein gewisses Maß an Fachkenntnissen voraus. Nach Rupp et al. (2007) werden Anwendungsfälle in der Praxis am häufigsten in natürlicher Sprache formuliert, sie sehen jedoch auch Nachteile. Kontrollanweisungen und Nebenläufigkeiten lassen sich nur mühselig und zeitaufwändig erstellen; nachträgliche Änderungen sind kaum möglich, ohne den gesamten Text zu lesen.

Cockburn (2003) unterscheidet zwischen Systemanwendungsfällen und Geschäftsprozessen. Systemanwendungsfälle beschreiben das technische Zusammenspiel der am System beteiligten Softwarekomponenten. Geschäftsprozesse beschreiben die funktionalen Anforderungen aus Sicht des Benutzers.

Tabelle 3.1: Anwendungsfall „WFS-Anfrage mit Proxy-Caching“

Bezeichnung (ID):	WFS-Anfrage mit Proxy-Caching (Use Case 01)
Aufgabe:	Anfrage von WFS-Daten durch den WPS mit Caching-Proxy
Name:	WFS-Anfrage mit Proxy-Caching
Kurzbeschreibung:	Der Anwendungsfall beschreibt eine vom WPS initiierte WFS-Anfrage, die über einen Caching-Proxy umgeleitet wird. Die WFS-Daten werden als Referenz in der WPS-Anfrage des Clients übergeben.
Akteure:	WPS, WFS, Caching-Proxy, Benutzer als Auslöser
Vorbedingung:	Der Client hat einen Execute-Request mit referenzierten WFS-Daten via HTTP-Post an den WPS gesendet. Der WPS hat eine Vorverarbeitung durchgeführt und die WFS-Anfrage aus der WPS-Anfrage extrahiert. Die WFS-Daten liegen auf einem (entfernten) Server vor. Es besteht eine Netzverbindung zwischen WPS und WFS. Der Proxy steht zur Verfügung. Der WPS hat die Aufgabe die WFS-Daten abzufragen. Die Anfrage wird über den Proxy umgeleitet.
Nachbedingung:	Die WFS-Daten stehen dem WPS für die nachfolgende Analyse zur Verfügung.
Standardablauf:	1. Der WPS stellt eine GetFeature-Anfrage via HTTP-Get an den (entfernten) WFS-Server. Die Anfrage wird über den Proxy umgeleitet. 2. Der Proxy prüft (1) ob, die WFS-Daten in seinem Cache verfügbar sind, (2) ob, die Gültigkeit der Daten innerhalb des im Proxy eingestellten Gültigkeitszeitraums liegt. Falls beides (1) und (2) zutrifft: a. Der Proxy erstellt einen internen Protokolldatensatz. b. Der Proxy stellt dem WPS die WFS-Daten zur Verfügung. c. Der WPS erstellt einen Protokolldatensatz. d. Der Ablauf ist beendet. Falls die Daten im Cache verfügbar (1) sind, die Gültigkeit (2) jedoch außerhalb des eingestellten Zeitraums liegt: a. Der Proxy prüft die Gültigkeit über eine „If-Modified-Since“-Anfrage via HTTP-Head an den WFS-Server Falls die Daten im Cache gültig sind:

Bezeichnung (ID):	WFS-Anfrage mit Proxy-Caching (Use Case 01)
	a) Der Proxy erstellt einen Protokolldatensatz. b) Der Proxy stellt dem WPS die WFS-Daten aus seinem Cache zur Verfügung. c) Der WPS erstellt einen Protokolldatensatz. d) Der Ablauf ist beendet. Falls die Daten im Cache nicht mehr aktuell sind: a) Der Proxy fordert die Daten vom WFS-Server an. b) Der Proxy legt die WFS-Daten in seinem Cache ab. c) Der Proxy erstellt einen Protokolldatensatz. d) Der Proxy stellt dem WPS die WFS-Daten zur Verfügung. e) Der WPS erstellt einen Protokolldatensatz. f) Der Ablauf ist beendet. falls keine der unter 2. genannten Bedingungen zutrifft: (1) Der Proxy fordert die Daten direkt vom WFS-Server an. (2) Der Proxy legt die Daten im Cache ab. (3) Der Proxy erstellt einen Protokolldatensatz. (4) Der Proxy stellt dem WPS die WFS-Daten zur Verfügung. (5) Der WPS erstellt einen Protokolldatensatz. (6) Der Ablauf ist beendet.

Abbildung 3.3 zeigt den Ablauf dieser Execute-Operation. Die WFS-Datei ist im Cache des Proxys verfügbar, jedoch nicht mehr gültig. Der Proxy muss die WFS-Daten vom Server anfordern.

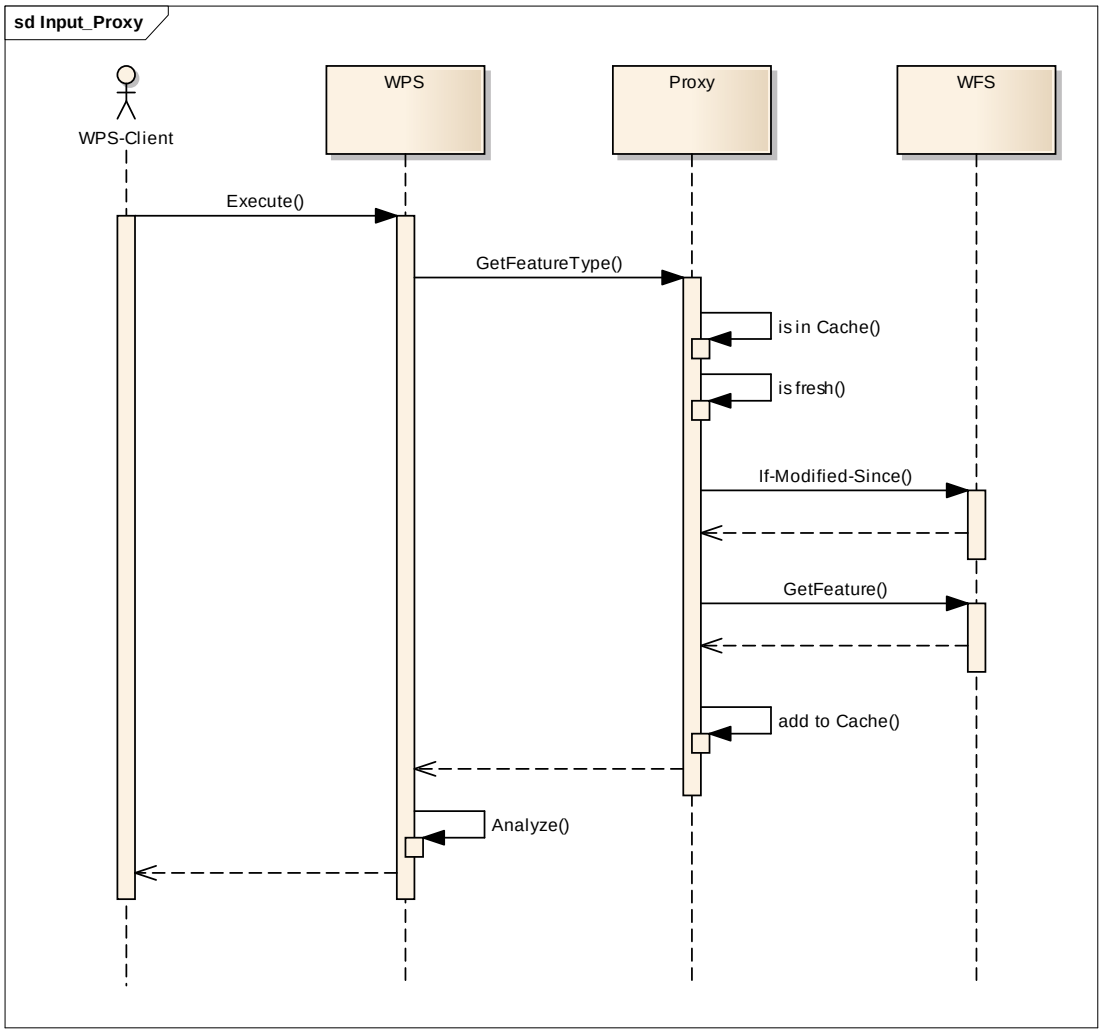


Abbildung 3.3: Ablauf einer WPS-Execute-Operation
1. Client stellt WPS-Anfrage; 2. WPS stellt WFS-Anfrage, die über einen Proxy umgeleitet wird; 3. Proxy prüft, ob Datei im Cache ist; 4. Proxy prüft die Gültigkeit an Hand der Parameter der Konfigurationsdatei; 5. Proxy stellt *If-Modified-Since*-Anfrage; 6. WFS-Server antwortet mit HTTP-Status 200 (OK); 7. Proxy stellt WFS-Anfrage; 8. WFS-Server liefert Daten; 9. Proxy übernimmt die Daten in seinen Cache; 10. Proxy liefert WFS-Daten an WPS; 11. WPS führt Analyse durch; 12. WPS liefert Analyse-Ergebnis an Client; (eigene Darstellung)

Die zur Umsetzung dieser Variante erforderliche Spezifikation wird im Abschnitt 3.2.4 vorgestellt.

3.2.2.2 Variante 2: Transparenter Proxy

Eine besondere Form des Proxys ist der sogenannte transparente Proxy. Transparente Proxys werden eingesetzt, wenn der Client die Verwendung eines Proxys nicht bemerken soll (Dithardt, 2007). Der Proxy ist für den Client unsichtbar oder transparent. Der HTTP-Verkehr zwischen Client und dem Server wird mit Netzwerktechniken umgeleitet. Dabei werden die folgenden zwei Varianten unterschieden:

(1) Umleitung durch *Policy based Routing*, für den ein Rechner mit zwei Netzwerkkarten benötigt wird. Der Rechner wird in den Datenstrom zwischen WPS und den (entfernten) WFS-Servern eingerichtet. Der ankommende HTTP-Verkehr wird nach konfigurierbaren Regeln ausgefiltert und ggf. auf den Proxy umgeleitet (Wessels, 2001 und Dithardt, 2007).

(2) Der HTTP-Verkehr zwischen dem WPS und den (entfernten) WFS-Servern wird über einen WCCP-fähigen Router ausgefiltert. Das hierzu verwendete Web Cache Coordination Protocol (WCCP) ist ein von Cisco entwickeltes Content-Routing Protocol zur Umleitung von HTTP-Verkehr in Echtzeit. Das Protokoll wird von mehreren Proxys unterstützt (Wessels, 2001 und Dithardt, 2007). Der WCCP-fähige Router analysiert die Anfrage und bestimmt an Hand der Port-Nummer, ob eine Umleitung zum Proxy erfolgen soll. Abbildung 3.4 zeigt das Zusammenwirken der Komponenten.

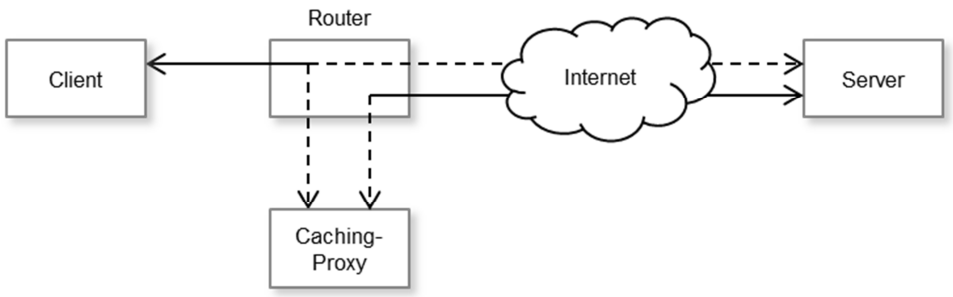


Abbildung 3.4: Filtern des HTTP-Verkehrs mit einem WCCP-fähigen Router (eigene Darstellung)

Bei beiden Varianten sind keine softwaretechnischen Anpassungen des WPS notwendig.

3.2.2.3 Variante 3: WPS-File-Cache

Bei dieser Variante implementiert der WPS einen eigenen Cache. Die zu speichernden Daten werden in einem Serververzeichnis des WPS abgelegt. Der Lösungsansatz basiert auf der Annahme, dass der WPS die angeforderten WFS-Daten zur weiteren Verarbeitung ohnedies temporär speichert. Die vorhandene temporäre Zwischenablage der WFS-Daten wird zu einem permanenten Input-Cache erweitert werden. Abbildung 3.5 zeigt den Ablauf dieser Lösung.

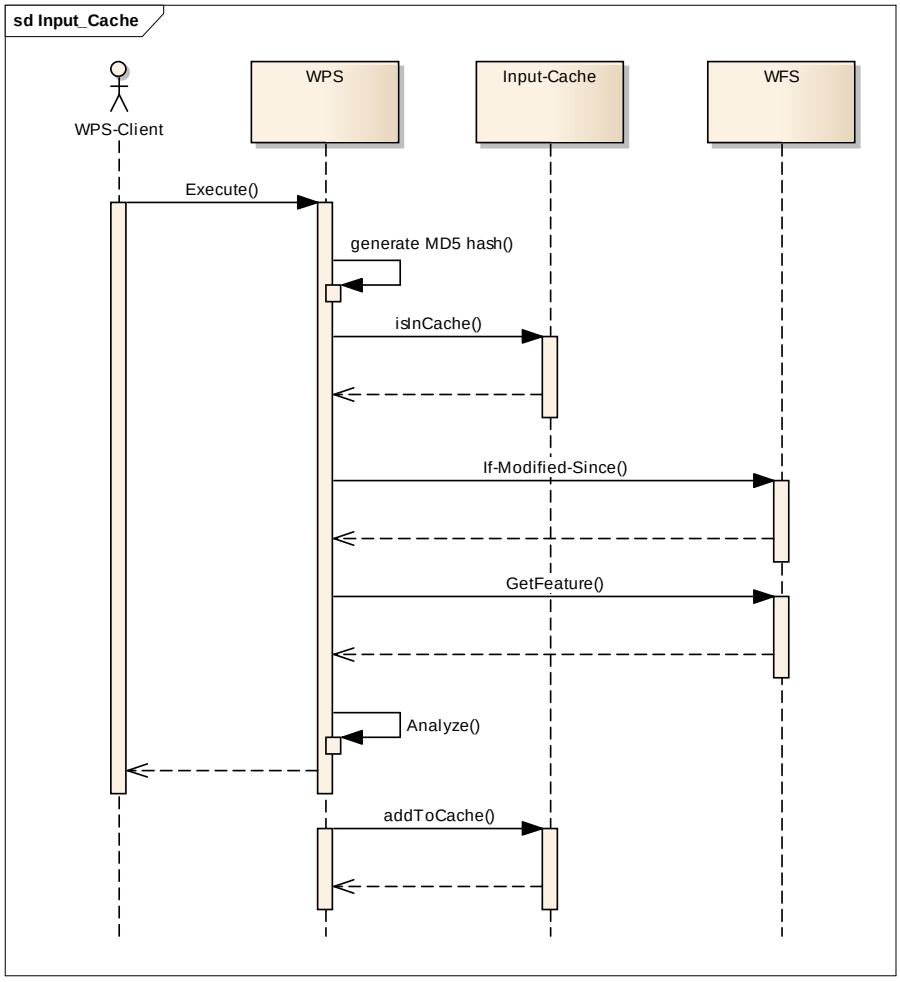


Abbildung 3.5: Ablauf des WPS-File-Caching
1. Client stellt WPS-Anfrage; 2. WPS generiert einen MD5-hash; 3. WPS prüft, ob eine zur Anfrage passende Datei im Input-Cache vorhanden ist; 4. Input-Cache antwortet: Datei ist vorhanden; 5. WPS stellt *If-Modified-Since*-Anfrage; 6. WFS-Server antwortet mit HTTP-Status 200; 7. WPS stellt WFS-Anfrage; 8. WFS-Server liefert Daten; 9. WPS führt Analyse durch; 10. WPS überführt Analyse-Ergebnis in Input-Cache; 11. Input-Cache bestätigt Übernahme der Daten; 12. WPS liefert Analyse-Ergebnis an Client; (eigene Darstellung)

Um den WFS-Anfragen die Cache-Objekte zuordnen zu können, wird für jede WFS-Anfrage ein eindeutiger Schlüssel generiert. Aus dem generierten Schlüssel wird ein eindeutiger Name für die Cache-Datei gebildet. Die Methode zur Generierung eines eindeutigen Schlüssels wird im nächsten Abschnitt beschrieben. Die zur Umsetzung dieser Variante erforderliche Spezifikation wird in Abschnitt 3.2.4 vorgestellt.

3.2.3 Caching von WPS-Ergebnisdaten (Output-Cache)

Komplexer gestaltet sich die Konzeption für das Output-Caching. Nach den Ausführungen des zweiten Kapitels sind HTTP-Post-Anfragen nur dann zum Zwischenspeichern geeignet, wenn die Antwort eine Ablauffrist (expiration time) enthält oder eine der cache-control-Direktiven die Voreinstellungen überschreibt (Wessels, 2001). Nach Wessels (2001) sind zum Zwischenspeichern geeignete HTTP-Post-Anfragen in der Praxis äußerst selten. Der für die Implementierung benutzte Proxy-Server Squid unterstützt das Zwischenspeichern von HTTP-Post-Anfragen überhaupt nicht (Wessels, 2004, Dithardt, 2007). Die Möglichkeit Web-Caching-Techniken für HTTP-Post-Anfragen bzw. für die Antworten einzusetzen wird daher in dieser Arbeit nicht untersucht. Der WPS kann jedoch eigene Caching-Mechanismen implementieren. Wie könnte eine solche Lösung aussehen?

Zum Zwischenspeichern von WPS-Analyseergebnissen ist es notwendig die Gleichartigkeit von WPS-Anfragen festzustellen. Identische Anfragen führen zu identischen Ergebnissen. Bei Anfragen via HTTP-Get wird die URL zum Identifizieren der zwischengespeicherten Objekte herangezogen. Jedem Objekt ist eine eindeutige URL zugeordnet. Bei HTTP-Post-Anfragen kann die URL zur Identifizierung nicht herangezogen werden, da alle Anfragen an den gleichen WPS gerichtet werden und demzufolge identische URLs enthalten. Zur eindeutigen Unterscheidung der Anfragen eignet sich der Nachrichtenkörper der Anfragen. Hierzu wird der relativ lange Nachrichtenkörper der WPS-Anfrage in einen eindeutigen (kurzen) Schlüsselwert überführt. WPS-Anfragen können dann als identisch betrachtet, wenn die generierten Schlüssel der zugehörigen Nachrichtenkörper identisch sind.

Zur Umsetzung wird auf Techniken der Kryptologie zurückgegriffen. Aus dem Nachrichtenköper der HTTP-Anfrage wird eine Prüfsumme, ein sogenannter Hashwert, berechnet. Ein Verfahren zur Berechnung eines Hashwertes wird Hashfunktion

genannt. Ist H die Hashfunktion, h der Hashwert und m die zu codierende Nachricht, die auch als Urbild bezeichnet wird, so gilt nach Schmeh (2013):

$$h = H(m)$$

Eine weit verbreitete kryptographische Hashfunktion ist MD5 (Message Digest Algorithm 5). Das Verfahren liefert einen Hashwert der Länge 128-Bit (Schmeh, 2013). Mit dem Hashverfahren MD5 lässt sich der Nachrichtenkörper einer WPS-Anfrage in einen 32-stelligen, eindeutigen Wert umwandeln aus dem sich wiederum ein eindeutiger Dateiname bilden lässt. Der Datei wird das Ergebnis der Anfrage als Inhalt zugewiesen. Somit kann jeder WPS-Anfrage eine eindeutige Kennung, die zur Identifizierung eines zwischengespeicherten Objektes herangezogen wird, zugewiesen werden. Liegt für die WPS-Anfrage eine Datei im Output-Cache vor, können die Daten aus dem Cache geliefert werden. Der Ablauf kann Tabelle 3.2 sowie Abbildung 3.6 entnommen werden.

Tabelle 3.2: Anwendungsfall „WFS-Anfrage mit Input-Caching“

Bezeichnung (ID):	WFS-Anfrage mit File-Caching (Use Case 02)
Aufgabe:	Anfrage von WFS-Daten durch den WPS mit Input-Cache
Name:	WFS-Anfrage mit Input-Cache
Kurzbeschreibung:	Der Anwendungsfall beschreibt eine vom Anwender ausgelöste WPS-Anfrage. Liegt das zur Anfrage gehörende Ergebnis im internen File-Cache vor und ist gültig, so werden die Daten für die WPS-Analyse herangezogen.
Akteure:	WPS, WFS, File-Cache, Benutzer
Vorbedingung:	Der Client hat einen Execute Request mit eingebetteten WFS-Daten via HTTP-Post an den WPS gesendet. Der WPS hat den Request analysiert und an den WPS-Handler weitergeleitet. Die WFS-Daten liegen auf einem Server vor. Es besteht eine Netzverbindung zwischen WPS und WFS. Der WPS-Handler soll prüfen, ob die Daten im Cache vorliegen und gültig sind.
Nachbedingung:	Die WPS-Daten wurden entweder aus dem eigenen Cache oder durch eine Anfrage beim Ursprungsserver an den Client geliefert.

Bezeichnung (ID):	WFS-Anfrage mit File-Caching (Use Case 02)
Standardablauf:	1. Der WPS-Handler entnimmt der WPS-Konfigurationsdatei die eingestellte Caching-Methode (WPS-Input-Caching) und die Caching-Dauer. 2. Der WPS-Handler berechnet eine Prüfsumme aus dem Execute Request. 3. Der WPS-Handler generiert aus der Prüfsumme einen eindeutigen Dateinamen. 4. Der WPS-Handler prüft, ob eine Datei mit einem zur berechneten Prüfsumme passenden Namen im File-Cache vorhanden ist. 5. Falls dies zutrifft: a. Der WPS-Handler prüft die Gültigkeit der Cache-Datei über „If-Modified-Since“-Anfragen, die an alle Ursprungsserver gerichtet werden. Falls alle zugrundeliegenden WFS-Daten gültig sind: a. Die WPS-Datei im WPS-Cache wird dem Client als Ergebnis zur Verfügung gestellt. b. Der Ablauf ist beendet. ansonsten, wenn mindestens ein zugrundeliegender WFS-Datensatz nicht mehr gültig ist: a. Die WPS-Verarbeitung wird „normal“ durchgeführt. Die eingebetteten WFS-Daten werden wie im Use Case 01 beschrieben abgefragt. b. Die WPS-Funktion, z.B. eine Puffer-Berechnung wird durchgeführt. c. Der Ergebnisdatensatz wird erzeugt. d. Der Ergebnisdatensatz wird mit dem unter (1) ermittelten Namen im File-Cache abgelegt. e. Der WPS erstellt einen Protokolldatensatz. f. Der Ablauf ist beendet. ansonsten, wenn keine Datei mit einem zur unter 2. berechneten Prüfsumme im File-Cache vorhanden ist: (1) Die WPS-Verarbeitung wird „normal“ durchgeführt. Die eingebetteten WFS-Daten werden wie im Use Case 01 beschrieben abgefragt. (2) Die WPS-Funktion, z.B. eine Puffer-Berechnung wird durchgeführt. (3) Der Ergebnisdatensatz wird erzeugt. (4) Der Ergebnisdatensatz wird mit dem unter (1) ermittelten Namen im File-Cache abgelegt.

Bezeichnung (ID):	WFS-Anfrage mit File-Caching (Use Case 02)
	(5) Der WPS erstellt einen Protokolldatensatz. (6) Der Vorgang ist beendet.

Allerdings kann bis hierher noch keine Aussage zur Gültigkeit der Cache-Datei gemacht werden. Die Gültigkeit des Cache-Objekts muss vor der Auslieferung noch geprüft werden. Eine Gültigkeitsprüfung könnte wie folgt aussehen: Der WPS stellt vor der Auslieferung der Cache-Datei fest, ob die zugrundeliegenden WFS-Daten noch gültig sind. Die Cache-Datei ist dann gültig, wenn sich alle dem Berechnungsergebnis zugrundeliegenden WFS-Daten seit der letzten Aktualisierung der Cache-Datei nicht verändert haben. Die Gültigkeit einer zugrundeliegenden WFS-Datei kann nach den Ausführungen des vorangegangenen Kapitels über eine HTTP-Head-Anfrage überprüft werden. Im Gegensatz zu HTTP-Get-Anfragen werden bei HTTP-Head-Anfragen keine Daten übermittelt. Der WPS-Server sendet eine If-Modified-Since-Anfrage via HTTP-Head an den WFS-Server. Der WFS-Server vergleicht das Datum mit der originären Datei und sendet einen entsprechenden Status an den WPS zurück. Liegt der HTTP-Status 304 vor, so ist die Cache-Datei aktuell. Nur wenn alle der Analyse zugrundeliegenden WFS-Daten gültig sind, wird die Datei aus dem Output-Cache zur Verfügung gestellt.

Es sei explizit darauf hingewiesen, dass die vorgestellte Variante für das Output-Caching gegen das Prinzip der Zustandslosigkeit verstößt.

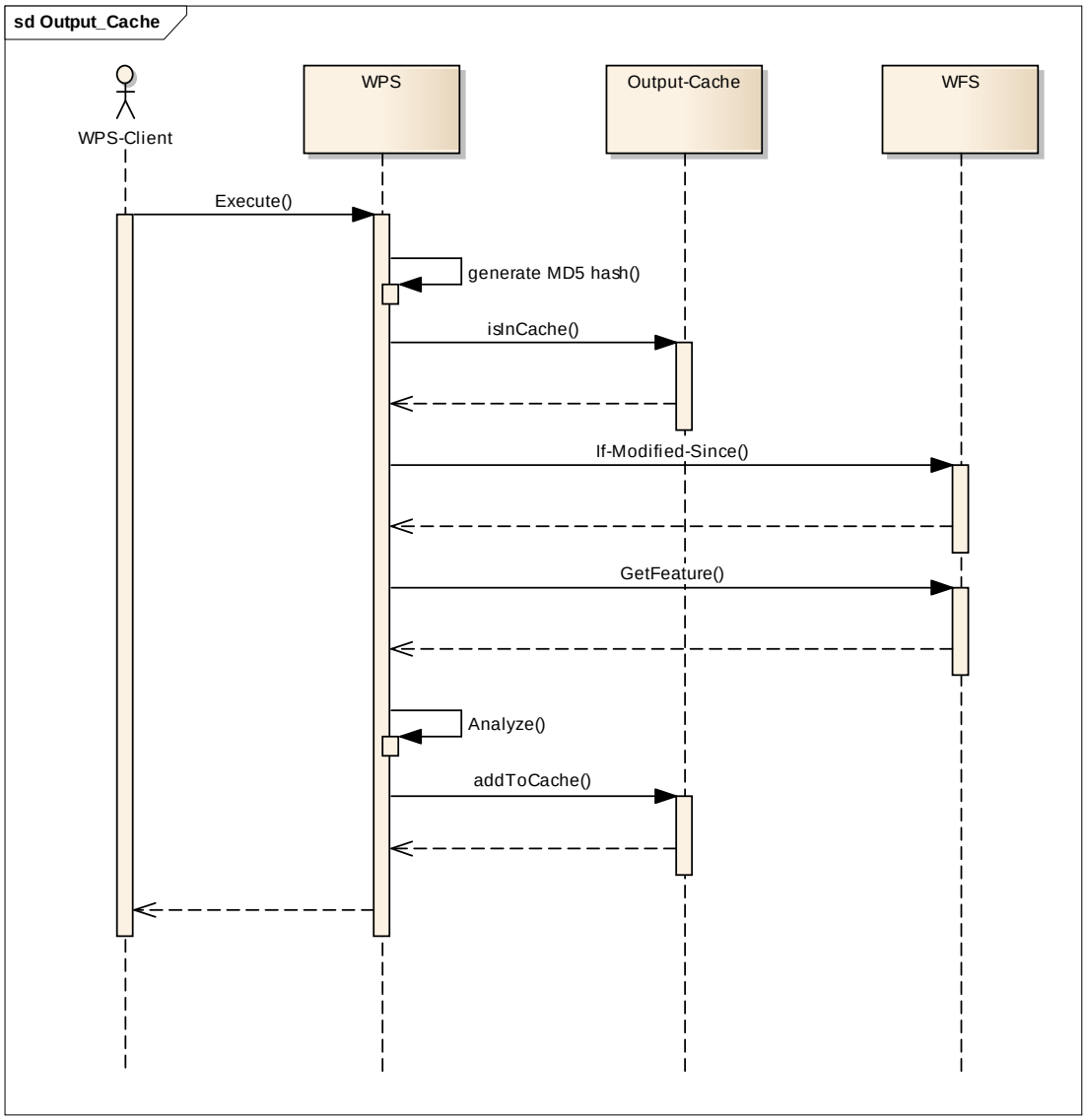


Abbildung 3.6: Ablauf einer WPS-Anfrage mit Output-Caching
1. Client stellt WPS-Anfrage; 2. WPS generiert einen MD5-hash; 3. WPS prüft, ob eine zur Anfrage passende Datei im Output-Cache vorhanden ist; 4. WPS erhält Antwort: Datei nicht vorhanden; 5. WPS stellt *If-Modified-Since*-Anfrage; 6. WFS-Server antwortet mit HTTP-Status 200 (OK); 7. WPS stellt WFS-Anfrage; 8. WFS-Server liefert Daten; 9. WPS führt Analyse durch; 10. WPS überführt Analyse-Ergebnis in Output-Cache; 11. Output-Cache bestätigt Übernahme der Daten; 12. WPS liefert Analyse-Ergebnis an Client; (eigene Darstellung)

3.2.4 Spezifikation

In den vorangegangenen Abschnitten wurden die Systemanforderungen in natürlicher Sprache und mit Sequenzdiagrammen dargestellt. Der Fokus war auf die Interaktion der Systemkomponenten und den zeitlichen Ablauf gerichtet. Für die Implementierung werden die Anforderungen verfeinert in Form von Aktivitätsdiagrammen dargestellt.

Das Aktivitätsdiagramm ist eine Diagrammart von UML (Unified Modeling Language). Es wird benutzt, um Aktivitäten mit nichttrivialen Charakter darzustellen (Rupp et. al 2007). Im Sinne von UML Version 2 spezifiziert eine Aktivität, die Menge von potenziellen Abläufen, die sich in der Realität unter bestimmten Randbedingungen abspielen (Rupp et. al 2007). Mit einem Aktivitätsdiagramm werden Aktion, Verzweigungen, Parallelbearbeitungen, Nebenläufigkeiten, Schleifen sowie Start- und Endpunkte der Abläufe dargestellt. Die Symbolik ist wie bei allen UML-Diagrammarten normiert. Aktivitätsdiagramme eignen sich zur Darstellung von Geschäftsprozessen oder System-Anwendungsfällen gleichermaßen. Die graphische Darstellung vermittelt einen Überblick über die Abläufe der Aktivität.

Die Anwendungsfälle für das Input-Caching können Abbildung 3.7 entnommen werden. Der einzige Akteur, der die Anwendungsfälle ausführt ist eine Systemkomponente. Dieser Akteur repräsentiert eine eigenständige Subkomponente des WPS.

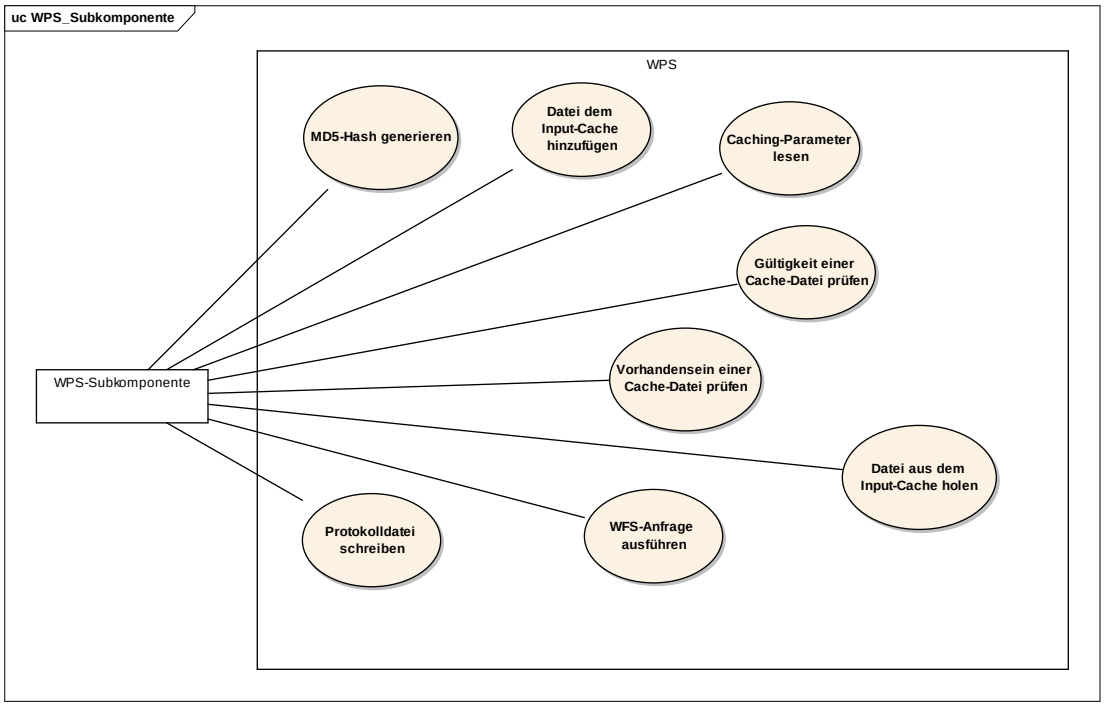


Abbildung 3.7: Systemanwendungsfälle für das WPS-Caching
(eigene Darstellung)

In den folgenden Abschnitten werden die Anwendungsfälle beschrieben und teilweise als Aktivitätsdiagramm aufbereitet. Aus Gründen der Übersicht wird nur der Standardablauf dargestellt. Auf die Darstellung von Ausnahmen wird in den Aktivitätsdiagrammen verzichtet. Für den produktiven Einsatz sollten die Abläufe so erweitert werden, dass Fehler abgefangen, der WPS-Prozess beendet und eine entsprechende Fehlermeldung ausgegeben wird.

3.2.4.1 Anwendungsfall: MD5-Hash generieren

Aus dem Nachrichtenkörper der WFS-Anfrage werden Leerräume (Leer- und Tabulatorzeichen sowie Zeilenumbrüche) entfernt. Danach wird ein auf dem Verschlüsselungsverfahren AES basierender Hash-Wert der Länge 128-Bit erzeugt. Die meisten Programmiersprachen stellen hierfür entsprechende Funktionen oder Bibliotheken zur Verfügung.

3.2.4.2 Anwendungsfall: Objekt dem Input-Cache hinzufügen

Nachdem eine WFS-Anfrage korrekt beendet wurde, wird die Antwort im Cache-Verzeichnis abgelegt. Dabei werden ggf. vorhandene Dateien mit gleichem Namen überschrieben. Der Name des Cache-Verzeichnisses wird einer Konfigurationsdatei des WPS entnommen.

3.2.4.3 Anwendungsfall: Objekt aus dem Input-Cache holen

Nachdem die Gültigkeit eines Objektes im Input-Cache festgestellt wurde, wird die Datei als Stream aus dem Cache-Verzeichnis gelesen. Das Cache-Verzeichnis wird einer Konfigurationsdatei entnommen.

3.2.4.4 Anwendungsfall: Caching-Parameter lesen

Das Caching-Verhalten (Input-Cache, Proxy-Cache oder kein Caching) und weitere für das Caching benötigte Parameter (Cache-Verzeichnis, IP-Adresse und Port des Proxy-Servers, Gültigkeitsdauer, Name der Protokolldatei) werden in einer Konfigurationsdatei festgelegt. Die Datei wird bei Bedarf ausgelesen. Beim Lesen der Daten wird eine Plausibilitätsprüfung durchgeführt. Werden sachlogisch falsche oder unsinnige Werte vorgefunden, so werden hart-kodierte Standardwerte herangezogen.

3.2.4.5 Anwendungsfall: Protokolldatensatz generieren

Nachdem der WFS auf eine Anfrage des WPS geantwortet hat, werden die Ergebnisse in eine Protokolldatei überführt. Es werden folgende Werte in einer Textdatei protokolliert: Zeitpunkt der Anfrage (Datum und Uhrzeit der Anfrage oder Zeitpunkt der Anfrage im Unix-Format), Verarbeitungszeit in Millisekunden, Größe der Antwort in Bytes, HTTP-Status der Antwort sowie URL mit Query String. Der Name der Protokolldatei wird in einer Konfigurationsdatei angegeben.

3.2.4.6 Anwendungsfall: Vorhandensein und Gültigkeit prüfen

Der Ablauf, mit dem Vorhandensein und Gültigkeit einer Datei im Input-Cache geprüft wird, sieht wie folgt aus: Nach dem Start wird zunächst das Cache-Verzeichnis ermittelt. Anschließend wird aus dem Nachrichtenkörper der WFS-Anfrage ein eindeutiger Schlüsselwert berechnet. Aus dem berechneten Schlüsselwert wird ein eindeutiger Dateiname generiert. Danach wird geprüft, ob eine Datei mit diesem Namen im Verzeichnis existiert und gültig ist. Für jeden möglichen Ausgang wird ein entsprechender Status zurückgeliefert.

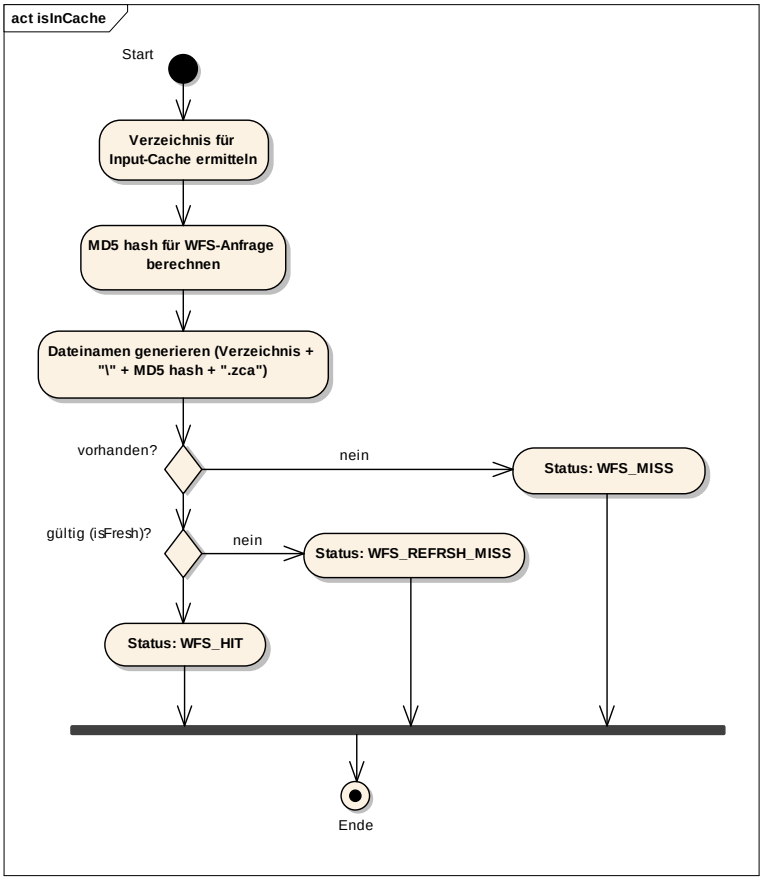


Abbildung 3.8: Vorhandensein und Gültigkeit prüfen; (eigene Darstellung)

3.2.4.7 Anwendungsfall: WFS-Anfragen ausführen

Der Ablauf einer WFS-Anfrage mit Speicherung der Antwort im Input-Cache sieht wie folgt aus: Zu Beginn der Ausführung wird die Systemzeit als Startzeitpunkt ausgelesen. Der weitere Programmablauf ist abhängig vom Wert der ausgelesenen Caching-Methode. Die Zeitmessung wird angehalten, wenn die Daten aus der WFS-Anfrage oder aus dem Cache zur weiteren Verarbeitung zur Verfügung stehen. Anschließend wird die WFS-Anfrage protokolliert.

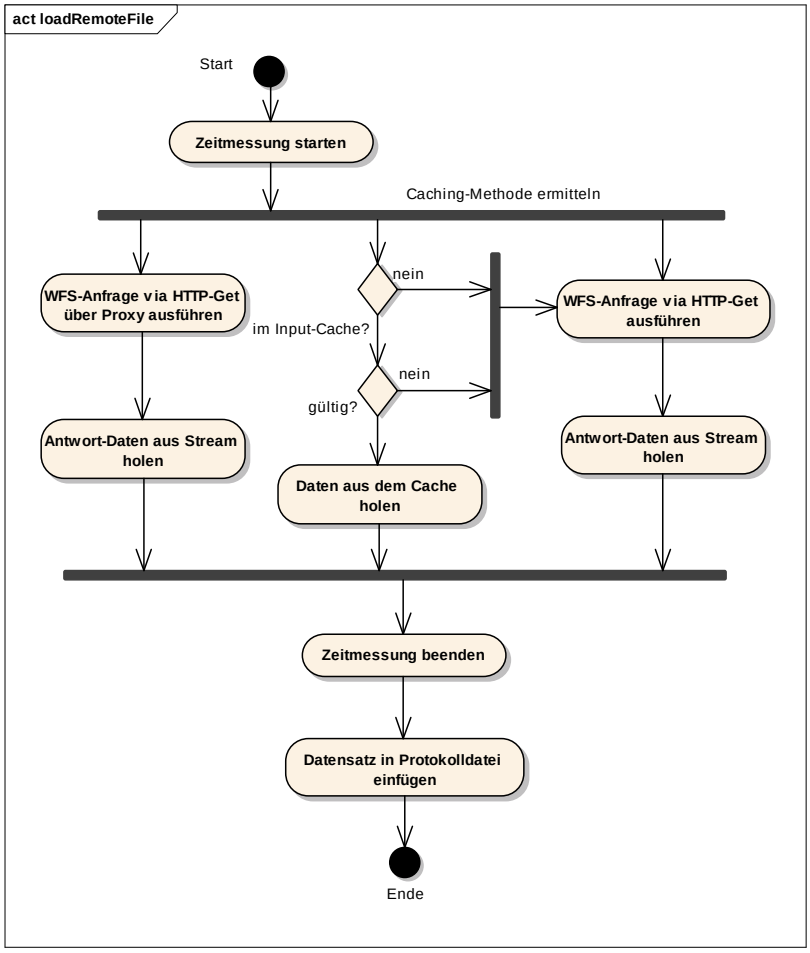


Abbildung 3.9: Durchführen der WFS-Anfragen; (eigene Darstellung)

Die Implementierung der vorgestellten Konzepte und Spezifikationen wird im nächsten Kapitel vorgestellt.

3.3 Test der Software

Die Qualität von Software ist nach Spillner und Linz (2012) ein entscheidender Faktor für den Erfolg von Produkten oder Unternehmen geworden. Dies gilt für technische und kommerzielle Softwaresysteme gleichermaßen. Hansen und Neumann (2009) definieren einen Softwaretest als einen Prozess, bei dem geprüft wird, ob ein bestimmtes Softwaresystem den zugrundeliegenden Spezifikationen entspricht und ob es in der dafür vorgesehenen Systemumgebung lauffähig ist. Sie führen weiter aus, dass Testen ein wichtiger Bestandteil der Qualitätssicherung ist. Hansen und Neumann (2009) unterscheiden zwischen drei verschiedene Testarten: Modultest, Integrationstest und Systemtest. Spillner und Linz (2012) erweitern die Aufzählung um folgende Testarten: Abnahmetest und Test nach Änderung.

Die oben aufgeführten Testarten geben die zeitliche Entwicklung eines Softwareprojektes wieder. Die erste quasi niedrigste Stufe ist der Komponenten- oder Modultest. Im Rahmen dieses Tests wird eine einzelne Softwarekomponente isoliert und erstmalig überprüft. Was unter einem Modul oder eine Komponente zu verstehen ist, hängt von der eingesetzten Programmiersprache ab. Bei den in der Programmiersprache C durchgeführten Erweiterungsentwicklungen des Zoo-Kernels, kann dies eine entworfene Datenstruktur oder einzelne Funktion sein. Bei der in C# entwickelten Benchmark-Software kann dies abhängig von Struktur und Größe eine gesamte Klasse oder eine einzelne Klassenfunktion sein.

Der Komponententest findet gewissermaßen auf der Festplatte mit den gerade entwickelten Objekten statt (Spillner und Linz, 2012). Mit den Tests, die oftmals von den Entwicklern selbst durchgeführt werden, soll sichergestellt werden, dass die Testobjekte korrekt und vollständig laut Spezifikation realisiert wurden (Spillner und Linz, 2012). Die Modultests werden iterativ in der Implementierungsphase durchgeführt.

Nachdem jede einzelne Komponente durch den Modultest erfolgreich getestet wurde, werden im Rahmen des Integrationstests Subsysteme und Konfigurationen gemeinsam getestet (Hansen und Neumann, 2009). Mit diesem Test wird das Zusammenwirken mehrerer Komponenten überprüft. Ziel des Integrationstests ist es Fehlerzustände in Schnittstellen oder im Zusammenspiel der Komponenten aufzudecken (Spillner und Linz, 2012). Bei der durchgeführten Erweiterung des Zoo-Kernels muss das Zusammenwirken mit dem Proxy-Server und den WFS-Servern sowie dem Input- und Output-Cache getestet werden. Bei der entwickelten Benchmark-Software

steht ebenfalls das Zusammenspiel zwischen der Software und den Web-Servern oder dem Proxy-Server im Vordergrund. Als Testbasis können Anwendungsfälle oder schnittstellenübergreifende Prozesse herangezogen werden (Spillner und Linz, 2012). Bei der Anpassung des Zoo-Kernels muss dem Integrationstest eine besondere Beachtung zukommen, da ein Fehlerzustand in einer Schnittstelle Einfluss auf das Analyse-Ergebnis hat. Der Integrationstest wird im Anschluss an den Modultest in der Testumgebung durchgeführt.

Nach Abschluss des Integrationstests wird mit dem Systemtest das komplette Softwaresystem, prinzipiell das fertig installierte Informationssystem, getestet (Hansen und Neumann, 2009). Nach Spillner und Linz (2012) wird das System als Ganzes in einer Testumgebung betrachtet. Die Testumgebung soll der späteren Produktionsumgebung nahe kommen. Im Rahmen des Systemtests wird validiert, ob und wie gut die funktionalen und nicht funktionalen Systemanforderungen erfüllt wurden. Der Systemtest wurde nach Abschluss des Integrationstests in der Testumgebung durchgeführt.

3.4 Performance- und Lasttests

Performance- und Lasttests gehören zu den nicht funktionalen Tests mit denen die Qualität einer Software bestimmt wird. Bei den nicht funktionalen Tests steht die Funktionsweise einer Software im Vordergrund. Molyneaux (2009) unterscheidet zwischen serviceorientierten und effizienzorientierten Indikatoren. Zu den serviceorientierten Indikatoren gehören die Verfügbarkeit und die Antwortzeit eines Systems. Zu den effizienzorientierten Indikatoren werden der Datendurchsatz und die Kapazität einer Software gezählt.

Nach Molyneaux (2009) wird den nicht funktionalen Tests nicht die Bedeutung beigemessen, die ihr über den gesamten Lebenszyklus zugemessen werden sollte. Er führt weiter an, dass die Einsicht zur Durchführung dieser Tests in den Unternehmen noch immer nicht vorhanden ist. Ganz im Gegensatz zu den funktionalen Tests, die in Unternehmen mittlerweile einen hohen Standard erreicht haben.

Mit Performance-Tests wird die Verarbeitungsgeschwindigkeit oder die Antwortzeit für bestimmte Prozesse oder Dienste gemessen (Spillner und Linz, 2012). Mit Lasttests wird das Systemverhalten in Abhängigkeit steigender Systemlast gemessen (Spillner und Linz, 2012). Eine zunehmende Anzahl parallel arbeitender Anwender oder eine zunehmende Anzahl von Transaktionen führen zu einer steigenden Systemlast.

Nach Molyneaux (2009) sollte die Anzahl der zu erwartenden Anwender zum Zeitpunkt der Systemeinführung sowie sechs Monate und zwei Jahre danach festgelegt werden. Molyneaux führt weiter aus, dass Tests auf der Grundlage einer stabilen Anwendung, mit qualitativ hochwertigen Daten und automatisiert durchgeführt werden sollten.

3.5 Anforderungen an Messwerkzeuge zur Durchführung von Performance- und Lasttests

Im Rahmen der Machbarkeitsuntersuchung wird das Antwortzeitverhalten am Client für unterschiedliche Anwendungsfälle gemessen werden. Für Stress- oder Lasttests wird ein Werkzeug benötigt, mit dem Messungen für mehrere parallele Anfragen automatisiert ausgeführt werden können. Die Gesamtanzahl der Abfragen und die Anzahl der konkurrierenden Abfragen sollen konfigurierbar sein. Die Messungsergebnisse der Einzelabfragen und die gesamte Verarbeitungszeit sollen protokolliert werden. Die Messergebnisse sollen statistisch ausgewertet werden. Im Einzelnen sollen aus den Messergebnissen folgende Werte berechnet werden:

- Minimalwert einer Einzelmessung
- Maximalwert einer Einzelmessung
- Median der Einzelmessungen
- arithmetischer Mittelwert der Einzelmessungen
- Standardabweichung der Einzelmessungen

Der Ablauf der Performance- und Lasttests kann der folgenden Beschreibung und Abbildung 3.10 entnommen werden:

Nach dem Start werden u.a. die Werte der folgenden Parameter eingelesen:

- Anzahl der durchzuführenden Abfragen
- Anzahl der konkurrierenden Abfragen

Jede WPS-Anfrage wird in einem eigenen Thread durchgeführt. Die Anzahl der vom System parallel zur Verfügung gestellten Threads entspricht dem Wert für die Anzahl der konkurrierenden Anfragen. Nachdem die Threads vom System freigegeben wurden, wird die Gesamtzeitmessung gestartet.
Im weiteren Ablauf werden die Anfragen auf die freien Threads verteilt. Stehen nicht genügend freie Threads zur Verfügung, so muss gewartet werden bis eine Anfrage beendet wurde.

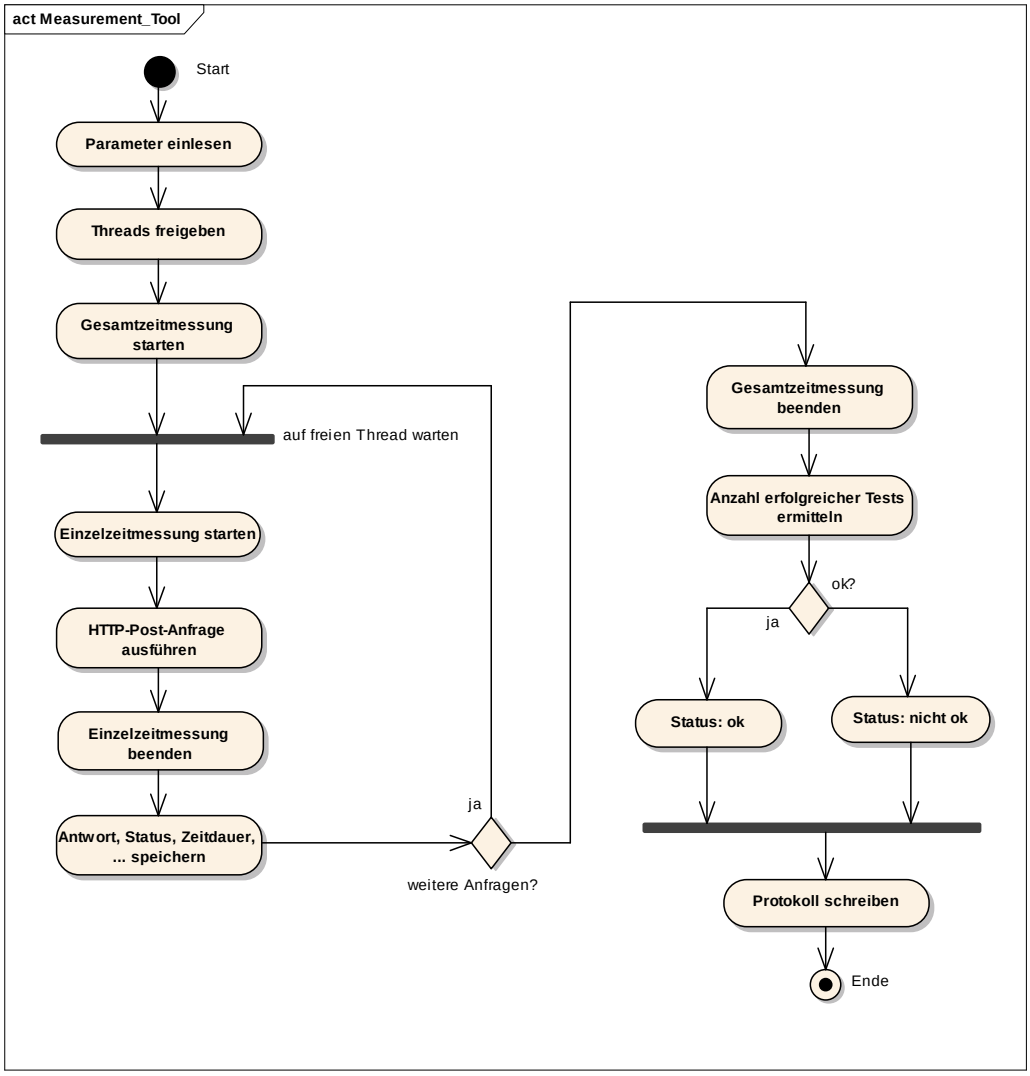


Abbildung 3.10: Ablauf der Performance- und Lasttests; (eigene Darstellung)

Steht ein freier Thread zur Verfügung, so wird zunächst die Zeitmessung für diese Einzelanfrage gestartet. Danach erfolgt die Verarbeitung der WPS-Anfrage. Die Zeitmessung für eine Einzelabfrage wird beendet, wenn das Analyseergebnis komplett zum Client übertragen wurde und dort zur weiteren Verarbeitung zur Verfügung steht. Eine ggf. notwendige clientseitig Aufbereitung der Daten wird zeitlich nicht erfasst. Danach wird das Ergebnis der Zeitmessung protokolliert.

Wurden alle Anfragen beendet, wird die Gesamtzeitmessung beendet. Der weitere Ablauf ist abhängig vom Anteil der erfolgreich durchgeführten Einzeltests an der Gesamtzahl der angeforderten Tests. Wurden mindestens 90% der Einzeltests erfolgreich durchgeführt, so werden die Messungen statistisch ausgewertet und es wird eine Erfolgsmeldung zurückgegeben. Andernfalls wird eine Fehlermeldung zurückgeliefert. Auf die Messergebnisse und die statistischen Daten kann über entsprechende Funktionen zugegriffen werden.

Die softwaretechnische Umsetzung wird im nächsten Kapitel vorgestellt. Ein Konzept sowie die Durchführung von Performance- und Lasttests werden im fünften Kapitel präsentiert.

3.6 Messwerkzeuge

Die Apache Software Foundation bietet zwei Werkezuge zur Durchführung von Lasttests an, Apache Benchmark und Apache JMeter.

Apache Benchmark ist eine Kommandozeilen-Anwendung und Teil des Apache HTTP-Webserver-Projektes. Nach Loechel und Schmid (2012) ist Apache Benchmark ein De-facto-Standard zur Durchführung von HTTP-Benchmark-Tests an Webservern. Das Werkzeug analysiert HTTP-Header und stellt statistische Daten der Messungen wie die mittlere Antwortzeit, den mittleren Fehler einer Einzelmessung, den mittlerer Fehler aller Messungen und die Übertragungsrate der Messungen zur Verfügung.

Apache JMeter ist eine Java-Anwendung mit einer graphischen Benutzeroberfläche, die im Rahmen des Apache Jakarta Projektes entwickelt wurde. Mit der Benutzeroberfläche können Testpläne spezifiziert und die zu durchlaufenden Komponenten festgelegt werden. Die Ausführung kann wahlweise über die Kommandozeile oder

über die Benutzeroberfläche erfolgen. Es können Tests für statische oder dynamische Komponenten (Beispiele: Servlets, Scripte, Datenbankabfragen) durchgeführt werden.

4 Implementierung

Das Kapitel beschreibt die Implementierung der im vorangegangenen Kapitel vorgestellten Anforderungen und Spezifikationen. Es werden Auswahl und Funktionsweisen der verwendeten Systemkomponenten, die softwaretechnischen Anpassungen sowie die Referenzarchitektur vorgestellt. Des Weiteren wird die Entwicklung eines objektorientierten Werkzeugs zur Durchführung von Performance- und Lasttests aufgezeigt.

4.1 Verwendete Software-Komponenten

Für die Implementierung wird freie Software herangezogen. Freie Software zeichnet sich dadurch aus, dass das Programm zu jedem Zweck ausgeführt, untersucht und verändert, verbreitet sowie verbessert werden darf. Verbesserungen sollen verbreitet werden, um damit einen Nutzen für die Gemeinschaft zu erzeugen. Der Begriff Freie Software und dessen Definition entstanden zu Beginn des GNU-Projektes, einem Vorhaben zur Erstellung eines *freien* Betriebssystems (URL 2013).

Die genaue Kenntnis des Quellcodes ist gerade bei GIS von entscheidender Bedeutung, da die zugrundeliegenden Algorithmen komplex sein können und daher einen großen Einfluss auf die Ergebnisse räumlicher Analysen und Modellierung haben können (Neteler und Mitasova 2008). Freie Software fördert die kontinuierliche Erneuerung und Verbesserung von GIS-Software. Neben der weit verbreiteten proprie-

tären Software, spielt freie Software bei der Anpassung von GIS-Software eine wichtige Rolle, da sie experimentelle Ansätze fördert (Neteler und Mitasova 2008). Die Auswahl der Software orientierte sich an diesen Grundgedanken sowie an den Kenntnissen des Autors. Die Verwendung alternativer Software-Produkte ist denkbar.

4.1.1 Zoo-WPS

Zoo-WPS ist eine Implementierung der OGC WPS-Spezifikation 1.0.0. Zur Umsetzung wurde im Jahr 2008 das Open-Source-Projekt Zoo gegründet, das unter der Lizenz MIT/X-11 veröffentlicht wurde (Zoo-URL-1 2013). Zoo-WPS stellt ein leistungsstarkes Framework zur Verfügung und besteht im Wesentlichen aus folgenden drei Komponenten (Zoo-URL-1 2013):

- Zoo-Kernel: Der in der Programmiersprache C entwickelte Kernel führt Zoo-Services aus und stellt Funktionen zum Verketten von Zoo-Services zur Verfügung. Der Kernel unterstützt mehrere Programmiersprachen. Als Web-Server wird Apache eingesetzt. Es können alle Vektor- und Rasterformate verarbeitet werden, die von der Bibliothek GDAL/OGR unterstützt werden.
- Zoo-Services: Eine Sammlung beispielhafter Web-Dienste, die auf verschiedenen offenen Bibliotheken basieren. Die Dienste bestehen aus Algorithmen und Funktionen und können in verschiedenen Programmiersprachen implementiert werden, z.B. C, C++, Fortran, Java, Python, PHP und JavaScript.
- Zoo-API: Eine JavaScript Bibliothek zur Erstellung und Verkettung von Zoo-Services.

Als Webserver wird Apache HTTP-Server benötigt.

4.1.2 HTTP-Proxy Squid

Squid ist ein weitverbreiteter Caching- und Proxy- Server, der die Protokolle HTTP/HTTPS, FTP über HTTP und Gopher unterstützt (Squid URL-1 2013). Er zeichnet sich durch seine gute Skalierbarkeit aus und steht als freie Software unter der GNU General Public Licence (Squid URL-1).

Nach Loechel und Schmid (2012) stellt Squid einen De-facto-Standard dar. Squid wird von mehreren Unternehmen und Projekten zum Zwischenspeichern von Inhalten eingesetzt. Wikipedia ist ein bekanntes Großprojekt, das Squid zum Zwischenspeichern der eigenen Server nutzt (Loechel und Schmid 2012). Das Wikimedia-Projekt hatte Ende 2010 weltweit 155 Squid-Server im Einsatz, die ca. 75% der Anfragen beantworten (Wikipedia URL 2013).

Squid steht für viele Betriebssysteme darunter auch Microsoft Windows zur Verfügung. Im Rahmen dieser Arbeit wird die stabile Windows-Version 2.7 von Squid eingesetzt (Squid URL-2 2013).

4.2 Anpassung des Zoo-Kernels

Da Zoo-WPS keine Möglichkeit bietet den HTTP-Verkehr mittels Konfiguration umzuleiten, wird der Kernel softwaretechnisch angepasst. Als Entwicklungsumgebung wird Microsoft Visual C++ 2010 Express (URL 2013) eingesetzt. Folgende Bibliotheken und Tools müssen in die Entwicklungsumgebung integriert werden (URL-2 2013):

cgic: Die Bibliothek stellt umfangreiche Funktionen für die Entwicklung von cgi-Programmen zur Verfügung (URL 2013)

cURL: Die Bibliothek implementiert zahlreiche Funktionen zum Übertragen von Daten über das Internet via Get oder Put. Es werden die Protokollarten HTTP, HTTPS, FTP, FTPS, DICT, LDAP, RTMP und Gopher unterstützt (URL 2013).

FastCGI: FastCGI wurde entwickelt um die Performance-Probleme des Common Gateway Interface (CGI) zu umgehen. Bei der Ausführung eines CGI-Programms wird für jede Anfrage ein eigener Interpreter geladen. Im Gegensatz dazu stellt ein FastCGI-Programm eine zentrale Schleife auf dem Web-Server zur Verfügung, die mehrere Prozesse quasi gleichzeitig bedienen kann (URL 2013).

Libxml2: Die Bibliothek implementiert Funktionen zum Parsen von xml-Dokumenten (URL 2013).

OpenSSL: Die Bibliothek stellt Funktionen für die Protokolle Transport Layer Security (TLS) und Secure Sockets Layer (SSL) zur Verfügung (URL 2013).

Python: Die Bibliothek ermöglicht es Python-Programme aus einem C-Programm heraus zu starten. Sie wird für Programmteile verwendet, für die Python eine bessere Flexibilität bietet. (URL 2013)

Bison und Flex: Bison und Flex sind Generatoren zum Parsen, die für die Suche von Mustern in Eingabedaten eingesetzt werden (URL 2013 und URL 2013).

Die Einrichtung der Entwicklungsumgebung ist zeitaufwendig und komplex. Die eingesetzte Windows-Version des Zoo-Kernels steht zu Beginn der Arbeit nicht lauffähig zur Verfügung und muss zunächst gemeinsam mit den Entwicklern des Zoo-Projektes in einen funktionierenden und ausreichend stabilen Zustand überführt werden. Auf Grund dieser Konstellation und der komplexen Architektur des Zoo-Kernels werden bei der Implementierung Schwerpunkte gesetzt.

Tabelle 4.1: Übersicht der am Zoo-Kernel durchgeführten Erweiterungen

Funktionsname	Beschreibung
isInCache	prüft, ob eine zur WFS-Anfrage passende Datei im Input-Cache vorhanden ist
addToCache	fügt dem Input-Cache WFS-Daten hinzu
loadRemoteFile	bereitet WFS-Anfragen vor und ruft diese auf
writeAccessLog	fügt der Protokolldatei einen Eintrag hinzu
isFresh	prüft an Hand der Caching-Parameter, ob eine zwischengespeicherte Datei gültig (fresh) ist
getCachingParam	lädt die Caching-Parameter aus der Konfigurationsdatei und führt eine Plausibilitätsprüfung durch
InternetOpenUrlHttpGet	holt Daten via HTTP-Get von (entfernten) WFS-Servern; durch Setzen eines Parameters kann der HTTP-Verkehr über einen Proxy-Cache umgeleitet werden
getMd5	berechnet einen MD5-Hash

Bei der durchgeführten Anpassung des Zoo-Kernels handelt es sich nach Hansen und Neumann (2009) um eine Erweiterungsprogrammierung. Die funktionalen Erweiterungen waren nicht vorgesehen. Die Anwendung bietet auch keine Möglichkeit zum Aktivieren dieser Funktionen. Die neu entwickelten oder angepassten Funktionen mit ihren Beschreibungen können Tabelle 4.1 entnommen werden.

4.3 Konfiguration von Squid

Wie im zweiten Kapitel ausgeführt werden Anfragen, die URLs mit einem Query String enthalten, standardmäßig nicht zwischengespeichert. Mit der Konfigurations-Direktive *resfreh_pattern* stellt Squid eine Möglichkeit zur indirekten Kontrolle des Caches zur Verfügung. Mit der Direktive können URLs mit Query String gefiltert werden. Squid kann auf Grund von Parametern entscheiden, ob das zu einer Anfrage gehörende Objekt aus dem Cache geliefert werden kann (Dithardt 2007 und Wessels 2004). Die Direktive kann in der Konfigurationsdatei *Squid.conf* angegeben werden. Der Direktive sind vier obligatorische Parameter zu übergeben.

```
refresh_pattern [i] regex min percent max [options]
```

Der Parameter *regex* bestimmt einen regulären Ausdruck für einen URL-Filter. Der Parameter *min* gibt ein Zeitfenster, in dem die Objekte auch ohne ausdrückliches Verfallsdatum als gültig angenommen werden. Der Parameter *max* definiert den maximalen Zeitraum, in dem Objekte als gültig betrachtet werden. Mit dem Parameter *percent* kann ein Prozentsatz des Objektalters definiert werden. Objekte, deren Alter innerhalb des Prozentsatzes liegen, werden als aktuell betrachtet. Über den optionalen Parameter *i* kann die Unterscheidung von Groß-/Kleinschreibung vorgesehen werden.

4.4 Werkzeug zum Messen von Antwortzeiten

Die im vorangegangenen Kapitel beschriebenen Anforderungen und Spezifikationen werden objektorientiert modelliert. Unter einer Klasse im Sinne von UML versteht man eine Menge von Exemplaren, die über gemeinsame Eigenschaften, Einschränkungen und Semantik verfügen (Rupp et al 2007). Zur Darstellung stellt UML das Klassendiagramm zur Verfügung. Mit dem Klassendiagramm werden Verhalten und Daten eines Systems in strukturierter Form abgebildet. Es werden Klassen mit ihren

Attributen und Operationen sowie die zwischen Klassen bestehenden Assoziationen dargestellt.

Das Klassendiagramm wird als Vorstufe für die softwaretechnische Umsetzung entworfen. Als Programmiersprache wird C# auf der Grundlage von Microsoft .NET Framework 4.0 eingesetzt. Die freie Entwicklungsumgebung SharpDevelop wird zur softwaretechnischen Umsetzung eingesetzt (URL 2013). Die Anwendung wird als Konsolenanwendung realisiert. Das UML-Klassendiagramm kann Abbildung 4.1 entnommen werden.

Die Klasse *_Tools* stellt drei öffentliche Funktionen zum Schreiben und Lesen von Streams zur Verfügung.

Die Klasse *_Statistics* stellt öffentliche Funktionen zur Berechnung der Standard-Abweichung und des Medians (aus den Ergebnissen der Einzelmessungen) zur Verfügung.

Die Klasse *_WebRequest* implementiert Funktionen zum Ausführen von HTTP-Anfragen via Get und Post. Die Antwort einer Anfrage kann über die öffentliche Funktion *GetResponse* abgefragt werden. Der HTTP-Status wird in der öffentlichen Variable *Status* zur Verfügung gestellt.

Die Klasse *_WebServiceBenchmark* implementiert Funktionen zur Durchführung von HTTP-Benchmark-Tests sowie zum Speichern und Anfragen der Messergebnisse. Die HTTP-Anfragen werden über die Funktion *PerformBenchmarkTest* gesteuert, die weiter unten erläutert wird. Auf die Messergebnisse kann über öffentliche Funktionen zugegriffen werden. Die Funktion *GetStatus* liefert beispielsweise den Gesamtstatus der Messungen; die Funktion *GetMaxTimePerRequest* liefert beispielsweise die längste gemessene Verarbeitungszeit einer Einzelmessung.

Abbildung 4.1: Klassenmodell WebServiceBenchmark; (eigene Darstellung)

Die Operation, die Ablauf und Aufruf der HTTP-Anfragen steuert, soll näher betrachtet werden. Zunächst wird der Programmcode vorgestellt, danach folgt eine Beschreibung des Ablaufs.

Listing der Funktion *PerformBenchmarkTest* der Klasse *_WebServiceBenchmark*:

```
// steuert Ablauf und Aufruf der HTTP-Requests
public void PerformBenchmarkTest()
{
  Thread[] requestThreads;
  if (totalRequests < 1 || concurrentRequests > totalRequests ||
      concurrentRequests < 1)
  {
    benchmarkStatus = -1;
    return;
  }

  requestThreads = new Thread[totalRequests];
  totalStartTime = DateTime.Now;

  for (int j = 0; j < totalRequests; j++)
  {
// Anzahl der parallel ausgeführten Threads begrenzen
    while (GetTotalRunningRequests() >= concurrentRequests)
        ;

    int temp = j;
    requestThreads[j] = new Thread(PerformBenchmark);
    requestThreads[j].Start(temp);
    status[j] = 1;   // running;
  }

// warten bis alle Requests verarbeitet wurden
  while (GetTotalCompletedRequests() != totalRequests)
     ;

  if (GetTotalSuccessfulRequests() > (long) (totalRequests *
         minimumSuccesRate))
    benchmarkStatus = 1;
  else
    benchmarkStatus = -1;
}
```

Die Funktion *PerformBenchmarkTest* steuert Ablauf und Aufruf der angeforderten HTTP-Anfragen. Zur Steuerung werden die folgenden Variablen, die beim Aufruf des Klassen-Konstruktors übergeben werden, herangezogen.

```
url:                URL für die Web-Anfrage
requestData:        Nachrichtenkörper einer HTTP-Post-Anfrage
totalRequests:      Anzahl der durchzuführenden Anfragen
concurrentRequests: Anzahl der Anfragen, die parallel verar-
                    beitet werden können
```

Zu Beginn wird geprüft, ob die Werte der Variablen *totalRequest* und *concurrentRequests* sachlogisch richtig sind. Ist dies nicht der Fall, wird der Variablen *benchmarkStatus* ein entsprechender Status zugewiesen und die Funktion wird beendet. Das rufende Programm kann den Status über die Funktion *GetStatus* ermitteln.

Wurden keine Fehler festgestellt, wird der Startzeitpunkt der Variablen *totalStartTime* zugewiesen und ein Array mit Threads angelegt. In der anschließenden *For-Schleife* wird zunächst die Anzahl der parallel laufenden Threads geprüft. Erst wenn die Anzahl der laufenden Threads kleiner als der Wert der Variable *concurrentRequests* ist, wird eine neue Anfrage ausgeführt. Beim Anlegen des Threads wird automatisch die Funktion *PerformBenchmark* ausgeführt. Die *For-Schleife* wird verlassen wenn alle Anfragen verarbeitet wurden.

Nachdem alle Threads verarbeitet wurden, wird der Endzeitpunkt der Variablen *totalEndTime* und die verbrauchte Zeitspanne in Millisekunden der Variablen *totalElapsedTime* zugewiesen. Zum Schluss wird geprüft, wie viele Verarbeitungen erfolgreich beendet wurden. Bei mehr als 90% korrekt durchgeführter Anfragen, wird das Gesamtergebnis als erfolgreich übernommen. Der Status der gesamten Verarbeitung wird der Variablen *benchmarkStatus* zugewiesen.

Der Programmcode steht auf dem beigefügten Datenträger zur Verfügung.

4.5 Infrastruktur

Zur Durchführung der Integrations- und Systemtest sowie für die weiteren Untersuchungen wird eine Referenzarchitektur aufgebaut. Als Hardware stehen zwei Notebooks zur Verfügung. Das erste Notebook wird als Server eingesetzt und ist mit folgenden Leistungsmerkmalen ausgestattet:

Betriebssystem: Windows 7 Professional 64 Bit, Service Pack 1
CPU: Intel® Core™ i7-2620M CPU, 2.70 GHz, 4 Kerne
Arbeitsspeicher: 16 GByte

Auf diesem Notebook werden zwei virtuelle Maschinen eingerichtet. Als Virtualisierungssoftware wird VMWare Workstation Version 8.0.4 eingesetzt.

Auf der ersten virtuellen Maschine wird ein lokaler WFS betrieben. Die VM wird wie folgt konfiguriert:

interne IP-Adresse:	192.168.2.121	
Betriebssystem:	Windows 7 Professional 32 Bit, Service Pack 1	
CPU:	Intel® Core™ i7-2620M CPU, 2.70 GHz, 2 Kerne	
Arbeitsspeicher:	6 GByte	
Software:	Datenbank:	Oracle XE 11 R2
	WFS:	GeoServer 2.2.3
	Webserver:	Apache Tomcat 6.0.35

Auf der zweiten virtuellen Maschine werden ein WPS und ein Proxy betrieben:

interne IP-Adresse:	192.168.2.122	
Betriebssystem:	Windows 7 Professional 32 Bit, Service Pack 1	
CPU:	Intel® Core™ i7-2620M CPU, 2.70 GHz, 2 Kerne	
Arbeitsspeicher:	8 GByte	
Software:	WPS:	Zoo-WPS 1.0
	Webserver:	Apache 2.2.14
	Proxy-Server:	Squid 2.7

Die virtuellen Maschinen werden über eine Netzwerkbrücke verbunden.

Das zweite Notebook wird als Client eingesetzt:

interne IP-Adresse:	192.168.2.128	
Betriebssystem:	Windows 7 Professional 64 Bit, Service Pack 1	
CPU:	Intel® Core™ i3-2310M CPU, 2.10 GHz, 2 Kerne	
Arbeitsspeicher:	4 GByte	
Software:	Messwerkzeuge:	Apache Benchmark, WebServiceBenchmark
	Framework:	.NET-Framework 4.0

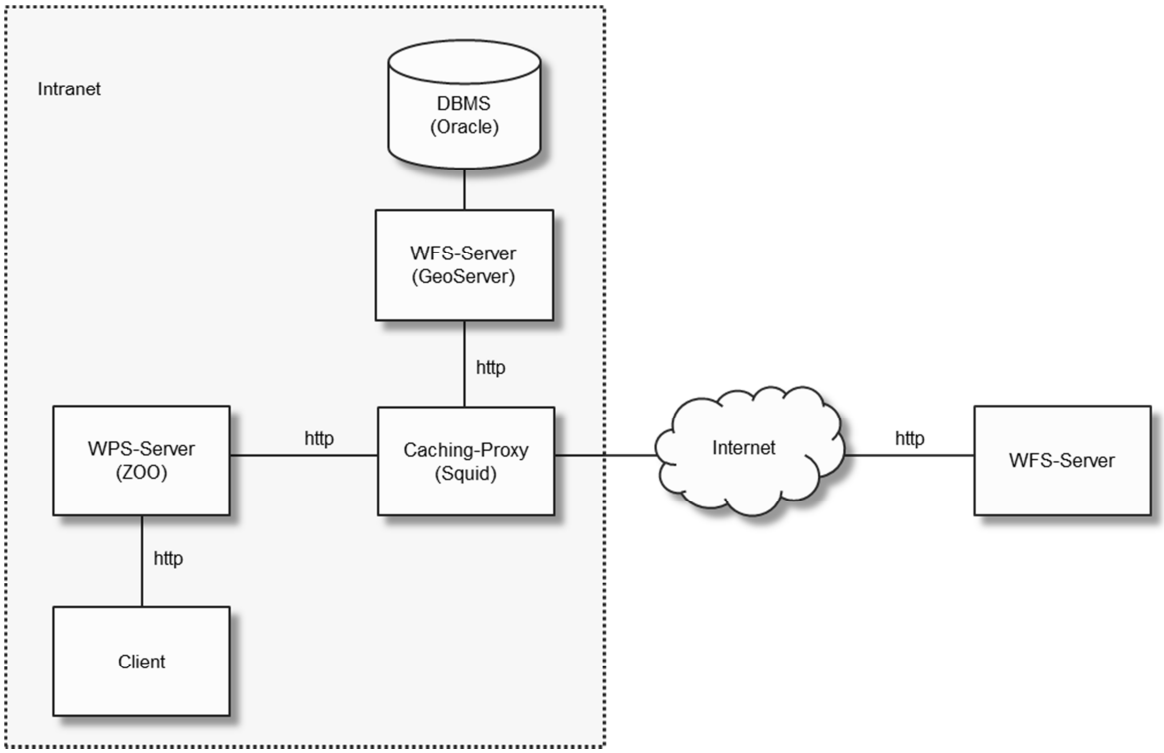


Abbildung 4.2: Referenzarchitektur; (eigene Darstellung)

Für die Durchführung von Integrations- und Systemtest wird auf einen WFS-Server im lokalen Netzwerk und auf einen entfernten WFS-Server zurückgegriffen. Im lokalen Netzwerk werden ein Polygondatensatz mit ca. 8000 Objekten sowie ein Polygondatensatz mit ca. 800 Objekten unter Oracle Express Edition 11g Release 2 (Oracle,

2013) installiert und über GeoServer Version 2.2.3 (URL 2013) als WFS zur Verfügung gestellt. Als Web-Server wird Apache Tomcat Version 6.0.35 (URL 2013) benutzt. Beiden Datensätzen wird der Namensraum *mt* zugewiesen.

Darüber hinaus werden Tests mit folgenden Daten aus dem Internet durchgeführt:

- http://demo.opengeo.org/geoserver/ows
- http://map1.naturschutz.rlp.de/service_lanis/mod_wfs/wfs_getmap.php?mapfile=naturschutzgebiet

Die Architektur der Referenzumgebung ist in Abbildung 6.1 dargestellt. Das Konzept und die Durchführung der Performance- und Lasttests werden im nächsten Kapitel vorgestellt.

5 Ergebnisse

Im ersten Teil des Kapitels wird der angepasste Zoo-WPS vorgestellt. Im zweiten Teil des Kapitels wird der Einfluss der durchgeführten Anpassungen auf die Performance untersucht. Hierzu wird ein Konzept entwickelt, mit dem die Einflüsse automatisiert, mit steigender Last und zunehmender Komplexität gemessen werden können. Am dritten Teil werden die Ergebnisse des Tests vorgestellt.

5.1 Zoo-Kernel

Nach den durchgeführten softwaretechnischen Anpassungen und Erweiterungen lässt sich das Caching-Verhalten (WPS-Input-Caching, Proxy-Caching, kein Caching) des Zoo-Kernels über Einträge in der Konfigurationsdatei *main.cfg* steuern. Die Parameter des neuen Abschnitts Caching und deren Aufgabe können der folgenden Aufstellung entnommen werden.

```
Method:          Caching-Methode [WPS_IN | PROXY | NONE]
                   WPS_IN:   Cachen der WPS-Eingangsdaten
                   PROXY:    Proxy-Caching
                   NONE:     kein Caching
Proxy:           IP-Adresse und Port des Proxy-Servers
                 [IP-Adresse:Port]
maxCachingTime:  Zeitraum in dem eine Datei als gültig
                 betrachtet wird [Minuten]
LogFile:         Verzeichnis der Protokoll-Datei
```

```
Cache:          Cache-Verzeichnis
```

Das folgende Beispiel zeigt die Einstellungen für das Proxy-Caching (Method = PROXY). Der Caching-Proxy hat die IP-Adresse 127.0.0.1 und erwartet Anfragen auf Port 3128. Der Parameter *maxCachingTime* wird beim Proxy-Caching ignoriert, da die Gültigkeitsdauer im Proxy eingestellt wird. Beträgt die im Proxy eingestellte Gültigkeitsdauer beispielsweise 1440 Minuten (24 Stunden), so verlieren Caching-Objekte nach dieser Zeitspanne ihre Gültigkeit. Wird der gleiche Datensatz innerhalb dieser Zeitspanne erneut angefragt, so stellt Squid die Daten aus dem Cache zur Verfügung, sofern sie im Cache verfügbar sind. Werden die Daten nach Ablauf von 24 Stunden erneut angefragt, leitet Squid die Anfrage an den Ursprungsserver weiter und legt anschließend eine Kopie der Daten im Cache ab. Die im Cache neu abgelegten Daten haben wieder die gleiche Gültigkeitsdauer von 24 Stunden oder 1440 Minuten. Die Caching-Methode WPS_IN ignoriert den Parameter *Proxy*, die Methode NONE ignoriert zusätzlich den Parameter *maxCachingTime*.

```
Beispiel:
  [Caching]
  Method = PROXY
  Proxy = 127.0.0.1:3128
  maxCachingTime = 1440
  LogFile = M:\zooWPS\Logs
  Cache = M:\zooWPS\Cache
```

Nach Anpassung des Zoo-Kernels werden alle WFS-Anfragen automatisch in einer Datei mit der Bezeichnung *Access.log* auf dem WPS-Server protokolliert. Die protokollierten Werte können der folgenden Aufstellung entnommen werden.

- Datum und Uhrzeit der Anfrage
- Zeitpunkt der Anfrage im Unix-Format
- Zugriffszeit in Millisekunden; Zeitspanne zwischen dem Absenden der Anfrage und dem Erhalt der WFS-Daten
- Größe der Antwort in Byte
- Status der Antwort
- URL mit Anfrage-String

Die nächste Auflistung zeigt einen Ausschnitt eines Datensatzes aus der neuen Protokolldatei. Die Reihenfolge der sechs Werte entspricht der oben angegebenen Aufstellung; die einzelnen Werte sind durch das Zeichen | getrennt.

29.06.2013 00:51:59 | 1372459919 | 10 | 497532 | PROXY_127.0.0.1:3128 | http://...

Aus der Protokolldatei lässt sich die Übertragungszeit für den Transport der WFS-Daten ablesen. Dem Datensatz ist zu entnehmen das die Daten aus dem Proxy-Cache geliefert wurden (PROXY_127.0.0.1:3128); der Transport der Daten vom Proxy zum WPS betrug 10 Millisekunden.

5.2 Konzeption der Performance- und Lasttests

Dieser Abschnitt beschreibt wie der Einfluss der durchgeführten WPS-Anpassungen auf das Antwortzeitverhalten gemessen werden kann. Ausgehend von den in Kapitel 4 vorgestellten Überlegungen zu Performance- und Lasttests werden die folgenden Testszenarien entwickelt.

Es werden die Antwortzeiten für die drei WPS-Konfigurationen

- internes WPS-Caching,
- Proxy-Caching,
- kein Caching

untersucht und gegenübergestellt. Dabei wird die Anzahl der konkurrierenden (parallelen) Nutzerzugriffe zunehmend vergrößert und die Komplexität der Analysen kontinuierlich erhöht. Die Messungen werden automatisiert und nachvollziehbar durchgeführt. Um mögliche Fehler der eingesetzten Messwerkzeuge zu vermeiden, werden zwei unterschiedliche Werkzeuge eingesetzt.

Für den Test werden die folgenden vier Anwendungsfälle herangezogen.

(1) Puffer-Analyse mit lokalen WFS-Daten
(2) Puffer-Analyse mit entfernten WFS-Daten
(3) Verschneidung lokalen WFS-Daten
(4) Verschneidung entfernter WFS-Daten

Der Test wird nach folgendem Ablauf durchgeführt:
Der Gesamttest für einen Anwendungsfall besteht aus drei Testserien, eine Serie für jede WPS-Konfiguration. Jede Testserie besteht aus mehreren Testläufen. Die Anzahl der Testläufe ist abhängig von der Anzahl der Puffer- oder Verschneidungsberechnungen sowie der Anzahl der konkurrierenden Anfragen. Jeder Testlauf besteht wiederum aus 100 Einzelanfragen. Aus den Ergebnissen der Einzelanfragen wird das arithmetische Mittel gebildet. Dies ergibt das Ergebnis eines Testlaufs. Die Ergebnisse der Testläufe werden untersucht und bewertet. Die Einzelergebnisse dienen der Kontrolle.

Zur Verdeutlichung wird der Ablauf für Anwendungsfall (1) vorgestellt. Der Test wird mit folgenden Werten durchgeführt:

Tabelle 3.1: Testparameter

Parameter / Bezeichnung	Wert(e)
WFS-Datensatz:	Naturschutzgebiete in Rheinland-Pfalz
WPS-Funktion:	Puffer
Anzahl Puffer:	10, 20, 50, 100, 200
Anzahl Einzelabfragen pro Testlauf:	100
Anzahl konkurrierender Anfragen:	1, 2, 5, 10, 20, 50, 100

Diese Parameter führen zur folgenden Anzahl von Testserien und -läufen für Anwendungsfall (1):

Tabelle 3.2: Testserien, Testläufe und Einzeltests

Bezeichnung	Anzahl
Anzahl Testserien (Proxy-Caching, Input-Caching, kein Caching):	3
Anzahl Testläufe [Anzahl Puffer * Anzahl konkurrierender Anfragen]:	35
Anzahl Einzeltests je Serie [Anzahl Testläufe * Anzahl Einzelabfragen]:	3500
Gesamtzahl aller Einzeltests:	10500

Dies führt zu 10.500 Einzelmessungen für eine Testserie. Alle Messungen werden mit WebPerformanceBenchmark durchgeführt. Es werden 2 Serien gemessen. Mit Apache Benchmark werden zum Vergleich zusätzliche Messungen durchgeführt.

5.3 Ergebnisse der Performance- und Lasttests

Bei der Durchführung des Tests wird aus zeitlichen Gründen der Schwerpunkt auf die Untersuchung von Puffer-Anfragen externer WFS-Daten gesetzt. Die graphisch aufbereiteten Ergebnisse beider Messungen können dem Anhang entnommen werden. Zwei Messungen werden näher vorgestellt. Abbildung 5.1 zeigt die graphische Aufbereitung der Messergebnisse für die Berechnung von 20 Puffern. Auf der horizontalen Achse ist die Anzahl der konkurrierenden (parallelen) Zugriffe dargestellt. Auf der vertikalen Achse sind die gemessenen Werte in Millisekunden aufgetragen. Es fällt auf, dass die Performance für alle drei Konfiguration von einem bis 20 Benutzern gleichmäßig ansteigt. Bei 50 Anwendern ist die Performance des Web-Cache am schlechtesten. Dieses etwas unerwartete Verhalten konnte auch mit anderen Messparametern vereinzelt beobachtet werden.

Abbildung 5.2 zeigt die graphische Repräsentation das Messergebnis für 200 Puffer. Berechnungen für 50 Puffer ohne Cache waren nicht möglich, da der Rechner an seine Leistungsgrenze gestoßen ist. Das Ergebnis ist offensichtlich einfacher zu interpretieren als das erste Beispiel. Der Graphen der beiden Caching-Konfigurationen verlaufen bis 20 Nutzer fast identisch. Bei 50 Nutzern ist der WPS-Cache etwas schneller als der Web-Cache. Beide Caches weisen eine stetig ansteigende bessere Performance auf als die Konfiguration ohne Cache.

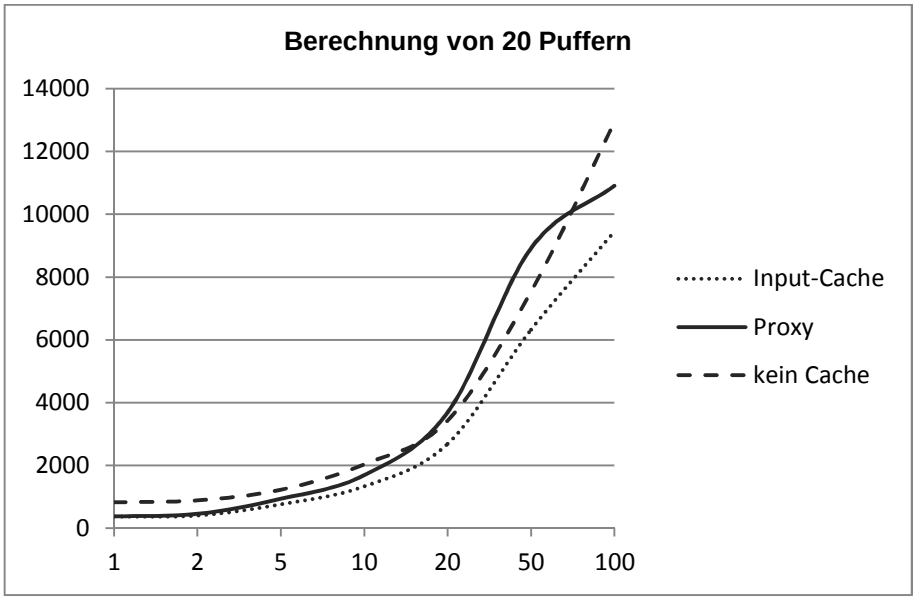


Abbildung 5.1: Messergebnis für 20 Puffer-Berechnungen

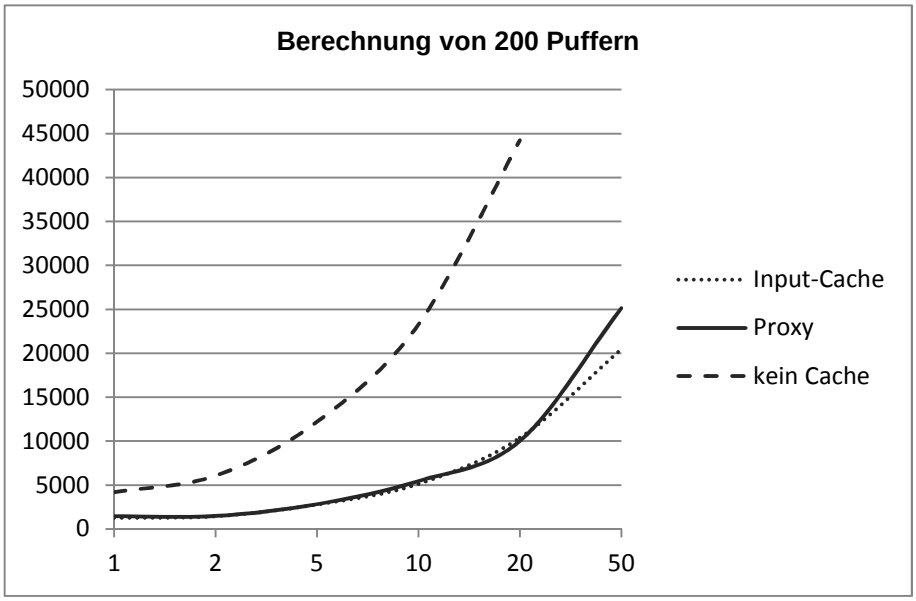


Abbildung 5.2: Messergebnis für 200 Puffer-Berechnungen

Die Werkzeuge Apache Benchmark und WebPerformanceBenchmark liefern in etwa identische Messergebnisse. Die festgestellten geringen Differenzen können für diese Untersuchung vernachlässigt werden. Allerdings hat sich Apache Benchmark als

weniger stabil erwiesen. Der Anteil der fehlerhaften Messungen oder Abstürze war deutlich höher als beim Einsatz von WebPerformanceBenchmark.

Für die Messungen wird eine WPS-Anfrage benutzt, bei der die Anzahl der zu berechnenden Puffer und somit die Komplexität der Analyse mit dem Parameter *maxFeature* gesteuert wird. Die WPS-Anfrage kann dem Anhang entnommen werden.

Die Messergebnisse für das Proxy-Caching sind nahezu identisch mit den Zugriffszeiten, die Squid in der internen Datei Access.log protokolliert. Die Werte der Squid-Protokolldatei sind geringfügig kleiner als die vom WPS gemessenen Werte.

Im abschließenden Kapitel werden die Ergebnisse diskutiert.

6 Diskussion und Schlussfolgerung

In diesem Kapitel werden die im vorangegangenen Kapitel vorgestellten Ergebnisse diskutiert. Es wird untersucht, ob die in der Einleitung aufgestellte Hypothese bestätigt oder falsifiziert wird. Ergebnisse aus benachbarten Arbeiten werden in die Untersuchung einbezogen. Darüber hinaus werden Hinweise zu Modifikationen oder weiteren Forschungsbedarfen gegeben. Das Kapitel endet mit einer zusammenfassenden Schlussfolgerung.

6.1 Diskussion

In dieser Arbeit wurden Konzepte für Web-Caching-Techniken für WPS entwickelt und umgesetzt. Im Rahmen einer Studie wurden die Auswirkungen von Web-Caching auf das Antwortzeitverhalten untersucht. Es wurde eine Referenzarchitektur mit drei unterschiedlichen WPS-Konfigurationen, mit einem Proxy-Cache, mit einem integrierten Cache und ohne Cache, aufgebaut. Zur Durchführung der Performance-Messungen wurde eine auf .NET Framework basierende Konsolen-Anwendung entwickelt. Zusätzlich wurden Messungen mit dem Werkzeug Apache Benchmark durchgeführt. Nachfolgend werden die Erkenntnisse und Ergebnisse mithilfe der Forschungsfragen bewertet.

Wie können Web-Caching-Techniken zum Zwischenspeichern eingesetzt werden?
Im Rahmen dieser Arbeit werden verschiedene WPS-Caching-Varianten entwickelt und gegenübergestellt. Es wird zwischen einem Input-Cache und einem Output-Cache unterschieden. Im Input-Cache werden die zur Durchführung der WPS-benötigten WFS-Daten gepuffert. Im Output-Cache werden die Analyse-Ergebnisse der WPS-Anfrage gepuffert.

Es werden drei Caching-Lösungen zum Speichern von WFS-Daten vorgestellt; zwei Web-Caches sowie ein in den WPS integrierter Cache. Bei beiden Web-Caching-Varianten wird der HTTP-Verkehr zwischen WPS und WFS über einen Proxy-Cache umgeleitet. Bei der ersten Variante erfolgt die Umleitung des Netzwerkverkehrs durch Konfiguration oder durch softwaretechnische Anpassung des WPS. Im Rahmen der Arbeit wird der generische WPS des Zoo-Projektes zur Umleitung des HTTP-Verkehrs softwaretechnisch angepasst. Die Ergebnisse der Arbeit zeigen, dass der Einsatz von Web-Caching-Techniken zum Zwischenspeichern von WFS-Daten grundsätzlich möglich ist.

Allerdings wird hierbei die angestrebte Zustandslosigkeit teilweise aufgegeben, da die Daten auch nach Beenden der WPS-Analyse im Proxy-Cache zur Verfügung stehen. Das Zeitfenster, in dem Cache-Objekte vom Proxy als gültig angesehen werden, lässt sich mittels Parameter im Proxy konfigurieren. Ist das Cache-Objekt älter als der angegebene Wert, so werden die Daten nicht aus dem Cache geladen, sondern vom Ursprungsserver geholt. Die Herstellung der völligen Zustandslosigkeit und Caching schließen sich offensichtlich aus. Wählt man eine kurze Gültigkeitsdauer der Cache-Objekte, so nähert man sich der völligen Zustandslosigkeit. Nach Meinung des Autors sollte die (vollständige) Zustandslosigkeit bei einer Abfolge von WPS-Einzelprozessen, die zu einem Analyseergebnis führen, zugunsten der Performance aufgegeben werden.

Als zweite Variante zum Zwischenspeichern von WFS-Daten kann ein transparenter Proxy betrieben werden. Hierbei wird der Netzwerkverkehr hardwaremäßig über einen Router umgeleitet; softwaretechnische Anpassungen des WPS sind nicht erforderlich. Diese Variante wird theoretisch beschrieben jedoch nicht praktisch erprobt. Die technische Umsetzung sollte problemlos möglich sein.

Zum Speichern von Ergebnisdaten (Output-Cache) lassen sich Proxy-Server nicht einsetzen, da sie nach Wessels (2004) keine Anfragen via HTTP-Post zwischenspeichern können. Diese Variante wird im Rahmen dieser Arbeit nicht weiter untersucht. Es wird jedoch ein in die WPS-Anwendungshierarchie integrierter Cache vorgestellt. Die praktische Erprobung des vorgestellten Konzeptes ggf. auch in Kombination mit dem einer Web-Caching-Variante könnte im Rahmen zukünftiger Vorhaben untersucht werden.

Welche Auswirkungen hat der Einsatz von Web-Caching-Techniken auf das Antwortzeitverhalten?

Caching ist nur dann sinnvoll, wenn hiermit eine Steigerung der Performance oder eine Reduzierung der Netzwerklast verbunden ist. Im Rahmen der im fünften Kapitel vorgestellten Studie wurden Performance- und Lasttests für verschiedene WPS-Anfragen durchgeführt. Mit den Messungen sollte die Performance einer Web-Caching-Lösung festgestellt werden. Zum Vergleich wurden Messungen für Konfigurationen ohne Caching sowie für einen in den WPS integrierten Cache durchgeführt. Für alle Tests wurde die elementare GIS-Funktionen Puffer benutzt. Die Anzahl der parallel durchgeführten WPS-Prozesse und die Komplexität der Anfragen wurden zunehmend bis zur Belastungsgrenze der CPU gesteigert. Die Komplexität der Anfragen wurde durch Erhöhung der zu berechnenden Puffer-Operationen gesteigert.

Die Messergebnisse der Referenzumgebung zeigen, dass Web-Caching der WFS-Daten nicht automatisch zu einer verbesserten Performance führt. Bei hoher Komplexität der Analysen oder einer großen Anzahl paralleler Prozesse konnten durch Caching deutliche Performancevorteile erzielt werden. Bei einfachen Analysen und kleinen Datenvolumen zeigten sich der durch Caching bedingte Performancevorteile erst bei mehr als 50 parallelen Zugriffen. Bei wenigen parallelen Zugriffen und einer damit einhergehenden gering belasteten CPU waren die Unterschiede zwar messbar aber nicht signifikant. In sehr wenigen Konstellationen war die Konfiguration ohne Caching performanter als die Web-Caching-Variante.

Die Referenzumgebung lässt keine Aussagen zum Wirkungsgrad der Caching-Szenarien zu. Ist die Anzahl der Cache-Misses (kein Treffer) im Vergleich zur Anzahl der Cache-Hits (Treffer) zu hoch, können Caches ineffektiv werden. Die Effek-

tivität ist vom Nutzerverhalten abhängig. Eine Erhebung des Nutzerverhaltens ist zu empfehlen und könnte Bestandteil zukünftiger Untersuchungen werden.

Die Vorgehensweise, mit der die Auswirkungen des Web-Caches untersucht und durchgeführt werden, wird als sinnvoll angesehen. Als wünschenswert werden weitere Messanordnungen mit anderen Datenbeständen und Analysefunktionen angesehen. Der Aufbau einer produktionsnahen Infrastruktur wäre ebenfalls empfehlenswert.

Nach den in der Referenzumgebung festgestellten Ergebnissen und den obigen Ausführungen konnte die in der Einleitung formulierte Hypothese nicht in Gänze bestätigt werden. Die Ergebnisse zeigen, dass Web-Caching bei komplexen Berechnungen in Verbindung mit vielen parallelen Prozessen zu einer deutlich besseren Performance führt. Bei einfachen Analysen, geringen Datenvolumen und wenig Nutzern sind die Vorteile des Web-Caches eher gering oder gar nicht vorhanden. In Einzelfällen wurde eine geringe Verschlechterung der Performance festgestellt. Web-Caching führt nicht immer zu einer besseren Performance. Die Caching-Strategie sollte sich auch am Nutzerverhalten orientieren. Eine mögliche Übertragung der festgestellten Ergebnisse auf andere Messanordnungen könnte zukünftig untersucht werden.

Durch Verwendung des Web Service Resource Framework (WSRF) der Organization for the Advancement of Structured Information Standards (OASIS) lässt sich nach Keens (2006) ein redundanter Datentransport vermeiden und die Performance verbessern. Mehrfach benötigte Datensätze werden bereits vor Ausführung des WPS auf den Server geladen. Kurze Zeit nach Beenden des WPS-Prozesses werden die hochgeladenen Daten wieder gelöscht. Mit der Vorgehensweise wird die angestrebte Zustandslosigkeit zwar aufgegeben, jedoch nur für eine kurze Zeitspanne. Ein ähnliches Verhalten wird mit dem vorgestellten Proxy-Cache (Input-Cache) erreicht, wenn die Gültigkeitsdauer im Proxy entsprechend kurz eingestellt wird. Mit der vorgestellten und implementierten Lösung kann jedoch nur eine für alle Anfragen geltende Zeitdauer im Proxy eingestellt werden. Im Rahmen weiterer Forschungsvorhaben könnte eine auf Proxy-Caching basierende Variante untersucht werden, mit der die Gültigkeitsdauer der Caching-Objekte für jede Anfrage oder für jede Abfolge von Einzelanfragen spezifisch eingestellt werden kann.

Neben der Expertenmeinung, ist die Rückkopplung mit den Anwendern wünschenswert. Aus dem Nutzerverhalten lässt sich die Notwendigkeit für WPS-Caching aus

der fachlichen Sicht herleiten. Die fachlichen Anforderungen sollten auch für ein mögliches Output-Caching erhoben werden. Eine mögliche Frage lautet: Mit welcher Häufigkeit sind identische WPS-Anfragen oder identische Abfolgen von WPS-Einzelanfragen in Ihrer Organisation zu erwarten? Auch wenn die Zustandslosigkeit offensichtlich nicht mit dem Aufbau eines Output-Caches vereinbar ist, sollten die Anwender mit dieser oder ähnlichen Fragen konfrontiert werden. Die Ergebnisse der Erhebung könnten zu weiteren Untersuchungen führen.

Für die Einbindung von Proxy-Caches (Input-Caches) ist es Sicht dieser Arbeit wünschenswert, wenn Web Processing Services (zukünftig) über eine konfigurierbare Möglichkeit zur Umleitung des HTTP-Verkehrs verfügen.

Die Arbeit beschränkt sich auf die Untersuchung von WPS-Anfragen mit referenzierten Vektordaten. Das vorgestellte Input-Caching sollte grundsätzlich auch mit referenzierten Rasterdaten möglich sein und könnte in weiteren Vorhaben untersucht werden.

Große Organisationen mit vielen Anwendern benötigen zukünftig leistungsfähige WPS-Installationen für die ein Proxy-Cache nicht ausreichend sein könnte. Zur Lastverteilung und zur Reduzierung von Bandbreiten könnten weitere Proxys notwendig werden. Nach Dithardt (2007) können mehrere Squid-Proxys im Verbund betrieben werden. Im Rahmen weiterer Vorhaben könnten grundlegende Fragestellungen für einen Proxy-Verbund in Verbindung mit WPS untersucht und Implementierungen erprobt werden.

Die Performance- und Lastmessungen ließen sich mit dem Werkzeug *WebBenchmarkPerformance* einfach und schnell durchführen. Die Automation gesamter Testserien wurde über zusätzliche Batchprogramme hergestellt. Diese Möglichkeit könnte zukünftig in die Anwendung integriert werden.

6.2 Schlussfolgerung

In dieser Arbeit wurden Caching-Techniken für WPS vorgestellt. Durch Zwischenspeichern der WFS-Ausgangsdaten (Input-Caching) lässt sich das mehrfache Anfragen der Ursprungsserver innerhalb einer Kette von WPS-Einzelprozessen vermeiden. Der Aufwand beschränkt sich auf die softwaretechnische Anpassung oder Konfigura-

tion des WPS zum Umleiten des Netzwerkverkehrs sowie die Konfiguration des Proxy-Servers. Auf die von der OGC geforderte Zustandslosigkeit sollte zugunsten der Performance bewusst verzichtet werden.

Die Arbeit kann das komplexe Thema sicherlich nicht abschließend behandeln.

Web Processing Services und Web-Caching bieten vielfältigen Raum für Forschungsansätze. In dieser Arbeit wurden beide Gebiete miteinander verbunden. Es steht zu hoffen, dass sich auch in Zukunft weitere interessante Forschungsansätze in beiden Themenfeldern ergeben. Die Arbeit hat hierfür Hinweise gegeben.

7 Referenzen

Apache Tomcat (URL 2013): Apache Tomcat. Zugriff 20.05.2013. http://tomcat.apache.org/

Badach, Anatol; Rieger, Sebastian; Schmauch, Matthias (2003): Web-Technologien – Architekturen, Konzepte, Trends. Carl Hanser Verlag München Wien

Balzert, Helmut (2000): Lehrbuch der Software-Technik - Software-Entwicklung. Zweite Auflage. Heidelberg, Berlin : Spektrum Akademischer Verlag

Béjar, R. et al (2012): Improving the Performance of the WPS in the EuroGeoSource System: an ETL Process to Maintain a WFS Data Cache. Universidad Zaragoza, Zaragoza, Spain

Bison (URL 2013): Bison - GNU parser generator. Zugriff 02.03.2013. http://www.gnu.org/software/bison/

Brauner, Johannes (2008): Web Processing Service Schnittstelle für GIS-Funktionalitäten; Diplomarbeit; Universität Münster

CGIc (URL 2013): cgic: an ANSI C library for CGI Programming. Zugriff 10.02.2013. http://www.boutell.com/cgic/

cURL (URL 2013): curl groks URLs. Zugriff 02.02.2013. http://curl.haxx.se/

Cockburn, Alistair (2003): Use Cases effektiv erstellen. Mitp-Verlag, Heidelberg

Dithardt, Dirk (2007): Squid. Administrationshandbuch zum Proxyserver. dpunkt.verlag GmbH Heidelberg

FastCGI (URL 2013): FastCGI. Zugriff 02.02.2013. http://www.fastcgi.com/drupal/

Flex (URL 2013): flex: The Fast Lexical Analyzer. Zugriff 01.03.2013. http://flex.sourceforge.net/

Foerster, Theodor und Schäffer, Bastian (2007): A Client for Distributed Geo-processing on the Web; in Web and Wireless Geographical Information Systems - 7th International Symposium, W2GIS 2007, Cardiff, UK,

GeoServer (URL 2013): GeoServer: Zugriff: 10.05.2013. http://geoserver.org/display/GEOS/Welcome

GNU (URL 2013): GNU – Freie-Software-Definition. Zugriff 02.05.2013. http://www.gnu.org/philosophy//free-sw.html

Hansen, Hans Robert und Neumann Gustav (2005): Wirtschaftsinformatik 2 - Informationstechnik. 9. Auflage. Lucius & Lucius Verlagsgesellschaft mbH, Stuttgart

Hansen, Hans Robert und Neumann, Gustav (2009): Wirtschaftsinformatik 1 - Grundlagen und Anwendungen. 10. Auflage. Lucius & Lucius Verlagsgesellschaft mbH, Stuttgart

HTTP (URL 2013): Hypertext Transfer Protocol – HTTP/1.1. Zugriff 10.05.2013. http://tools.ietf.org/html/rfc2616

Keens, Steven (2006): OWS-4 WPS IPR: Discussions, findings, and use of WPS in OWS-4 / Open Geospatial Consortium Inc. December 2006 (OGC 06-182); Forschungsbericht

Kuhn, Daniel und Raith, Michael (2013): Performante Webanwendungen - Cli-ent- und serverseitige Techniken zur Performance-Optimierung. dpunkt.verlag GmbH Heidelberg

Libxml (URL 2013): The XML C parser and toolkit of Gnome. Zugriff 03.05.2013. http://xmlsoft.org/

Loechel, Alexander und Schmid, Stephan (2012): Verschiedene Caching-Techniken für Map Services. In: Strobl, Josef; Blaschke, Thomas und Griesebner, Gerald (Hrsg) (2012): Angewandte Geoinformatik 2012. Herbert Wichmann Verlag

Molyneaux, Ian (2009): The Art of Application Performance Testing. O'Reilly Media Inc. Sebastopol, CA, 95472

Neteler, Markus und Mitasova, Helena (2008): Open Source GIS – A GRASS GIS Approach, Third Edition. Springer New York

Open Geospatial Consortium Inc. (Hrsg.) (2007): OpenGIS Web Processing Service. 1.0.0. August 2007

Open Geospatial Consortium Inc. (Hrsg.) (2002): OpenGIS Abstract Specification Topic 12: OpenGIS Service Architecture. 4.3. January 2002

OpenSSL (URL 2013): OpenSSL Cryptography and SSL/TLS Toolkit. Zugriff 10.05.2013. http://www.openssl.org/

Papazoglou, Mike P. und Heuvel, Willem-Jan (2007): Service oriented architectures: approaches, technologies and research issues. In: The VLDB Journal 16 (2007)

Python (URL 2013): Python Programming Language – Official Website. Zugriff 03.04.2013. http://www.python.org/

Rupp, Chris; Queins, Stefan; Zengler, Barbara (2007): UML 2 glasklar - Praxiswissen für die UML-Modellierung, Carl Hanser Verlag München Wien

Schäffer, Bastian (2007): Integrated Web Geoprocessing Workflow Composition and Deployment, Institute for Geoinformatics, University of Münster, Diplomarbeit,

SharpDevelop (URL 2013): ic#code. Zugriff 20.05.2013. http://www.icsharpcode.net/opensource/sd/

Schmeh, Klaus (2013): Kryptografie - Verfahren, Protokolle, Infrastrukturen. 5. aktualisierte Auflage, dpunkt.verlag GmbH, Heidelberg

Spillner, Andreas; Linz, Tilo (2012): Basiswissen Softwaretest; 5. überarbeitete und aktualisierte Auflage 2012, dpunkt.verlag GmbH Heidelberg

Squid (URL-1 2013): squid-cache.org. Zugriff 12.03.2013. http://www.squid-cache.org/

Squid (URL-2 2013): Squid for Windows. Zugriff 12.04.2013. http://squid.acmeconsulting.it/

Visual Studio (URL 2013): Visual Studio 2010 Express. Zugriff 10.04.2013. http://www.microsoft.com/visualstudio/deu/downloads#d-2010-express

Wessels, Duane (2001): Web-Caching, O'Reilly & Associates, Inc.

Wessels, Duane (2004): Squid - The Definitive Guide. O'Reilly Media, Inc.

Wikipedia (URL 2013): Wikipedia – Die freie Enzyklopädie. Zugriff 10.04.2013. http://de.wikipedia.org/wiki/Squid

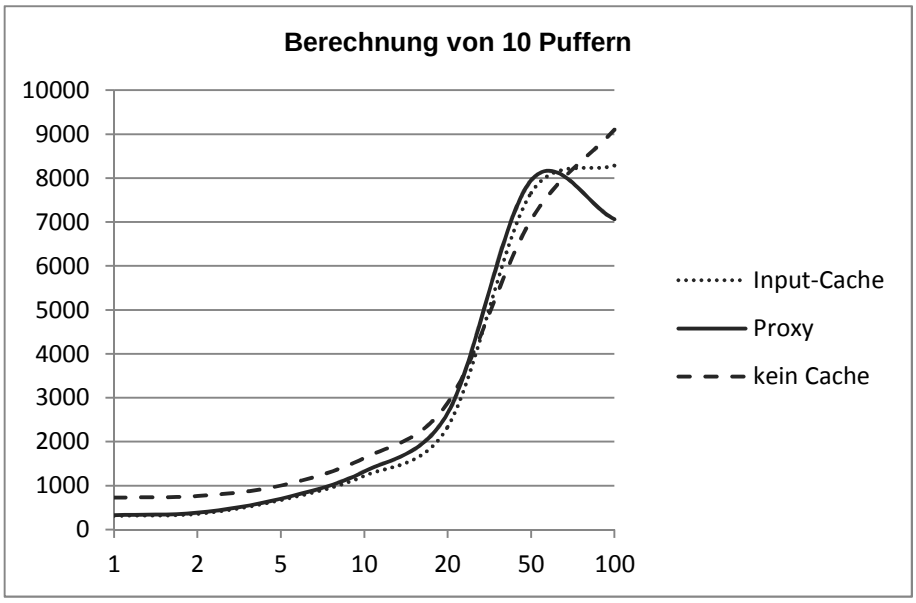
Zoo (URL-1 2013): Zoo – Open WPS Platform. Zugriff 16.05.2013. http://www.zoo-project.org/

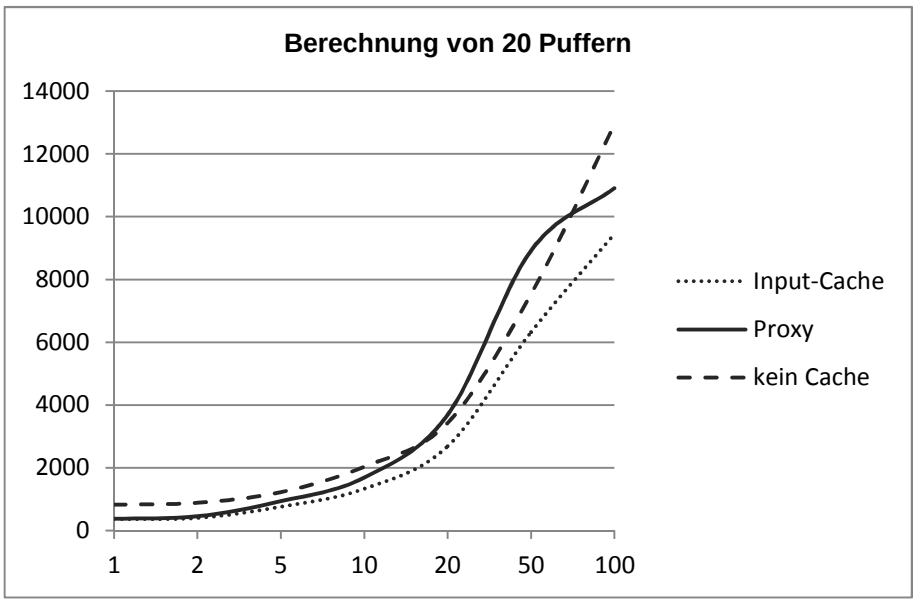
Zoo (URL-2 2013): Prerequisites. Zugriff 11.05.2013. http://zoo-project.org/docs/kernel/install-prerequisites.html#id1

Anhang

Anhang 1

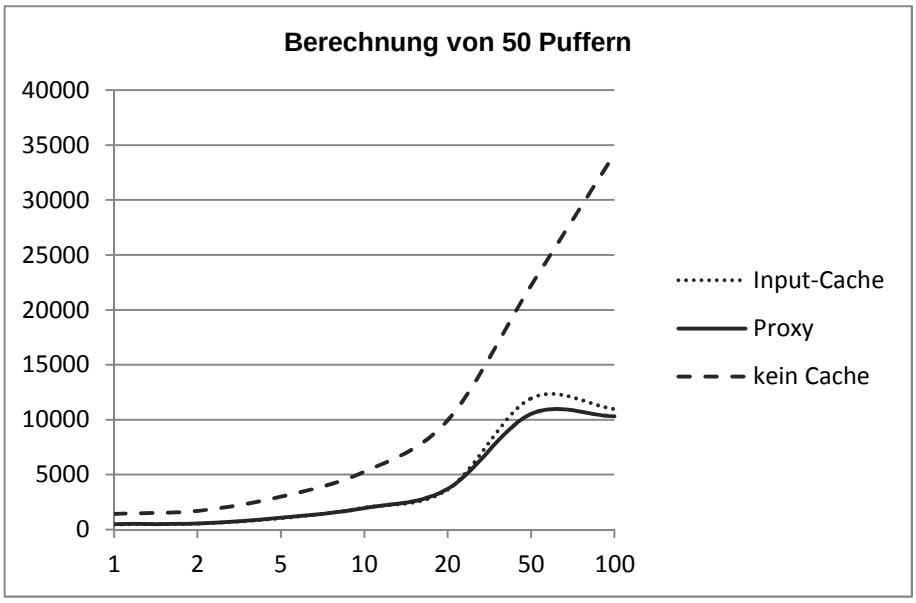
Messung 1: 29.05.2013

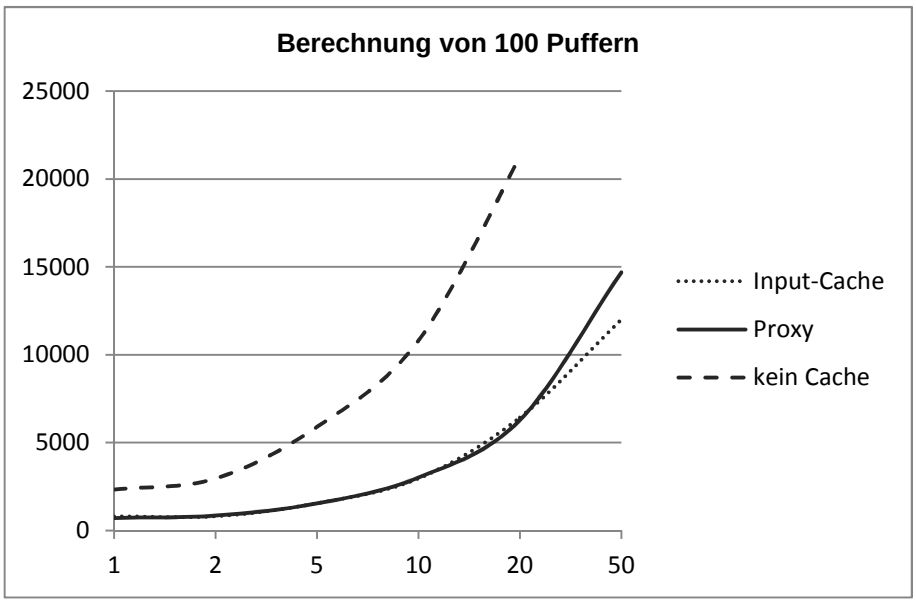




Anhang 1

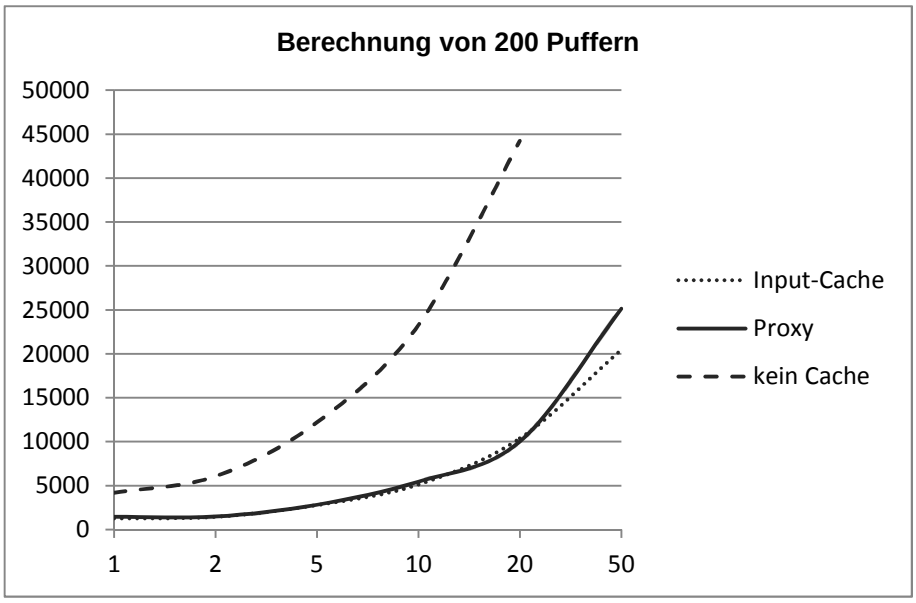
Messung 1: 29.05.2013





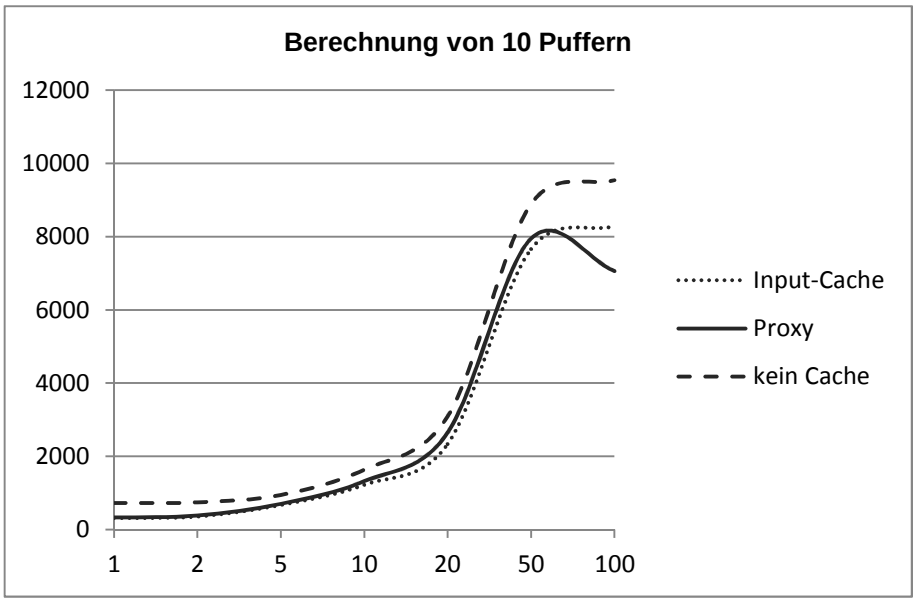
Anhang 1

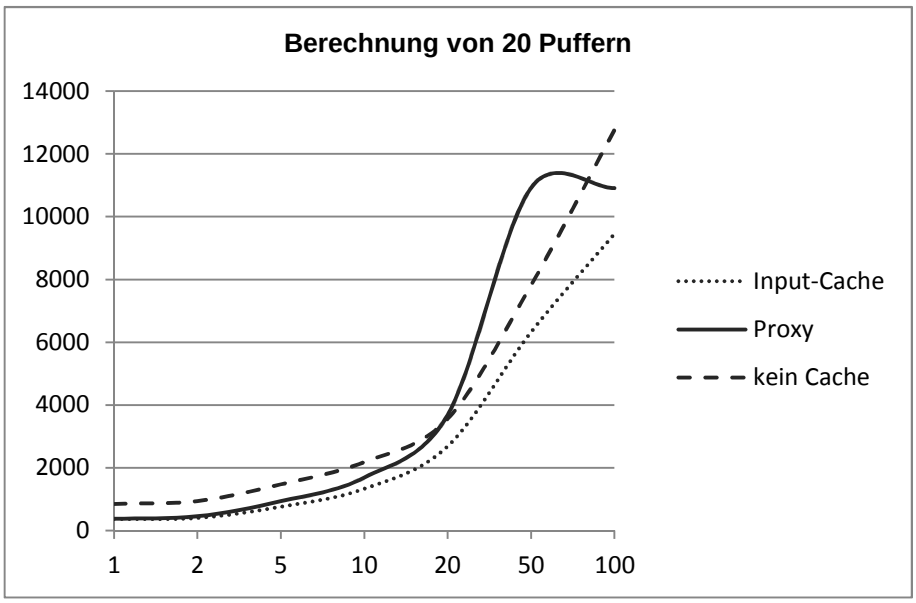
Messung 1: 29.05.2013



Anhang 1

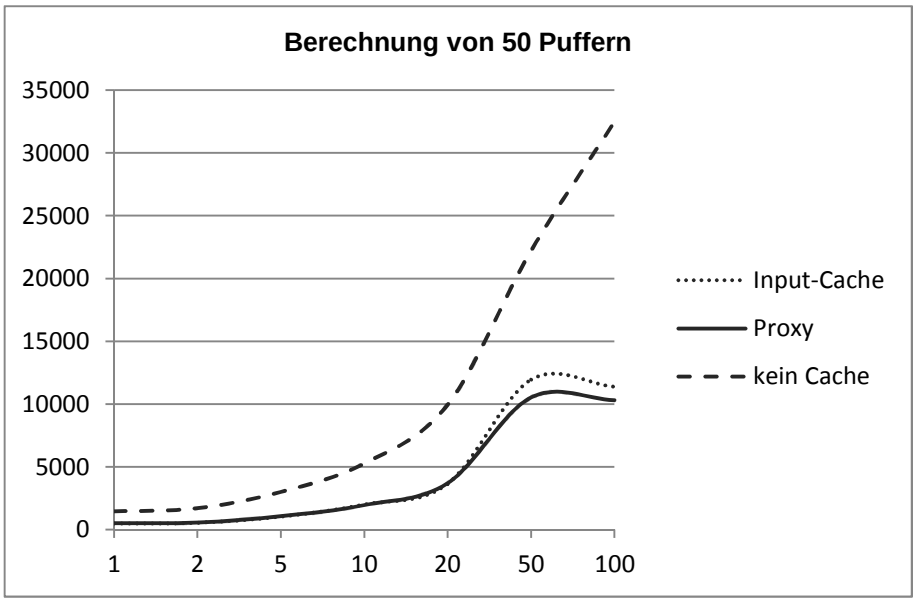
Messung 1: 31.05.2013

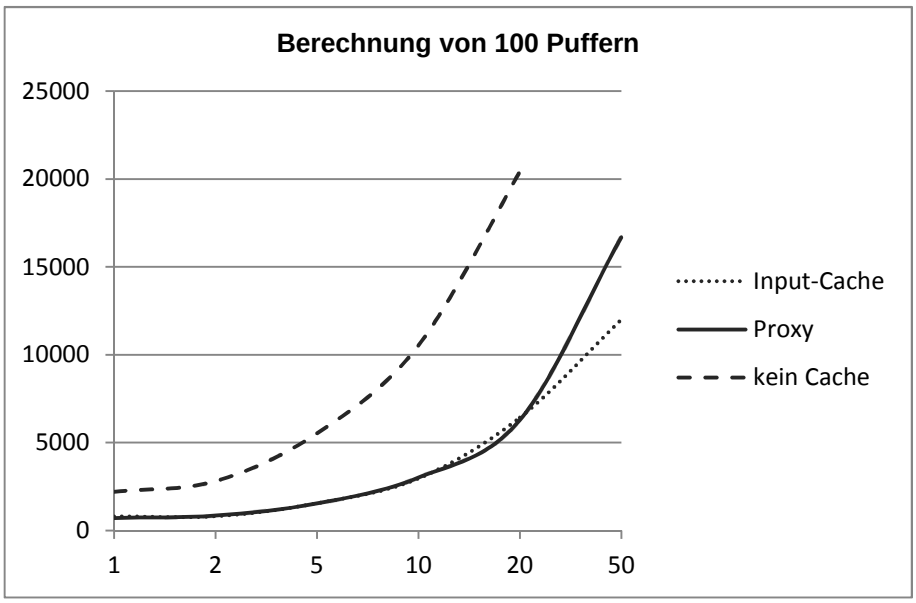




Anhang 1

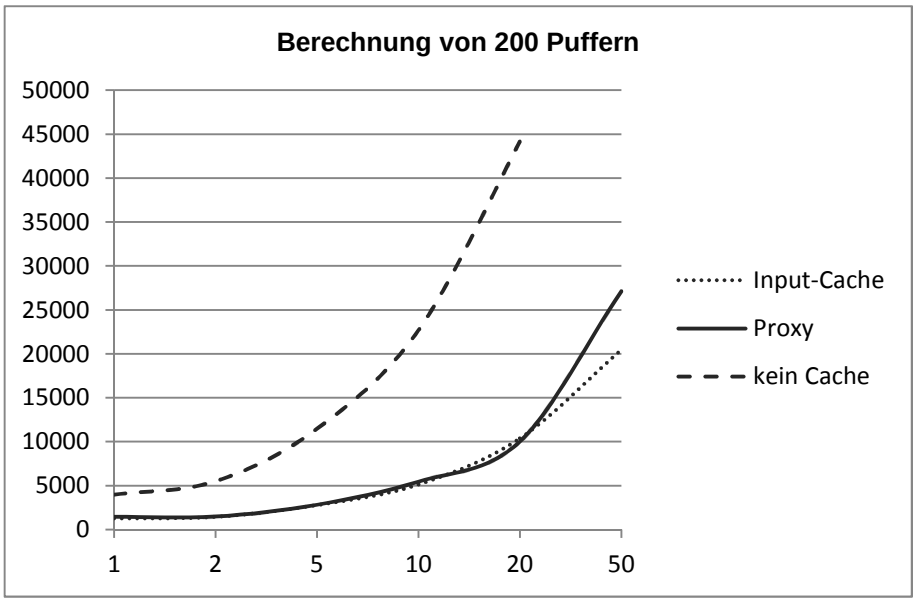
Messung 1: 31.05.2013





Anhang 1

Messung 1: 31.05.2013



Anhang 2

Bespiel für eine WPS-Anfrage mit 50 Puffern, die für die Performancemessungen herangezogen wurde.

```
<wps:Execute service="WPS" version="1.0.0"
xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_request.xsd">
  <ows:Identifier>BufferPy</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>InputPolygon</ows:Identifier>
      <ows:Title>zooWPS test</ows:Title>
      <wps:Reference mimeType="text/xml"
xlink:href="http://map1.naturschutz.rlp.de/service_lanis/mod_wfs/wfs
_getmap.php?mapfile=naturschutzgebiet&amp;service=WFS&amp;version=1.
0.0&amp;Request=GetFeature&amp;Typename=Naturschutzgebiet&amp;maxFea
tures=50&amp;SRS=EPSG:25832&amp;Outputformat=GML3" />
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="text/xml; subtype=gml/3.1.1">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

Anhang 3

Die beigefügte CD enthält die Quellcodes und eine elektronische Fassung dieses Dokuments.